

# An Action Interface Manager for ROSPlan

Stefan-Octavian Bezrucav,<sup>1</sup> Gerard Canal,<sup>2</sup> Michael Cashmore,<sup>3</sup> Burkhard Corves<sup>1</sup>

<sup>1</sup> Institute of Mechanism Theory, Machine Dynamics and Robotics, RWTH Aachen University

<sup>2</sup> Department of Informatics, King's College London

<sup>3</sup> Department of Computer and Information Sciences, University of Strathclyde

## Abstract

Task planning and task execution are two high-level robot control modules that often are working with representations of the scenario at different levels of abstraction. Thus, a further mapping module is required to connect the abstract planned actions to the robot-specific algorithms that must be called in order to execute these actions.

We present a novel implementation of such a module that allows a user to define this mapping for all actions through a single configuration file. This greatly reduces the amount of effort that is required to integrate an automated planner with a robotic platform.

This module has been integrated as an Action Interface of the automated task planning framework ROSPlan, and includes a Graphical User Interface through which the configuration file can be easily generated and updated. The use of the interface is demonstrated in two scenarios: with robot actors possessing only a single action, and a more complex scenario with multiple agents and types of actions.

## 1 Introduction

Task planning is the process of determining the actions that must be executed, the order in which they should be dispatched, and the actors that should carry them out in order to achieve the set goals. This process is usually done on an abstract and simplified representation of the considered scenario. A complete representation of the environment, the actors, and their possible actions is not feasible for the planning process as the solvers are not capable of handling that much information. On the other hand, the execution of the planned actions, whether in a real or simulated scenario, must consider these features and interactions. Thus, a flexible, but yet robust mapping between the abstract planned actions and their execution is required.

In this paper, we present the description of such a mapping interface for actions of plans generated with automated planning approaches and written in the Planning Domain Definition Language (PDDL) (McDermott et al. 1998). Furthermore, we describe how it has been integrated in the state-of-the-art task planning framework ROSPlan (Cashmore et al. 2015). The strength of this interface is that the mapping can be completely defined through a single configuration file, reducing the need for a hard-coded interface between PDDL action and action implementation. Moreover, this configura-

tion can be generated through the use of a Graphical User Interface.

In the next section we provide some background on the Robot Operating System (Quigley et al. 2009) (ROS) that is necessary to understand the particulars of interfacing actions. In Section 3, we analyze how different frameworks handle this mapping process in simulated or real robotic applications working with ROS. In Sections 4 and 5 we present the implementation of our action interface. The aim of this interface is to reduce the effort required to interface PDDL actions with a ROS action implementation, while maintaining flexibility in plan execution. This benefit is demonstrated in two simulated robotic scenarios described in Section 6. The Graphical User Interface is presented in Section 7.

## 2 Background

The Robot Operating System (ROS) (Quigley et al. 2009) is one of the standard meta-operating systems used in the robotic community. It sustains the integration of different algorithms in so-called ROS *nodes* and provides different structures for communicating data between these nodes. The basic communication structures are the *topics*, over which messages are routed according to the publish/subscribe paradigm, and the *services* that are based on the request/reply paradigm (ROS.org 2021). In addition, *actionlibs* are a complex communication strategy based on topics, that provide long-term non-blocking request/feedback/reply structures.

The robotic software realising the action execution considered in this paper is implemented in ROS as nodes, each of which offers either a service or actionlib interface. Through these communication structures, the algorithms integrated in each ROS node can be called or triggered and responses can be received and further processed. If it is required to call or trigger multiple ROS nodes in a given sequence, this can be realized through a Moore Finite State Machine (FSM) (Moore 1956), whose states represent such calls or triggers and whose transitions are dependent on the obtained responses.

Different task planning frameworks integrated in ROS use the above presented communication structures to trigger the concrete execution of the planned actions on each involved actor. The ROSPlan framework (Cashmore et al. 2015) is composed itself of different ROS nodes. The *Plan Parsing*

node is responsible for parsing the generated plan file in an internal C++ representation and for sending it to the *Dispatcher* node. The latter selects actions to be executed and sends them over a topic to *Action Interface* nodes that interpret the command and then interface with the corresponding robotic algorithms through services or actionlibs. Once the respective executions have finished, the interface passes this information to the dispatcher in form of an action feedback message. The dispatcher then decides how to continue.

Action interfaces are typically implemented per robotic algorithm with which they interface. Given the breadth of different actions offered by different robots within ROS, applying a framework for task planning such as ROSPlan might require the implementation of many such interface nodes. The contribution of this paper is a single light-weight node that is able to interface any action through the use of a configuration file.

### 3 Related Work

In this section we discuss how different frameworks handle the mapping of planned actions to their implementations in ROS. While there are many approaches to plan representation and execution discussed here, they must all handle the problem of interfacing with the services and actionlibs provided by ROS libraries.

In the modified ROSPlan framework developed by Sanelli et al. (2017) the generated plan is transformed into a Petri-Net Plan (PNP) and then passed to the PNP executor. The latter communicates with the software of the robot over the *PNP Service* and *PNP Action Server* modules, through services and actionlibs. Another modified version of the ROSPlan is presented in Harman et al. (2017). The authors have adapted the framework to communicate not only with the ROS nodes of the integrated robots, but also with IoT devices from the environment. They have replaced the ROSPlan *Action Interfaces* nodes with *Action Executor* nodes, but use the same communication strategies to the robots and the IoT devices, over ROS actionlibs.

In other works, as in the MaestROB framework (Munawar et al. 2018), the communication to the low-level robot controllers is done over the Intu middleware and corresponding ROS-Bridges. The communications structures are similar, as they are based on the publisher/subscriber concept. PLATINUM is another framework that uses the ROS-Bridges to send commands to the *Motion Planning* module of the serial manipulator and to receive the execution feedback from it (Cesta, Orlandini, and Umbrico 2018). ROS-TiPIEx (Viola et al. 2019), is a tool that allows the interaction between a planning and an expert in robotics for designing the planning problem and corresponding low-level control strategies for dispatching the planned actions. ROS-TiPIEx uses a timeline planning approach, while the orchestration of the low-level controls to ROS modules is based on FSMs. The tool comes with a GUI that simplifies the configuration process and improves the information sharing between the experts.

In Darvish et al. (2018), the FlexHRC architecture is presented. Similar to the above frameworks, it has a *Robot Execution Manager* that receives the high-level plan and sends individual actions to the *Controller*. The latter sends further

low-level commands to the robot itself, which returns sensory data in form of a feedback, using similar communication structures to ROS. Furthermore, the *Controller* maps the received actions to one low-level command or to a sequence of low-level commands, similar to a FSM action interface. A similar approach for grouping *atomic actions* in *compound skill*, which are hierarchical and concurrent state-machines, is utilized in (Johannsmeier and Haddadin 2017). In a similar manner, in (Munawar et al. 2018) they refer to the execution triggered by only one call as a *gesture*, while the execution of a sequence of such calls integrated in a structure equivalent to an FSM is named a *skill*.

The contribution of this paper is a general solution, in ROS, for the action interface component that all of these related approaches must implement. Specifically we present a highly flexible action interface node for the ROSPlan framework. Through the new action interface the concrete implementation of planned actions can be easily and intuitively configured. The interface allows the creation and the modification of execution strategies, including complex ones such as hierarchical finite state machines, in a user-friendly manner. Only a configuration file is required, which can be created through user interface, and no code must be generated.

### 4 ROSPlan Action Interface

In this section, we discuss the limitations of the existing action interface used in ROSPlan. The initial implementation of the mappings between the abstract PDDL actions and the low-level concrete ROS modules is through C++ code. Users could also implement such custom interfaces between ROSPlan and lower-level executors in the language of their choice (i.e., Python) handling all the updates to the knowledge base and planning state, which are handled in ROSPlan’s abstract C++ interface. For each action defined in the PDDL domain file of the planning instance, one action interface had to be defined as a ROS node. This action interface integrates the concrete implementation that interacts with the different ROS modules. For example, consider the action interface of a *move* PDDL action. This contains the generation of a ROS goal message and a call to the MOVE BASE actionlib of the actor, which achieves that goal. Further strategies, as cleaning the map in case that the first execution command to MOVE BASE has failed, must also be hard-coded into this interface.

Each planned PDDL action is a grounded action. This means that it should be executed for the specific values of its parameters. For example, the *move* action might have as parameters the actor *agent*, the starting pose *from* and the goal pose *to*. This implies that the *move* action interface must be able to interpret this and generate the correct ROS actionlib goal for all possible variations of these parameters. This challenge is also tackled in the hard-coded implementation, often with the definition and the integration of an action-specific configuration file. In the example case, the *move* action interface takes as input a *poses* configuration file, where all possible values of the *to* parameter are mapped to specific coordinates required to generate the corresponding MOVE BASE goal.



Although the actual action interfaces are apparently easy to generate, they actually have some important limitations. The most relevant one is in scalability. For each new PDDL action defined in the domain file, a new ROS node must be generated. Moreover, if the PDDL action has one or more parameters whose values are relevant for the execution, a specific configuration file must also be created.

Other issues are the reusability and readability. Each action interface may require an implementation that is unique to both the action implementation and the planning domain, as well as a configuration file that is unique to the planning instance. Thus, it is hard to define and maintain a clear structure that can be easily understood and modified by new users, or reused in case new PDDL actions are defined. As a consequence, code replication is common among interfaces for similar actions. This also complicates the code maintainability.

## 5 Action Interface Manager

The issues presented in the previous section motivated the remodelling of the ROSPlan action interfaces. Their new structure and the advantages that they bring with them are detailed in the following.

### Implementation

This new action interface has been implemented in Python. Similar to ROSPlan’s sensing interface (Canal et al. 2019), we have exploited the Python’s language reflection capabilities to load at run-time all the required modules to run the low-level controllers.

As depicted in Figure 1, we developed an action interface manager (AIM) that loads the single configuration file and creates the corresponding interfaces to different low-level modules for each of the actions. This object is also in charge of receiving the action dispatch messages that indicate which (grounded) PDDL action should be executed, and starting the corresponding interface. When the action execution finishes, the interface informs the dispatcher about the outcome of the execution.

We have developed three action interface types: Actionlib, Service, and FSM. However, our architecture has been designed such that it is easy to extend to handle other types of interfaces in a similar manner, such as to IoT devices, provided users write ROS wrappers for those APIs. In addition to specifying the type of each interface, the configuration file describes how each interface should construct service requests and actionlib goals, as well as their expected feedbacks or results. In this specification, PDDL parameters and values stored in the ROS parameter server can be accessed using the syntax (`$pddlparam x`) and (`$rosparam x`). This allows the same action interface configuration to scale to actions with many possible parameter variations without requiring additional lines. The interface types are described at a high-level below, followed by example configurations. The complete specification of the configuration file is presented in Appendix A.

**Service and Actionlib action interfaces** We provide interface implementations to the basic ROS communication

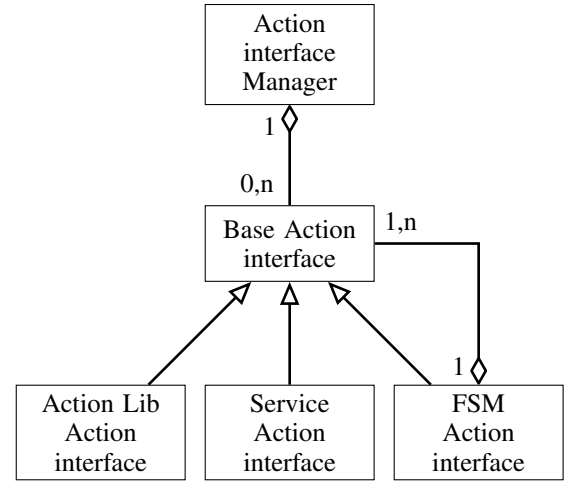


Figure 1: UML Diagram of the the Action Interface classes

protocols: actionlibs and services. They are both provide request/reply interaction, the former non-blocking and the latter blocking.

Both mechanisms require a request or goal to be achieved, and send back a response with the result once finished. The configuration file describes (a) the address of the service/actionlib, (b) how the default requests are constructed based on the PDDL parameters of the action, possibly using data from the ROS Parameter Server, (c) how these default requests may be overridden for specific combinations of PDDL parameters, for which a different request may be required, and (d) a description of the expected response to a service or actionlib call that indicates the action has been successfully executed.

When a planned action is dispatched, the AIM retrieves the configuration to run the action, and passes it to the corresponding interface. Then, the corresponding interface builds the request message, calls the underlying interface, checks its response when the action execution is completed, and returns the result to the AIM. If it has been defined, the expected response is used to determine whether the PDDL action has succeeded.

**FSM action interface** The third interface describes action executions as a Finite State Machine (FSM), allowing for more complex action definitions. The action interface defines a set of named states that could be executed, each linked to an action interface, including another nested FSM. Each state defines the transitions to other states in the case of successful or failed execution. Each transition defines both the next state that will be executed (by name or to the special state “goal.state” that completes the FSM), and the PDDL effects that will be applied before transitioning to the next state. The formulation of an FSM action is described fully by Bezrucav and Corves (Bezrucav and Corves 2020). The use of the composite design pattern allows the reuse of existing interfaces, while remaining powerful enough to create complex hierarchies of state machines.

## Integration in ROSPlan

Figure 2 depicts the interactions with different components of the ROSPlan framework. The new action interface is an independent node subscribed to the ROSPlan Dispatcher topics to be informed when actions need to be executed, and publishes feedback with the result of the execution.

The new node is fully compatible with existing action interfaces, which can be run alongside the new action interface manager. The dispatch message of an action will be ignored by the AIM if it is not in the configuration file, and will be picked by the corresponding legacy action interface, which will start the action execution.

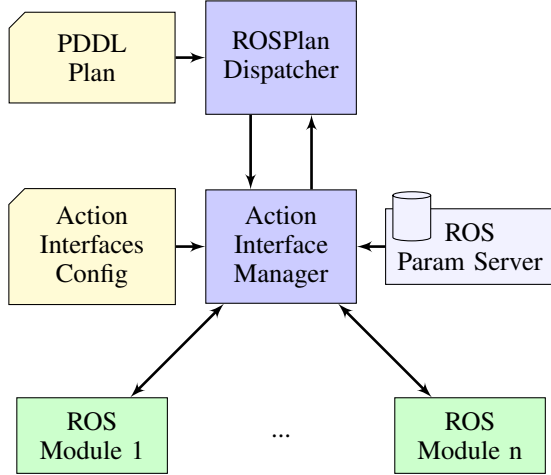


Figure 2: Interactions between the the ROSPlan framework, involved robots, ROS Parameter Server, plan file, and action interface configuration.

## 6 Configuration Examples

All task planning frameworks for robotics created so far contain a module through which the planned actions are mapped to their concrete execution. Because there is no standard for the development process of such a module, the implementations are diverse. This issue makes it hard to determine a set of indicators with respect to which such modules from different projects can be compared. With this motivation, in this section we demonstrate exactly what is required to map between the PDDL actions and their concrete execution with the AIM. Namely, the user needs to generate the configuration file. Two examples of the configuration files are presented:

1. The Navigation Scenario is a relatively simple scenario with only one PDDL action.
2. In the Factory Scenario, the execution of seven different complex actions are configured, for two different types of actors in a factory.

### Navigation Scenario

In the navigation scenario, one or more mobile base robots are tasked with navigating around a known map. The goal

```

1  ;; Move between any two waypoints, avoiding terrain
2  (:durative-action goto_waypoint
3    :parameters (?v - robot ?from ?to - waypoint)
4    :duration ( = ?duration (distance ?from ?to))
5    :condition (and
6      (at start (robot_at ?v ?from)))
7    :effect (and
8      (at end (visited ?to))
9      (at start (not (robot_at ?v ?from)))
10     (at end (robot_at ?v ?to)))
11  )

```

Listing 1: The *goto\_waypoint* PDDL action of the Navigation Scenario.

```

1  actions:
2    - name: goto_waypoint
3      interface_type: actionlib
4      default_actionlib_topic: /($pddlparam
5        ↪ v)/move_base
6      default_actionlib_msg_type:
7        ↪ move_base_msgs/MoveBase
8      default_actionlib_goal:
9        target_pose.header.frame_id: "map"
10       target_pose.pose.position.x: ($rosparam
11         ↪ /wp/($pddlparam to))[0]
12       target_pose.pose.position.y: ($rosparam
13         ↪ /wp/($pddlparam to))[1]
14       target_pose.pose.orientation.w: 1

```

Listing 2: Complete configuration for the navigation scenario.

of the automated planning problem is that each waypoint has been visited once by any robot, and from a planning perspective is very simple. The single PDDL action is named *goto\_waypoint*, and is depicted in Listing 1.

The execution of this action is carried out by the MOVE BASE module. Listing 2 shows the complete configuration file for the scenario, which connects the *goto\_waypoint* action with MOVE BASE. The action configuration specifies the actionlib topic for the action, based on the *robot* PDDL parameter (line 4). The target coordinates of the MOVE BASE action are set from values stored in the ROS parameter server, using the PDDL parameter *to* as key (lines 8-9). This is everything a user needs to supply to connect ROSPlan with MOVE BASE. As an informal comparison, the pre-existing C++ interface to MOVE BASE consists of 192 lines of code across two files and must be configured separately for each robot.<sup>1</sup> This stark contrast illustrates the utility of the new interface: it is fast and low effort (in this case 10 lines) to interface actions and assemble an automated planning system for ROS.

<sup>1</sup>The preexisting MOVE BASE interface in the ROSPlan repository [https://github.com/KCL-Planning/rosplan\\_demos/tree/master/rosplan\\_demos/interfaces/rosplan\\_interface\\_movebase](https://github.com/KCL-Planning/rosplan_demos/tree/master/rosplan_demos/interfaces/rosplan_interface_movebase).

## Factory Scenario

The AIM is demonstrated in the simulated Factory Scenario<sup>2</sup> developed as part of the Sharework<sup>3</sup> project. In this scenario there are two agents, a human and an Autonomous Guided Vehicle, which can execute seven different complex actions: *move*, *load*, *unload*, *attach tool*, *detach tool*, *manipulation action 1* and *manipulation action 2*. For the *move* action only one PDDL durative-action is defined, while for the other six actions two PDDL durative-actions are created, one for each type of actor. Furthermore, each of these actions requires a *multiple-step execution with failure-catching structures*. Multiple-step execution means that the execution of the planned action requires calling multiple ROS module in sequence to successfully execute. Failure-catching structures means that for the safe and robust execution of the action, failed execution of any step requires additional steps to return the system to a safe state from which further planning can occur. Given these requirements, a FSM action interface is configured for each PDDL action.

An excerpt of the configuration for the *move* PDDL action is depicted in Listing 3. As mentioned above, a finite state machine is required to describe the sequence of basic actions that need to be executed in order to reach the goal state or to prevent error situations. In Listing 3 the first two states of that FSM are presented. The execution of the first state (lines 5-17) is independent on the grounding of the PDDL *move* action. Therefore, the default values for the message type, topic, goal and result are set (lines 7 - 12). The default behaviour is to call a service named “action failure”. Such services are helpful in simulations, where failures may need to be generated on purpose.

Upon success of the first state (lines 14 - 15), the execution is continued with the state *bal*. The execution of state *bal* (lines 18 - 31) is dependent on the grounded *agent* and *to* parameters of the *move* PDDL action (line 21). This state performs an actionlib call to MOVE BASE for a given set of parameters. For example, given specific parameters of the PDDL action (line 23) the goal description is generated with coordinates corresponding to *to* and sent on topic corresponding to the *agent* (lines 26 - 28).

This is only a snippet of the entire configuration for the *move* PDDL action that shows the high flexibility of the new action interfaces in describing complex and robust behaviours for different execution variants. In order to better comprehend this complexity, Figure 3 shows half of the FSM action interface for the *move* PDDL action. In this graph, states are represented as nodes and transitions between them are represented as edges. This picture also illustrates the failure-catching behaviour. For example, in the last displayed state *move\_to\_goal\_reverse* the entire PDDL action is reversed to attempt to return the robot from an obstructed state to the initial position.

The implementation without the new AIM requires the creation of 13 different Action Interface ROS nodes, each with a specialized C++ class that contains the implementation of the function `concreteCallback`. In this function

```
1  actions:
2  - name: move
3    interface_type: fsm
4    states:
5      - name: start_state
6        interface_type: service
7        default_service_msg_type:
8          ↳ action_failure_srvs/ActionFailureSimulator
9        default_service:
10         ↳ /action_failure/action_failure_simulator
11        default_service_request:
12         probability: 0.0
13        default_service_response:
14         response: "success"
15        transitions:
16         succeeded:
17           - to_state: bal
18         failed:
19           - to_state: error_state
20      - name: bal
21        interface_type: actionlib
22        default_actionlib_msg_type:
23         ↳ move_base_msgs/MoveBase
24        pddl_parameters: [agent, to]
25        parameter_values:
26         - values: [summit_xl_1, workbench11]
27         actionlib_topic: /summit_xl_1/move_base
28         actionlib_goal:
29           target_pose.header.frame_id:
30             ↳ "summit_xl_1/map"
31           target_pose.pose.position.x: 0.6
32           target_pose.pose.position.y: -0.6
33         - values: [summit_xl_2, workbench12]
34         actionlib_topic: /summit_xl_2/move_base
35         actionlib_goal:
36         ...
```

Listing 3: Excerpt of the configuration file for the Factory Scenario.

both the sequence of the required actions for a successful execution of the planned PDDL action, as well as all the recovery procedures must be hard coded. Furthermore, the complex actions of the Factory Scenario, (i.e., *move*, *load* or *manipulation\_action\_1*) interface with both standard ROS modules (i.e. MOVE BASE or MOVE IT) and also custom modules. Each such module requires a unique implementation within the action interface. Last, in order to consider the configurations for all possible grounded parameters, a configuration file must be created for each of these 13 PDDL actions. In comparison, using the AIM no code was written outside of the main configuration file.

## 7 Graphical User Interface

While the AIM offers a low-effort approach to configure new actions, and offers the ability to set up more complex FSM actions, it can still require a very long configuration file. In the configuration file for the 13 PDDL actions of the Factory Scenario, almost 130 states are defined and the description of each one can have up to 15 lines. To intuitively manage this high amount of information, a Graphical User Interface

<sup>2</sup><https://youtu.be/8Onh9SKF1yk>

<sup>3</sup><https://sharework-project.eu/>

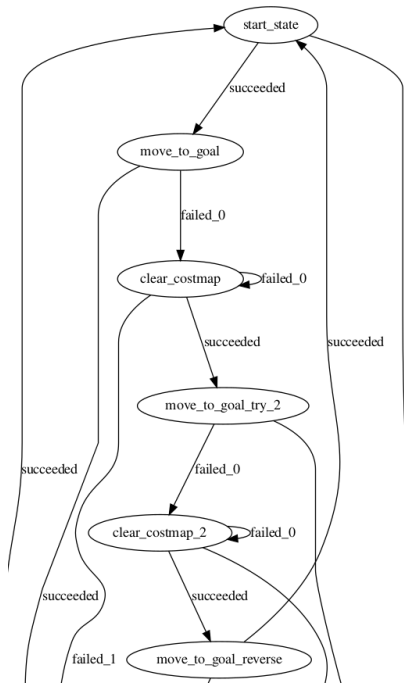


Figure 3: Part of the FSM Action Interface for the *move* action represented as a graph. After one failed attempt to move to goal, the navigation is retried. After two failed attempts the navigation map is cleared and the agent attempts to return to the initial position.

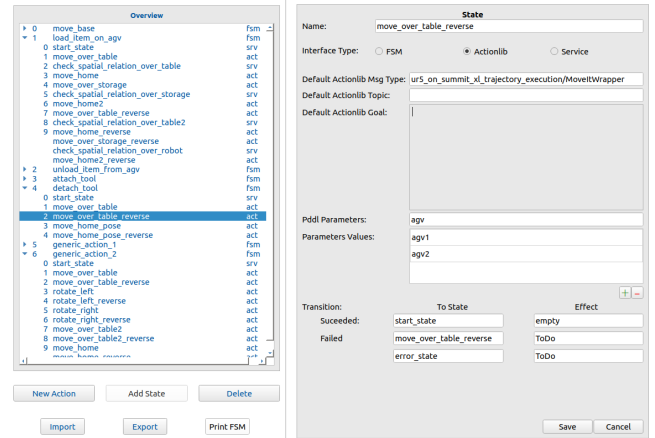
(GUI) was also developed. It sustains the generation and the maintenance of these configuration files and it is already integrated in the ROS ecosystem, as an *rqt* plug-in.

New action interfaces for the PDDL actions can be configured with the GUI. For each the user can select its type (*FSM*, *actionlib* or *service*). The *actionlib* and *service* interfaces can be directly configured by setting the *default* values and, if required, the values for the specific grounded parameters. The *FSM* interfaces require first to define its states. The states can be once again of type *FSM*, *actionlib* or *service*, can be individually configured and, for each of them, the transitions must be defined.

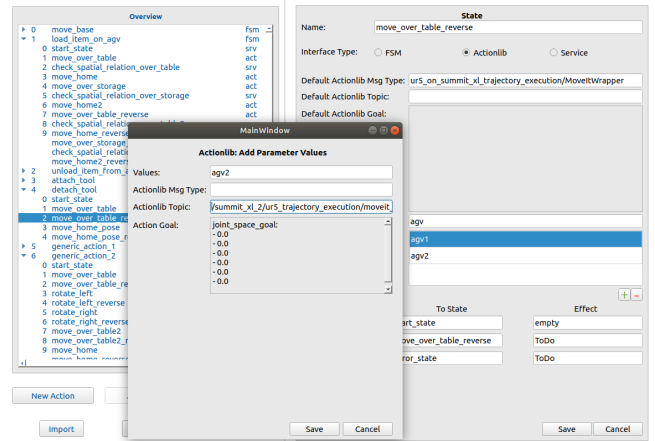
A screenshot of the GUI is depicted in Figure 4a. The main window of the GUI displays the overview of the action interfaces on the left. Those that are defined as *FSMs* may be expanded in a tree view. On the right side, the fields for the configurations are presented. Figure 4b presents the second window of the GUI, used to set the configuration values for the specific PDDL parameters.

Further functionalities of the GUI are the *Import* and *Export* options, through which configuration files can be loaded into the GUI and new or modified ones can be saved on disk. In order to assist the development process of the configuration for *FSMs* action interfaces, their structures can be printed out directly from the GUI. For example, Figure 3 was generated with this functionality.<sup>4</sup>

<sup>4</sup>A video demonstration of the user interface is available online



(a) Main window



(b) Second window for the parameter-specific configuration

Figure 4: The two windows of the GUI through which the different action interfaces can be configured

## Execution Monitor

The AIM provides a single node from which to monitor the progress of different action executions. A tool that monitors this execution progress was created in the ROS ecosystem as an *rqt* plug-in. It displays all the action interfaces, independent of their type, and highlights which of them are active. It also supports the asynchronous tracking of the execution of different planned PDDL actions of the same type. This feature is depicted in Figure 5, where the execution status of two *move* PDDL actions, carried out by two different actors from the Factory Scenario is shown.

## 8 Conclusion

In this paper we have introduced a new tool for the the mapping of planned PDDL actions to concrete execution strategies and integrated it into the ROSPlan framework. The original implementation requires the generation of many hard-coded C++ files and configuration files. The new implemen-

<https://youtu.be/7-nrpOe7hlg>.

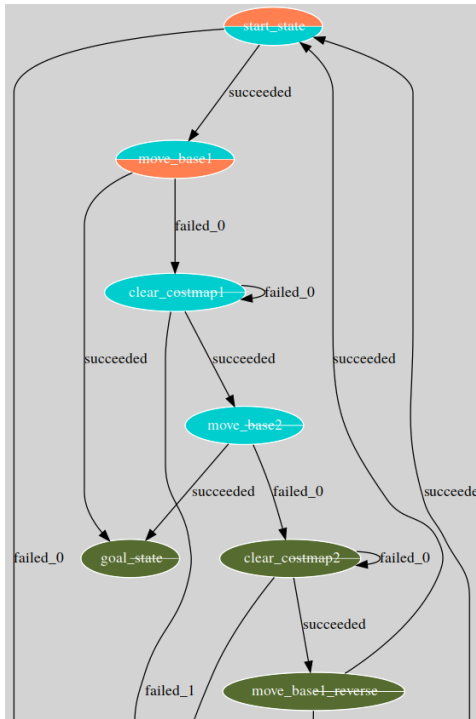


Figure 5: Monitoring of the PDDL actions execution through the configured action interfaces: the execution of one grounded PDDL action has reached the second state of the corresponding FSM action interface (orange), while the parallel execution of another PDDL action of the same type, but with different grounded parameters, has reached the fourth state of the FSM action interface (cyan).

tation is much more flexible and user-friendly, as it only requires the setup of a configuration file. In order to further improve this process, we developed different tools already integrated in the ROS ecosystem such as a GUI and Execution Monitor. The benefits the AIM have been illustrated on two scenarios of different complexity.

The aim of this work is to lower the barrier for inexperienced users of ROS and ROSPlan to build new scenarios that integrate automated planning and robotics. In future work we will continue to expand upon these tools to allow the user to define more complex interfaces, for example that account for durative constraints or non-deterministic effects.

## Acknowledgements

The industrial scenario simulation used for the experimental demonstration was developed as part of the Sharework project, that is funded through the European Union’s Horizon 2020 research and innovation programme under grant agreement No. 820807. This work has also been partially supported by the EPSRC grant THuMP (EP/R033722/1).

## References

Bezrucav, S.-O.; and Corves, B. 2020. Improved AI Planning for Cooperating Teams of Humans and Robots.

In *International Conference on Automated Planning and Scheduling workshop on Planning and Robotics (PlanROB)*.

Canal, G.; Cashmore, M.; Krivić, S.; Alenyà, G.; Magazzeni, D.; and Torras, C. 2019. Probabilistic Planning for Robotics with ROSPlan. In *Towards Autonomous Robotic Systems*, 236–250. Springer International Publishing. ISBN 978-3-030-23807-0. doi:10.1007/978-3-030-23807-0\_20.

Cashmore, M.; Fox, M.; Long, D.; Magazzeni, D.; Ridder, B.; Carrera, A.; Palomeras, N.; Hurtos, N.; and Carreras, M. 2015. ROSPlan: Planning in the Robot Operating System. In Brafman, R., ed., *Proceedings of the Twenty-Fifth International Conference on Automated Planning and Scheduling*, 333–341. Palo Alto, Calif.: AAAI Press. ISBN 9781577357315.

Cesta, A.; Orlandini, A.; and Umbrico, A. 2018. A Dynamic Task Planning System for Advanced Manufacturing Scenarios. In Finzi, A.; Karpas, E.; Nejat, G.; Orlandini, A.; and Srivastava, S., eds., *International Conference on Automated Planning and Scheduling workshop on Planning and Robotics (PlanROB)*, 65–74.

Darvish, K.; Bruno, B.; Simetti, E.; Mastrogiovanni, F.; and Casalino, G. 2018. Interleaved Online Task Planning, Simulation, Task Allocation and Motion Control for Flexible Human-Robot Cooperation. In *2018 27th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN)*, 58–65. IEEE. ISBN 978-1-5386-7980-7. doi:10.1109/ROMAN.2018.8525644.

Harman, H.; Chintamani, K.; and Simoens, P. 2017. Architecture for incorporating Internet-of-Things sensors and actuators into robot task planning in dynamic environments. In *2017 IEEE International Symposium on Robotics and Intelligent Sensors (IRIS)*, 13–18. IEEE. ISBN 978-1-5386-1342-9. doi:10.1109/IRIS.2017.8250091.

Johannsmeier, L.; and Haddadin, S. 2017. A Hierarchical Human-Robot Interaction-Planning Framework for Task Allocation in Collaborative Industrial Assembly Processes. *IEEE Robotics and Automation Letters* 2(1): 41–48. doi:10.1109/LRA.2016.2535907.

McDermott, D.; Ghallab, M.; Howe, A.; Knoblock, C.; Ram, A.; Veloso, M.; Weld, D.; and Wilkins, D. 1998. PDDL - The Planning Domain Definition Language. URL <https://homepages.inf.ed.ac.uk/mfourman/tools/propplan/pddl.pdf>.

Moore, E. F. 1956. Gedanken-experiments on sequential machines. *Automata studies* 34: 129–153.

Munawar, A.; Magistris, G. D.; Pham, T.-H.; Kimura, D.; Tsubori, M.; Moriyama, T.; Tachibana, R.; and Bouch, G. 2018. MaestROB: A Robotics Framework for Integrated Orchestration of Low-Level Control and High-Level Reasoning. URL <http://arxiv.org/pdf/1806.00802v1>.

Quigley, M.; Gerkey, B.; Conley, K.; Faust, J.; Foote, T.; Leibs, J.; Berger, E.; Wheeler, R.; and Ng, A. 2009. ROS: an open-source Robot Operating System. In IEEE, ed., *International Conference on Robotics and Automation Workshop on Open Source Software*, volume 3.

ROS.org. 2021. ROS Concepts. URL <http://wiki.ros.org/ROS/Concepts>.

Sanelli, V.; Cashmore, M.; Magazzeni, D.; and Iocchi, L. 2017. Short-Term Human-Robot Interaction through Conditional Planning and Execution. In Barbulescu, L., ed., *Proceedings of the Twenty-Seventh International Conference on Automated Planning and Scheduling*. Palo Alto, California, USA: AAAI Press. ISBN 978-1577357896.

Viola, C. L.; Orlandini, A.; Umbrico, A.; and Cesta, A. 2019. ROS-TiPIEx: How to make experts in A.I. Planning and Robotics talk together and be happy. In *2019 28th IEEE International Conference on Robot and Human Interactive Communication (RO-MAN)*, 1–6. doi:10.1109/RO-MAN46459.2019.8956417.

## **Appendix A. Action Interface Configuration**

The BNF for the configuration file is given below in four parts: the top-level structure of the file, and then the details of the three action interface types (actionlib, service, and FSM).



## Top-level Structure

```
<configuration> ::= actions:
                    <action>*
<action>          ::= - name: <var>
                    <service-action>|<actionlib-action>|<fsm-action>
<var>             ::= <string>|<rosparam>|<pddlparam>|<var>*
<rosparam>        ::= ($rosparam <var>) /* Use value from ROS param server */
<pddlparam>       ::= ($pddlparam <var>) /* Use value from PDDL parameter */
```

## Actionlib Interface

The actionlib interface configuration is defined below, and an example is shown in Listing 2. Optionally, a default actionlib topic, message type, goal, and result can be set. If no default is set, then it is expected that these will be set per PDDL parameter instead. The optional parameter list *params* is a set of variables matching the parameter labels of the PDDL operator. Each parameter configuration then specifies the objects bound to those parameters, and a topic, message type, or goal that will override the default configuration.

```
<actionlib-action> ::= interface_type: actionlib
                    [default_actionlib_topic: <var>]
                    [default_actionlib_msg_type: <var>]
                    [default_actionlib_goal: <ros-msg>]
                    [default_actionlib_result: <ros-msg>]
                    ::= pddl_parameters: [<var>*] /* PDDL parameters */
                    parameter_values:
                    <al-param-config>*
<al-param-config> ::= - values: [<var>*] /* PDDL objects */
                    [actionlib_topic: <var>]
                    [actionlib_msg_type: <var>]
                    [actionlib_goal: <ros-msg>]
                    [actionlib_result: <ros-msg>]
<ros-msg>         ::= <ros-msg-assign>|<ros-msg>*
<ros-msg-assign>  ::= <var>: <var>
```

## Service Interface

The configuration for service action interfaces is described below. Similar to the actionlib interface, a default service, service type, request, and result can be set. These defaults can be overridden by specified action parameters.

```
<actionlib-action> ::= interface_type: service
                    [default_service: <var>]
                    [default_service_type: <var>]
                    [default_service_request: <ros-msg>]
                    [default_service_result: <ros-msg>]
                    pddl_parameters: [<var>*] /* PDDL parameters */
                    parameter_values:
                    <srv-param-config>*
<srv-param-config> ::= - values: [<var>*] /* PDDL objects */
                    [service: <var>]
                    [service_type: <var>]
                    [service_request: <ros-msg>]
                    [service_result: <ros-msg>]
```

## FSM Interface

```
<fms-action>      ::= interface_type: fsm
                    states:
                    <fsm-state>*
<fsm-state>       ::= - <service-action>|<actionlib-action>|<fsm-action>
                    transitions:
                    succeeded:
                    - to-state: <fsm-transition>
                    failed:
                    - to-state: <fsm-transition>
<fsm-transition> ::= <var>|start_state|goal_state|error_state
```

# An Interactive Approach for the Analysis and Shielding of Partially Observable Monte Carlo Planning Policies

Giulio Mazzi, Giovanni Bagolin, Alberto Castellini, Alessandro Farinelli

Università degli studi di Verona,  
Dipartimento di Informatica,  
Strada Le Grazie 15, 37134, Verona, Italy

giulio.mazzi@univr.it, giovanni.bagolin@studenti.univr.it, alberto.castellini@univr.it, alessandro.farinelli@univr.it

## Abstract

Partially Observable Monte-Carlo Planning (POMCP) is a powerful online algorithm able to generate approximate policies for large Partially Observable Markov Decision Processes. The online nature of this method supports scalability by avoiding complete policy representation. This favors the use of this technique for planning in robotic applications. However, the lack of an explicit representation hinders interpretability. In this work, we present a methodology based on Satisfiability Modulo Theory (SMT) for analyzing POMCP policies by inspecting their traces, namely, sequences of belief-action-observation triplets generated by the algorithm. The methodology is used in an iterative process of trace analysis consisting of three main steps, *i*) the definition of a *question* by means of a parametric logical formula describing (probabilistic) relationships between beliefs and actions, *ii*) the generation of an *answer* by computing the parameters of the logical formula that maximizes the number of satisfied clauses (solving a MAX-SMT problem), *iii*) the analysis of the generated logical formula and the related decision boundaries for identifying unexpected decisions made by POMCP concerning the original question. The result can be used to build a shield that blocks unexpected decisions in future POMCP runs. A graphical user interface is here presented, which supports the iterative process facilitating the interaction between the user and the system, the interpretability of the results, and the generation of shields. The methodology and the related tools are applied to a real-world problem concerning mobile robot navigation.

## 1 Introduction

Planning in a partially observable environment is an important problem in artificial intelligence and robotics. A popular framework to model such problem is that of *Partially Observable Markov Decision Processes (POMDPs)* (Cassandra, Littman, and Zhang 1997) which represent dynamic systems where the state is not directly observable but must be inferred from observations. Computing optimal policies, namely functions that map beliefs (i.e., probability distributions over states) to actions, in this context is PSPACE-complete (Papadimitriou and Tsitsiklis 1987). However, recent approximate and online methods allow handling many real-world problems. A pioneering algorithm for this purpose is *Partially Observable Monte-Carlo Planning (POMCP)* (Silver and Veness 2010) which uses a particle

filter to represent the belief and a Monte-Carlo Tree Search based strategy to compute the policy online. The local representation of the policy made by this algorithm however hinders the interpretation and explanation of the policy itself. In fact, POMCP never represents the entire policy but it generates online the (approximated) Q-function for the current belief.

Explainability is instead becoming a key feature of artificial intelligence systems (Gunning 2019), robotics (Setchi, Dehkordi, and Khan 2020), and planning (Fox, Long, and Magazzeni 2017) since in several contexts humans need to understand why specific decisions are taken by the agent. The presence of erroneous behaviors in these tools (due, for instance, to the wrong setup of internal parameters) may have a strong impact on autonomous cyber-physical and robotic systems that interact with humans, and detecting these errors in automatically generated policies is very hard in practice. For this reason, improving policy explainability is fundamental.

In this work, we present a methodology that can be used to interpret POMCP policies, with a focus on detecting unexpected behavior compared to a set of high-level assumptions that the designer believes to be true. This is based on the XPOMCP methodology presented in (Mazzi, Castellini, and Farinelli 2021a), in which an expert provides qualitative information on system behaviors (e.g., “the robot should move fast if it is highly confident that the path is not cluttered”) and the proposed methodology supplies quantitative details of these statements based on evidence observed in traces (e.g., the proposed approach could say that the robot usually moves fast if the probability to be in a cluttered segment is lower than 1%). Experts are however also interested in identifying states in which the planner does not respect their assumptions. The proposed methodology supports this kind of analysis. It allows expressing partially defined assumptions employing logical formulas, called *rule templates*. The methodology then computes parameters of rule templates from traces using a Satisfiability Modulo Theory (SMT) solver. In this work we present also a web application that supports both the generation of the logical formulas by the expert and the analysis of the results by means of a graphical interface

This work proposes three main contributions to the state-of-the-art, which we summarize in the following:



- we propose an approach for analyzing properties of traces generated by POMCP. A logical rule is first specified by human experts, then variables are instantiated by an SMT-solver, and the achieved logical formula represents trace properties;
- we present a graphical user interface that supports the overall policy analysis process, from template generation to the trace analysis using the logical formula;
- we evaluate the methodology on a robotic navigation problem.

In Section 2 we highlight the main differences between the proposed approach and related works in the literature. Section 3 provides a formalization of the XPOMCP methodology and a description of the graphical user interface. In Section 4, we present a detailed analysis of a robotic navigation problem using the proposed methodology. Finally, Section 5 draws conclusions and sketches directions for future work.

## 2 Related Works

We have identified two main research topics with relationships with our method and goals, namely, policy verification and explainable planning. Formal logic is strongly employed in verification of machine learning and reinforcement learning algorithms.

In recent years, SMT-based approaches have been developed to verify the safety of neural networks (Huang et al. 2017; Katz et al. 2017; Bunel et al. 2018). These methodologies encode the neural network into SMT formulas and check if safety properties hold on these formulas, or they provide counterexamples for properties that are not satisfied. To the best of our knowledge, there is no equivalent approach to verify that a specific property holds on policies generated by POMDPs, and in particular by POMCP. A possibility to use SMT-based approaches to verify POMCP policies is to encode the POMDP problem in one of the logic-based frameworks presented in (Cashmore et al. 2016; Norman, Parker, and Zou 2017; Wang, Chaudhuri, and Kavraki 2018; Bastani, Pu, and Solar-Lezama 2018), where property guarantees can be formally proved. However, these frameworks use SMT-solvers to build a policy that satisfies predefined properties while we use a MAX-SMT representation of the problem with a different goal, namely to evaluate if the policy satisfies expert assumptions. Namely, we aim at enhancing policy explainability without altering the policy itself.

A work in which verification is achieved by exploiting a simplified representation of the problem provided by an expert is (Zhu et al. 2019). It describes a method for verifying properties related to the safety of fully observable systems modeled by Markov Decision Processes (MDPs). The approach works on a pre-trained neural network representing a black-box policy. It uses a linear formula summarizing the policy behavior to allow using off-the-shelf verification tools. This differs from our work for two reasons: first, we work on partially observable environments, and our logical formulas work on beliefs instead of states; second, in (Zhu et al. 2019) the formula is used as an input to the verification

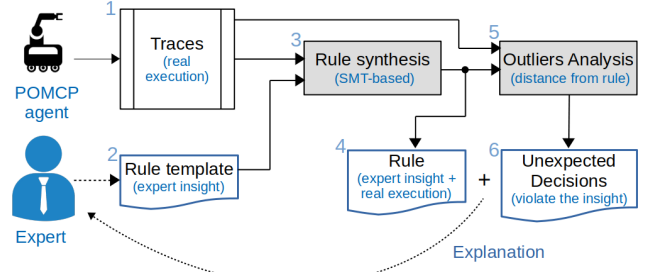


Figure 1: Methodology overview.

tool while we use it to interact with humans and improve policy explainability.

*Explainable Artificial Intelligence (XAI)* (Gunning 2019) is a rapidly growing research field focusing on human interpretability and understanding of artificial intelligence (AI) systems. In particular Explainable planning (XAIP) (Cashmore et al. 2019; Fox, Long, and Magazzeni 2017; Langley et al. 2017; Anjomshoar et al. 2019) aims at investigating planning tools that come with justifications for the decisions they make. In our work, we use the high-level insight provided by the user to build an explanation of the policy in use. A particularly interesting kind of questions analyzed in XAIP are known as *contrastive question* (Fox, Long, and Magazzeni 2017). They are used to structure the interaction between humans and the AI systems to be explained. In these questions, the expert asks the agent question as “Why have you made this decision instead of this other one, that I believe could be a better option?” and the system answers motivating its choice instead of the alternative one. These questions are, however, very difficult to answer in online frameworks as POMCP (Castellini et al. 2020) because the information required to build the answer may not be available to the agent at run time. We, therefore, do not use contrastive questions but ground the interaction between human and planner on logical formulas that are framed by the expert, using her/his insight, and then instantiated by the SMT solver according to the observed behavior of the system. The identification of decisions that violate user’s expectation allows then to generate an iterative process in which the expert, that can refine the rule interactively, acquires new understanding about the policy.

## 3 Methods

In this section, we provide a full description of the proposed methodology and a presentation of the web application that implements it. We also present a running example concerning a mobile robot to show a direct application of the most important concepts.

### 3.1 Method overview

The proposed methodology, called *XPOMCP*, is summarized in Figure 1. It leverages the expressiveness of logical formulas to represent specific properties of the investigated policy. As a first step, a logical formula with free variables is

defined (see box 2 in Figure 1) to describe a property of interest of the policy under investigation. This formula, called *rule template*, defines a relationship between some properties of the belief (e.g., the probability to be in a specific state) and an action. Free variables in the formula allow the expert to avoid quantifying the limits of this relationship. For instance, a template saying “Do this when the probability of avoiding collisions is at least  $\bar{x}$ ”, with  $\bar{x}$  free variable, is transformed into “Do this when the probability of avoiding collisions is at least 0.85”. These free variables are determined by analyzing a *trace* (see box 1). In the following, we call trace a set of runs performed by POMCP on a specific problem. Each run is a set of *steps*, and each step corresponds to an action performed by the agent having a belief and receiving an observation from the environment.

By defining a rule template the expert provides useful prior knowledge about the structure of the investigated property. Hence, the rule template defines the *question* asked by the expert. The *answer* to this question is provided by the SMT solver (see box 3), which computes optimal values for the free variables to make the formula explain as many actions as possible in the observed traces.

The rule (see box 4) provides a human-readable local representation of the policy function that incorporates the prior knowledge specified by the expert, and it allows to split trace steps into two classes, namely, those satisfying the rule and those not satisfying it. The approach, therefore, allows identifying unexpected decisions (see box 6), related to actions that violate the logical rule (i.e., that do not verify the expert’s assumption). The quantification of the violation, i.e., the distance between the rule boundary and the violation, also supports the analysis because it provides an explicit way to explain the violations themselves, which could even be completely unexpected due to expert imprecise knowledge or policy errors.

### 3.2 The eXplainable POMCP Web Application

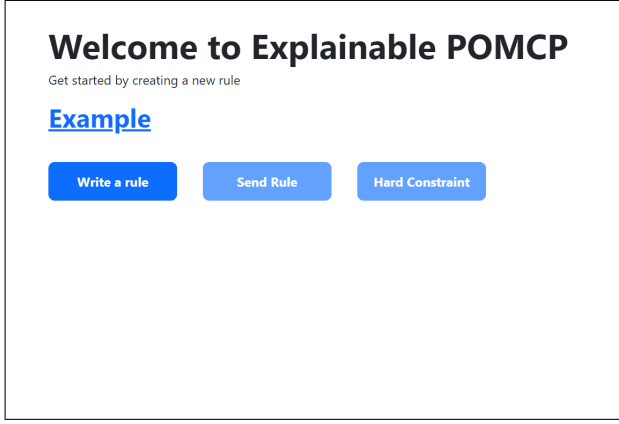
We present a web application that can be used to interact with the XPOMCP methodology. The front-end is written in *JavaScript* and uses the *React* library, the back-end is implemented using *Python* and the *Flask* library. XPOMCP employs the Z3 SMT-solver (De Moura and Bjørner 2008) to compute parameters for the rule templates. Figure 2a shows the homepage of the application. The *Example* link can be used to load a predefined example, or it is possible to use the *Write a rule* button to build a new rule from scratch. Right now, two domains are defined: *tiger* that presents the famous POMDP problem (Cassandra, Littman, and Zhang 1997), and *velocity regulation* that presents the robotic problem described in Section 3.3.

Figure 2b shows the template writing process. After clicking the *Write a rule* button, a sequence of pop-up panels presents questions on the template that the user wants to create. It asks which domain to consider (*tiger* or *velocity regulation*), which trace to use (it is possible to load a new trace file or use one of the traces already stored in the system), the action described by the rule, and a logical formula that describe the expert insight on when to select that specific actions. The logical formula is in disjunctive normal

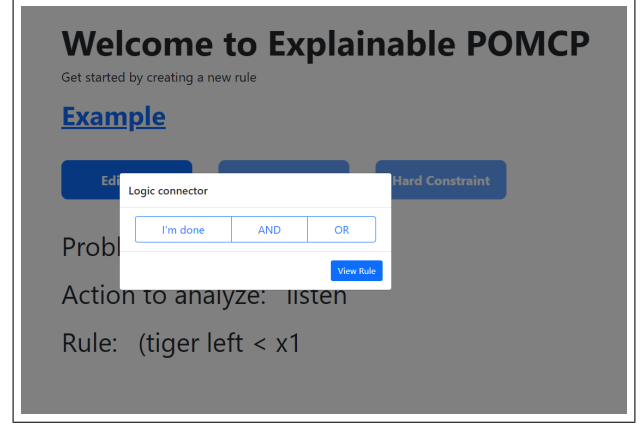
form and contains comparisons between properties of the trace and free-variables. The web interface makes it easy to create and reuse free-variables and to access the data stored in the trace. It is also possible to add hard constraints using the *Hard Constraint* button, as described in Section 3.6. Figure 2c presents an example of complete rule for the tiger domain. When the template is complete, the *Send Rule* button can be used to compute the rule parameters on the specific trace. The back-end uses XPOMCP to compute values for the free variables by analyzing the trace, and the results are presented in Figure 2d. The upper part of the page shows the final rule (i.e., the rule template with all its free variables instantiated). Under the rule, the page presents a list of unsatisfiable steps (on the left) and a graphical representation of the rule (right). The list of unsatisfiable steps is ordered in decreasing values of Hellinger distance ( $H^2$ ), a metric that quantifies the severity of an unsatisfiable step, as described in Section 3.7. They also show the run, the step, the action, and the belief of the steps. The graphical representation of the rule shows columns with the distribution of probabilities. All the unsatisfiable steps are also printed inside the graphical representation of the rule and can be selected to further analyze these problematic steps. For example, if we select the first unsatisfiable step, it shows that this is an unexpected decision because the agent decided to listen even if it was 96.7% sure that the tiger is in the left door (with this belief, the correct action is to open the right door). Section 4 presents other examples of the GUI.

### 3.3 Running example: velocity regulation in mobile navigation

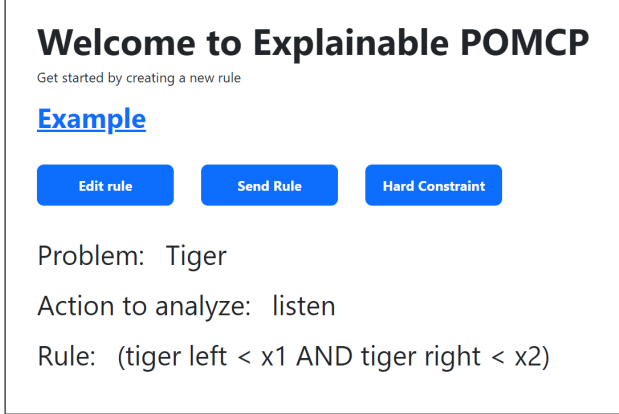
We present a problem of velocity regulation in robotic platforms as a case study to show how XPOMCP works. The same problem is used also in Section 4 to present a use case of the software. A robot travels on a pre-specified path divided into eight *segments* which are in turn divided into *sub-segments* of different sizes, as shown in Figure 3. Each segment has a (hidden) difficulty value among *clear* ( $f = 0$ , where  $f$  is used to identify the difficulty), *lightly obstructed* ( $f = 1$ ) or *heavily obstructed* ( $f = 2$ ). All the subsegments in a segment share the same difficulty value, hence the hidden state-space has  $3^8$  states. The goal of the robot is to travel on this path as fast as possible while avoiding collisions. In each subsegment, the robot must decide a *speed level*  $a$  (i.e., action). We consider three different speed levels, namely slow, medium, and fast. The reward received for traversing a subsegment is equal to the length of the subsegment multiplied by 1 for low speed, 2 for medium speed, and 3 for fast speed. The higher the speed, the higher the reward, but a higher speed suffers a greater risk of collision (see the collision probability table  $p(c = 1 \mid f, a)$  in Figure 3.c). The real difficulty of each segment is unknown to the robot (i.e., hidden part of the state), but in each subsegment, the robot receives an observation, which is 0 (no obstacles) or 1 (obstacles) with a probability depending on segment difficulty (see Figure 3.b). The state of the problem contains a hidden variable (i.e., the difficulty of each segment), and three observable variables (current segment, subsegment, and time elapsed since the beginning).



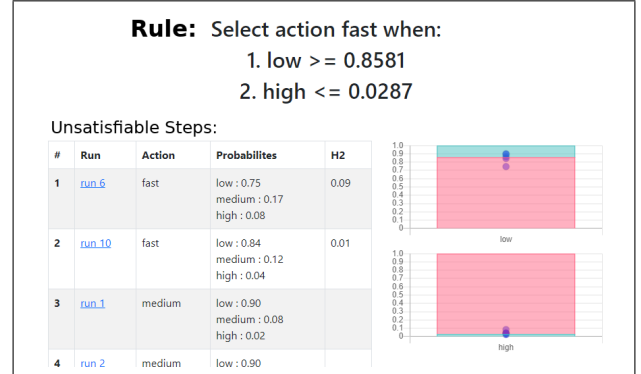
(a) Homepage of the XPOMCP Web App



(b) Creation of a rule template



(c) Complete rule template



(d) Example of results

Figure 2: Usage of the XPOMCP Web App.

We are interested in a rule describing when the robot travels at maximum speed (i.e.,  $a = fast$ ). We expect that the robot should move at that speed only if it is confident enough to be in an easy-to-navigate segment, but this level of confidence varies slightly from segment to segment (due to the length of the segments, the elapsed times, or the relative difficulty of the current segment in comparison to the others). To obtain a rule that is compact but informative, we want the rule to be a local approximation of the behavior of the robot, thus we only focus on the current segment without considering the path as a whole when we write this rule. The task of XPOMCP is to find the actual bounds on the probability distribution (i.e., belief) that the POMCP algorithm uses to make its decisions and to highlight the (unexpected) decisions that do not comply with this representation. The algorithm identifies a set of notable exceptions in which the robot decides to move at high speed even if it was not certain that the segment was free of obstacles.

### 3.4 Partially Observable Monte-Carlo Planning

A Partially Observable Markov Decision Process (POMDP) (Kaelbling, Littman, and Cassandra 1998) is a tuple  $(S, A, O, T, Z, R, \gamma)$ , where  $S$  is a set of

partially observable *states*,  $A$  is a set of *actions*,  $Z$  is a finite set of *observations*,  $T: S \times A \rightarrow \Pi(S)$  is the *state-transition model*, with  $\Pi(S)$  probability distribution over states,  $O: S \times A \rightarrow \Pi(Z)$  is the *observation model*,  $R: S \times A \rightarrow \mathbb{R}$  is the *reward function* and  $\gamma \in [0, 1]$  is a *discount factor*. An agent must maximize the *discounted return*  $E[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)]$ . A probability distribution over states, called *belief*, is used to represent the partial observability of the true state. To solve a POMDP it is required to find a *policy*, namely a function  $\pi: B \rightarrow A$  that maps beliefs  $B$  into actions.

We use *Partially Observable Monte-Carlo Planning (POMCP)* (Silver and Veness 2010) to solve POMDPs. POMCP is an *online* algorithm that solves POMDPs by using Monte-Carlo techniques. The strength of POMCP is that it does not require an explicit definition of the transition model, observation model, and reward. Instead, it uses a black-box to simulate the environment. POMCP uses a *Monte-Carlo Tree Search* (MCTS) at each time-step to explore the belief space and select the best action. *Upper Confidence Bound for Trees* (UCT) (Kocsis and Szepesvári 2006) is used as a search strategy to select the subtrees to explore and balance exploration and exploitation. The be-

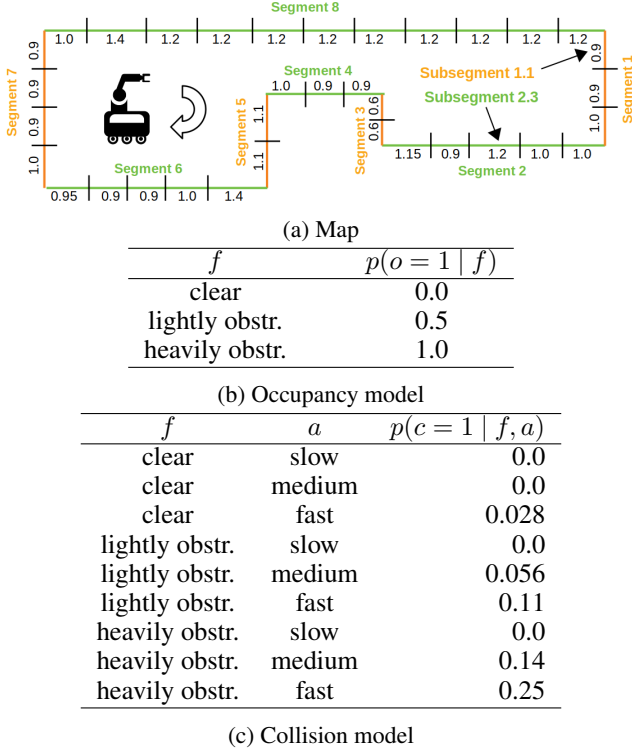


Figure 3: Main elements of the POMDP model for the velocity regulation problem. (a) Path map. The map presents the length (in meters) for each subsegment. (b) Occupancy model  $p(o | f)$ : probability of observing a subsegment occupancy given segment difficulty. (c) Collision model  $p(c | f, a)$ : collision probability given segment difficulty and action.

lief is implemented as a *particle filter*, which is a sampling over the possible states that is updated at every step. At each time-step a particle is selected from the filter, each particle represents a state. This state is used as an initial point to perform a simulation in the Monte-Carlo tree. Each simulation is a sequence of action-observation pairs and it collects a discounted return, and for each action, we can compute the expected reward that can be achieved. The particle filter is updated after receiving an observation. If required, new particles can be generated from the current state through a process of *particle reinvigoration*.

### 3.5 SMT and MAX-SMT

The problem of reasoning on the satisfiability of formulas involving propositional logic and first-order theories is called *Satisfiability Modulo Theory* (SMT). In XPOMCP, we use propositional logic and the theory of linear real arithmetic to encode the rules that describe the behavior of policies, and we use Z3 (De Moura and Bjørner 2008) to solve the SMT problem. We encode our formulas as a MAX-SMT problem, which has two kinds of clauses, namely, *hard*, that must be satisfied, and *soft* that can be satisfied. A model of the MAX-SMT problem hence satisfies all the hard clauses and

as many soft clauses as possible, and it is unsatisfiable only when hard clauses are unsatisfiable. Our rules are intended to describe as many decisions as possible among those taken by the policy hence MAX-SMT provides a perfect formalism to encode this requirement. The Z3 solver is used to solve the MAX-SMT problem (Bjørner, Phan, and Fleckenstein 2015). Subsection 3.6 presents the details of this encoding.

The key ingredient for the MAX-SMT formulation are *rules* and *rule templates*. A rule template represents the *question* the expert wants to investigate. It is a set of first-order logic formulas without quantifiers explaining some properties of the policy, and has the following form:

$$\begin{aligned}
 r_1: & \text{select } a_1 \text{ when } (\bigvee_{i^1} \text{subformula}_{i^1}); \\
 & \dots \\
 r_n: & \text{select } a_n \text{ when } (\bigvee_{i^n} \text{subformula}_{i^n}); \\
 & [ \text{where } \bigwedge_j (\text{requirements}_j); ]
 \end{aligned} \tag{1}$$

where  $r_1, \dots, r_n$  are *action rule templates*. A *subformula* is defined as  $\bigwedge_k p_s \approx \bar{x}_k$ , where  $p_s$  is the probability of state  $s$ , symbol  $\approx \in \{<, >, \geq, \leq\}$ , and  $\bar{x}_k$  is a free variable that is automatically instantiated by the SMT solver analyzing the traces (when the problem is satisfiable). In general, bold letters with an overline (e.g.,  $\bar{x}, \bar{y}$ ) are used to identify free variable while italic letters (e.g.,  $p, a_i$ ) are used for fixed values read from the trace. The *where* statement can be used to specify an optional set of hard requirements that can take different forms, such as the definition of a minimum value (e.g.,  $\bar{x}_0 \geq 0.9$ ) or a relation (e.g.,  $\bar{x}_2 = \bar{x}_3$ ). These are used to define prior knowledge on the domain which is used by the rule synthesis algorithm to compute optimal parameter values (e.g., equality between two free-variables belonging to different rules can be used to encode the idea that two rules are symmetrical).

For instance, in our running example the actions (i.e., *low*, *medium*, *high*) represent speeds. Each step  $t$  contains the partially observable state ( $\text{segment}^t, \text{subsegment}^t, \text{difficulty}^t$ ) and the selected action  $a^t$ . Since *difficulty* is a probability distribution on  $3^8 = 6561$  states we do not use this value directly. For the sake of brevity, we introduce the *diff* function which takes a distribution on the possible difficulties *distr*, a segment *seg*, and a required difficulty value *d* as input and returns the probability that segment *s* has difficulty *d* in the distribution *distr*. We can now write the rule template:

$$\begin{aligned}
 r_f: & \text{select } a_{fast} \text{ when } p(low) \geq \bar{x}_1 \vee \\
 & p(high) \leq \bar{x}_2; \\
 & \text{where } \bar{x}_1 \geq 0.9 \wedge \\
 & p(low) = \text{diff}(\text{distr}, \text{seg}, \text{clear}) \wedge \\
 & p(high) = \text{diff}(\text{distr}, \text{seg}, \text{heavy})
 \end{aligned} \tag{2}$$

The first literal of  $r_f$  specifies that we select action  $a_{fast}$  if the probability to be in a segment with low difficulty is greater than a certain threshold, where with  $\bar{x}_1 \geq 0.9$  in the

requirement we declare that this threshold must be at least 0.9 (an information that we expect to be true), while the second is an upper bound on the belief that the current segment is hard (i.e.,  $p(\text{high}) \leq \bar{x}_2$ ). To encode Equation (2), for each step  $t$  in the trace we add the clauses:

- $p(\text{low})^t = \text{diff}(\text{difficulty}^t, \text{segment}^t, \text{clear})$ ,
- $p(\text{high})^t = \text{diff}(\text{difficulty}^t, \text{segment}^t, \text{heavy})$
- if the robot performs action  $a_{\text{fast}}$  (moving fast), then the formula  $(p(\text{low})^t \leq \bar{x}_1 \vee p(\text{high})^t \geq \bar{x}_2)$  is added to the problem,
- if the robot performs a different action (i.e.,  $a_{\text{slow}}$  or  $a_{\text{medium}}$ ) then the formula  $\neg(p(\text{low})^t \leq \bar{x}_1 \vee p(\text{high})^t \geq \bar{x}_2)$  is added.

Finally, we add the constraints:

- $(\bar{x}_1 \geq 0.0 \wedge \bar{x}_1 \leq 1.0) \wedge (\bar{x}_2 \geq 0.0 \wedge \bar{x}_2 \leq 1.0)$  to ensure that  $\bar{x}_1$  and  $\bar{x}_2$  are probabilities,
- $\bar{x}_1 \geq 0.9$  to force the hard constraint.

A *learned rule* is a rule template with all free variables instantiated (e.g.,  $\bar{x}_1, \bar{x}_2$ ). For a rule to properly describe a trace generated by a policy, all steps in the trace should satisfy the rule (i.e., the action defined in the rule should be taken in a step if and only if the belief satisfies the rule conditions). This is however almost impossible in real traces because the policy is usually a complex formula. For this reason, we implemented a soft mechanism to check clause satisfiability, as described in Subsection 3.6. In our example, from rule template (2) and a trace we obtain the learned rule:

$$r_f: \text{select } a_{\text{fast}} \text{ when } p(\text{low}) \geq 0.945 \vee p(\text{high}) \leq 0.07;$$

while an example of output provided by XPOMCP when a rule is violated is:

```
Violation in run 2, step 4:
- Selected action: a_fast
- Belief: p(low) = 0.38,
          p(medium) = 0.31,
          p(high) = 0.31.
```

### 3.6 Rule Synthesis

Rule synthesis is performed by Algorithm 1 that takes as input a trace  $ex$  generated by POMCP and a rule template  $r$ . The output is the rule  $r$  with all free variables instantiated to satisfy as many steps of  $ex$  as possible. The *solver* is a Z3 instance used to find a model for the formulas.

The *solver* is first initialized and hard constraints are added in line 1 to force all parameters in the template to satisfy the probability constraint (i.e., to have value in range  $[0, 1]$ ). Then in the *foreach* loop in lines 2–10 the algorithm maximizes the number of steps satisfying the rule template  $r$ . In particular, for each action rule  $r_a$ , where  $a$  is an action, and for each step  $t$  in the trace  $ex$  the algorithm first generates a literal  $l_{a,t}$  (line 4) which is a dummy variable used by MAX-SMT to satisfy clauses that are not satisfiable by a free variable assignment. This literal is then added to the *cost*

---

#### Algorithm 1: RuleSynthesis

---

**Input:** a trace generated by POMCP  $ex$   
a rule template  $r$

**Output:** an instantiation of  $r$

```
1 solver ← probability constraints for thresholds in  $r$ ;
2 foreach action rule  $r_a$  with  $a \in A$  do
3   foreach step  $t$  in  $ex$  do
4     build new dummy literal  $l_{a,t}$ ;
5      $cost \leftarrow cost \cup l_{a,t}$ ;
6     compute  $p_0^t, \dots, p_n^t$  from  $t.\text{particles}$ ;
7      $r_{a,t} \leftarrow$  instantiate rule  $r_a$  using  $p_0^t, \dots, p_n^t$ ;
8     if  $t.\text{action} \neq a$  then
9        $r_{a,t} \leftarrow \neg(r_{a,t})$ ;
10     $solver.add(l_{a,t} \vee r_{a,t})$ ;
11 solver.minimize(cost);
12  $goodness \leftarrow 1 - \text{distance\_to\_observed\_boundary}$ ;
13  $model \leftarrow solver.maximize(goodness)$ ;
14 return  $model$ 
```

---

objective function (line 5) which is a pseudo-boolean function collecting all literals. This function essentially counts the number of fake assignments that correspond to unsatisfied clauses. Afterwards, the belief state probabilities are collected from the particle filter (line 6) and used to instantiate the action rule template  $r_a$  (line 7) by substituting their probability variables  $p_i$  with observed belief probabilities. This generates a new clause  $r_{a,t}$  which represents the constraint for step  $t$ . This constraint is considered in its negated form  $\neg(r_{a,t})$  if the step action  $t.\text{action}$  is different from  $a$  (line 9) because the clause  $r_{a,t}$  should not be true.

The set of logical formulas of the solver is then updated by adding the clause  $l_{a,t} \vee r_{a,t}$ . In this way the added clause can be satisfied in two ways, namely, by finding an assignment of the free variables that makes the clause  $r_{a,t}$  true (the expected behavior) or by assigning a true value to the literal  $l_{a,t}$  (unexpected behavior). The second kind of assignment however has a cost since the dummy variables have been introduced only to allow partial satisfiability of the rules. In line 11, in fact, the solver is asked to find an assignment of free variable which minimizes the cost function, which considers the number of dummy variables assigned to true. This minimization is a typical MAX-SMT problem in which an assignment maximizing the number of satisfied clauses is found. Since there can be more than a single assignment of free variables that achieves the MAX-SMT goal, the last step of the synthesis algorithm (lines 12–13) concerns the identification of the assignment which is closer to the behavior observed in the trace. This problem is solved by maximizing a goodness function that moves the free variables assignment as close as possible to the numbers observed in the trace, without altering the truth assignment of the dummy literals. Notice that this problem concerns the optimization of real variables and it is solved by the linear arithmetic module.



### 3.7 Identification of unexpected decisions

A key element of XPOMCP concerns the characterization of steps that fail to satisfy the rule. They can be seen as *unexpected decisions*, namely, exceptions to the general rule that the expert expects to be true. They can provide useful information for policy interpretation. We define two important classes of exceptions, namely, those related to the approximation made by the logical formula and those actually due to an unpredicted behavior (e.g., an error in the POMCP algorithm, or a decision that cannot be described with only local information). We expect exceptions in the first class to fall quite close to the rule boundary, while exceptions in the second class to be more distant from the boundary. In the following, we call the second kind of exceptions *unexpected decisions* since their behavior is unexpected compared to the expert knowledge on the policy.

In this section, we provide a procedure for differentiating the first class of exceptions from the second one, to find as many unexpected decisions as possible. The input of the procedure is a learned rule  $r$ , a set of steps (called *steps*) that violate the rule, and a threshold  $\tau \in [0, 1]$ . The output of the algorithm is a set of steps related to unexpected decisions. The procedure first randomly generates  $w$  samples (i.e., beliefs)  $\bar{b}_j, j = 1, \dots, w$ , that satisfy the rule. Then, for each belief  $b_i$  in *steps* a distance measure is computed between  $b_i$  and all  $\bar{b}_j, j = 1, \dots, w$ . The minimum distance  $h_i$  is finally computed for each  $b_i$  and compared to a threshold  $\tau$ . If  $h_i \geq \tau$  then  $b_i$  is considered an outlier because its distance from the rule boundary is high.

Since beliefs are discrete probability distributions, we use a specific distance measure dealing with such kinds of elements, namely, the *discrete Hellinger distance* ( $H^2$ ) (Hellinger 1909). This distance is defined as follow:

$$H^2(P, Q) = \frac{1}{\sqrt{2}} \sqrt{\sum_{i=1}^k (\sqrt{P_i} - \sqrt{Q_i})^2}$$

where  $P, Q$  are probability distributions and  $k$  is the discrete number of states in  $P$  and  $Q$ . An interesting property of  $H^2$  is that it is bounded between 0 and 1, which is very useful to define a meaningful threshold  $\tau$ . In Section 4 we discuss how we set this threshold for our experiments.

### 3.8 Shielding Decisions with XPOMCP

We present an extension that implements the unsatisfiable steps analysis presented in Section 3.7 directly into the POMCP algorithm. A complete description of this extension is available in (Mazzi, Castellini, and Farinelli 2021b). The methodology is summarized in Figure 4. We integrate the rules computed by XPOMCP directly into POMCP to preemptively prune undesired actions considering the current belief. The *shield* includes a set of rules trained as explained in Section 3.6 a set of samples generated as described in Section 3.7. To shield the action of the POMCP, we start by building a set of *legal actions*  $\mathcal{L}$  that satisfy the logical rules and can be performed on the current belief, and we force POMCP to only consider legal actions in the first step of the simulation. After a legal action is selected, the Monte-Carlo Tree Search is performed as usual. Notice that when

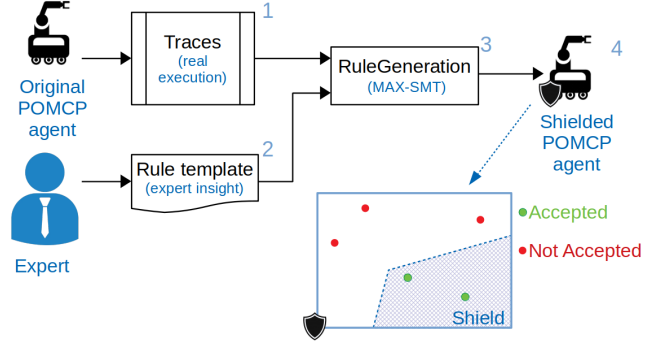


Figure 4: Shielding decisions with XPOMCP

---

#### Algorithm 2: Shielding

---

**Input:** a belief  $b$ , a shield  $s$ , safe action  $a_{safe}$   
**Output:** set of legal actions  $\mathcal{L}$

```

1  $\mathcal{L} \leftarrow \emptyset$ ;
2 foreach action  $a \in \mathcal{A}$  do
3   if  $a \notin s$  then
4      $\mathcal{L} \leftarrow \mathcal{L} \cup a$ ;
5   else if  $s.test\_constraints(b)$  then
6      $\mathcal{L} \leftarrow \mathcal{L} \cup a$ ;
7   else if  $\exists r \in s.Repr : H^2(b, r) < s.\tau$  then
8      $\mathcal{L} \leftarrow \mathcal{L} \cup a$ ;
9 if  $\mathcal{L} = \emptyset$  then
10   $\mathcal{L} \leftarrow \{a_{safe}\}$ ;
11 return  $\mathcal{L}$ ;

```

---

the original implementation of POMCP selects a particle in the simulation process, it assumes that the state encoded by the particle is the current state of the system (which for a POMDP is not observable) and thus the belief can only be considered in the first step.

With this mechanism, we can ensure that the rule of the shield is respected but we do not force POMCP to select a specific action, the best action is still decided using the regular POMCP but only among the legal ones. In more detail, as reported in Algorithm 2, for each possible action  $a$ , we consider  $a$  as a legal action if it satisfies at least one of these three conditions, namely, *i*) the shield does not define any rule (i.e., any restriction) for this action (line 3), *ii*) the current belief satisfies the constraints defined by an action rule for action  $a$  (line 5) *iii*) the Hellinger distance between the belief and the closest representative of the action rule for  $a$  is lower than the predefined threshold (line 7). These conditions could result in an empty set of legal actions  $\mathcal{L}$  (i.e., if the rules are very strict). In this case, it is important to define a default safe action  $a_{safe}$  that is used when no other action is possible. While this is a domain-specific requirement, it is reasonable to assume that most domains have such action (e.g., wait and listen to gather extra data, take low-risk action that yields low rewards). The computation of legal actions is performed only once for a simulation step since the current

belief does not change until a new observation is received from the real environment. Checking that the belief satisfies the constraints has a fixed cost, checking the  $H^2$  of the representative beliefs increases linearly with the number of beliefs. As shown in Section 4, this is a negligible cost, and the reduced number of actions that must be tested (because not all actions are now legal) can also slightly reduce the execution time.

## 4 Results

A full quantitative evaluation of the performance of XPOMCP is presented in (Mazzi, Castellini, and Farinelli 2021a). In this work, we provide a summary of the achieved results. Then, we focus on a specific use case for the proposed methodology that describes the behavior of a robotic agent in the velocity regulation domain. This is an iterative process, we start with a simple rule and we use the result to improve our description.

### 4.1 XPOMCP performance

To test the performance of XPOMCP in detecting unexpected behaviors, in (Mazzi, Castellini, and Farinelli 2021a) the methodology is used on different traces generated by POMCP in the *Tiger* and the *Velocity Regulation* domains. In both cases, hard-to-detect errors are injected into POMCP (by wrongly set one of its parameters), thus the traces contain wrong decisions. Using a proper rule template it is possible to detect the wrong decisions as unexpected behaviors. The results obtained by XPOMCP are then compared to *Isolation Forest (IF)*, a state-of-the-art anomaly detection algorithm. Similar to XPOMCP, IF does not require a training set and can be used to directly analyze a trace. In the *Tiger* domain it is possible to compute the correct policy, and this is used as ground truth to evaluate the performance of the two methodologies. The experiments show that XPOMCP yields a performance increase of up to 47% when compared to IF. XPOMCP achieves these results because it exploits the knowledge provided by the expert to better characterize anomalous behaviors.

### 4.2 Shielding performance

In (Mazzi, Castellini, and Farinelli 2021b), the rules generated by XPOMCP are used to build a shielding mechanism. The effectiveness of the shield is measured by computing the difference between the average discounted reward achieved by POMCP with and without the shield. The methodology is tested in *Tiger* and *Velocity Regulation*, with and without errors. Note that when a faulty POMCP was tested, the shield was trained using traces that contain errors. Table 1 shows a summary of the results. In the *Tiger* domain, the experiments show a performance improvement of up to 188.71% using the shielding mechanism. With a proper rule, it is possible to write a shield that recreates the correct policy, thus the POMCP performs well in all the instances. Similarly, in *Velocity Regulation* the shielding mechanism achieves a performance improvement of up to 136.53%. In this case, the correct policy is unknown, but it is possible to write a good shield using the explainability methodology presented in (Mazzi, Castellini, and Farinelli 2021a).

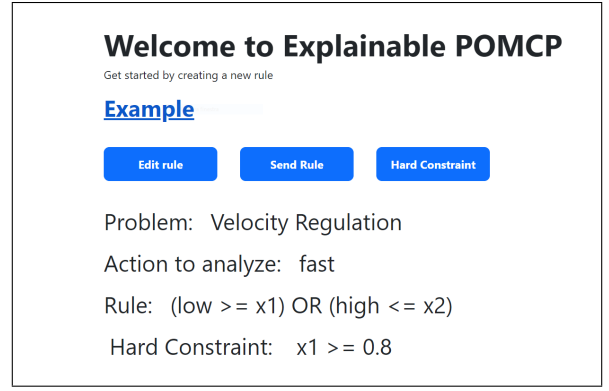


Figure 5: First template iteration

### 4.3 Detailed robotic case study



Figure 6: First Rule

**Iteration 1.** We start with a rule describing when the robot travels at maximum speed (i.e.,  $a = a_{fast}$ ). We expect that the robot should move at that speed only if it is confident enough to be in an easy-to-navigate segment. We express this with the template:

$$r_f : \text{select } a_{fast} \text{ when } p(low) \geq \bar{x}_1 \vee p(high) \leq \bar{x}_2;$$

$$\text{where } \bar{x}_1 \geq 0.8 \wedge$$

$$p(low) = \text{diff}(\text{distr}, \text{seg}, \text{clear}) \wedge$$

$$p(high) = \text{diff}(\text{distr}, \text{seg}, \text{heavy})$$

this template can be satisfied if the probability of being in a clear segment ( $p(low)$ ) is above a certain threshold or the probability of being in a heavily obstructed segment ( $p(high)$ ) is below another threshold. From our previous experience with this domain, we expect  $\bar{x}_1$  to be above 0.8, thus we add this information in the *where* statement. Finally, we add mappings between the belief on the difficulty of the current segment with the risk of moving in that segment (i.e.,

<i>c</i>	No Shield		Shield			
	return	time (s)	return	RI	time (s)	#SA
110	3.702(±0.623)	0.066(±0.027)	3.702(±0.623)	0.00%	0.065(±0.029)	0
80	3.593(±0.632)	0.067(±0.030)	<b>3.702 (± 0.623)</b>	3.03%	0.061(±0.027)	4
60	3.088(±0.673)	0.060(±0.025)	<b>3.702 (± 0.623)</b>	19.88%	0.061(±0.027)	121
40	-4.173(±1.101)	0.035(±0.017)	<b>3.702 (± 0.623)</b>	188.71%	0.052(±0.023)	647

a) Tiger

<i>c</i>	No Shield		Shield			
	return	time (s)	return	RI	time (s)	#SA
103	24.716(±3.497)	10.166(±0.682)	<b>26.045 (± 3.640)</b>	5.38%	10.118(±0.238)	7
90	18.030(±3.794)	10.173(±0.234)	<b>22.680 (± 3.524)</b>	25.79%	10.166(±0.241)	12
70	4.943(±5.260)	10.278(±0.234)	<b>8.970 (± 4.556)</b>	81.46%	10.377(±0.230)	51
50	0.692(±5.051)	10.374(±0.230)	1.638(±4.525)	136.53%	10.435(±0.336)	171

b) Velocity Regulation

Table 1: Experimental Results. Column *c* show the reward range parameter (a lower value generates more errors). The second (third) column shows the average return (time) achieved by the original POMCP and the relative standard deviation. The *Shield* section shows the average return and time achieved by POMCP using a shield (column four and six), values in bold show a statistically significant difference with respect to the shield counterpart (according to a paired t-test with 95% confidence level). Column *RI* shows the relative increase in performance between the two original and shielded POMCP. Finally, column *#SA* shows how many times the shield alters the decision during the execution

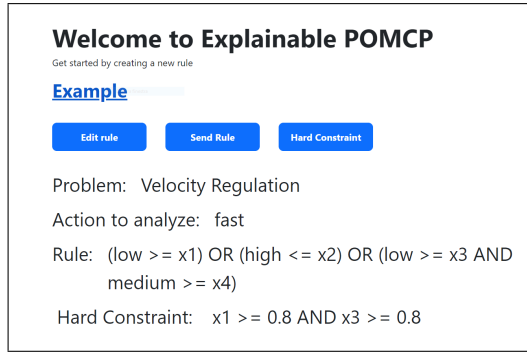


Figure 7: Second template iteration

*low* for clear segments, *medium* for slightly obstructed segments, *high* for heavily obstructed segments). For the sake of brevity, this mapping is omitted in the next rule templates.

Figure 5 shows how the rule can be written using the web app. By analyzing the specified trace, our methodology provides the rule:

$$r_f : \text{select } a_{fast} \text{ when: } p(low) \geq 0.8581 \vee p(high) \leq 0.0287;$$

that fails to satisfy 5 out of the 340 steps.

**Iteration 2.** The previous result is acceptable, but we try to further improve the precision of the rule by studying the unsatisfiable steps. Two of the unsatisfiable steps select the fast action even if the requirement of the rule is not satisfied. One of them (the robot moves at high speed with belief  $[p(low) = 0.747, p(medium) = 0.170, p(high) = 0.084]$ ) is far away from the rule (i.e., it present a behavior that is significantly different than the one described by the rule). The other unsatisfiable step is closer but cannot be described with this simple template, it is an approximation error. On

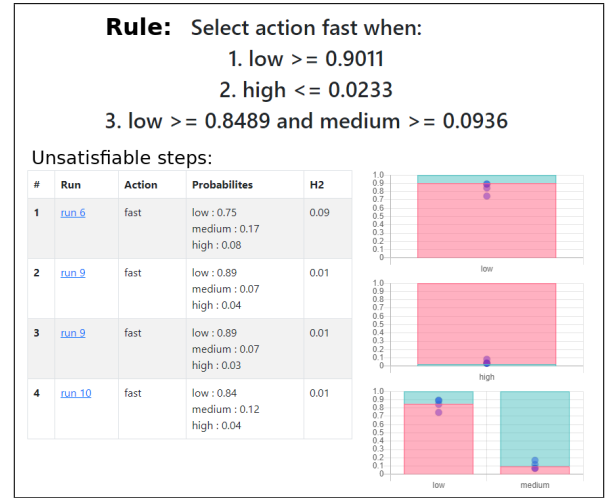


Figure 8: Second Rule

the other hand, three other unsatisfiable steps select medium speed even if their beliefs satisfy the boundaries of the rule. This means that our rule is too broad, and using this simple structure captures too many situations.

To improve the template, we add an additional literal  $(p(low) \geq \bar{x}_3 \wedge p(medium) \geq \bar{x}_4)$ , that use both difficulty low (i.e., clear segment) and medium (i.e., lightly obstructed) to describe the behavior of the policy. This is more complex, and we aim to use it to better describe some steps that are close to the rule but cannot be properly described. We obtain the new template (also presented in Figure 7):

$$r_f : \text{select } a_{fast} \text{ when} \\ p(low) \geq \bar{x}_1 \vee p(high) \leq \bar{x}_2 \vee \\ (p(low) \geq \bar{x}_3 \wedge p(medium) \geq \bar{x}_4); \\ \text{where } \bar{x}_1 \geq 0.8$$



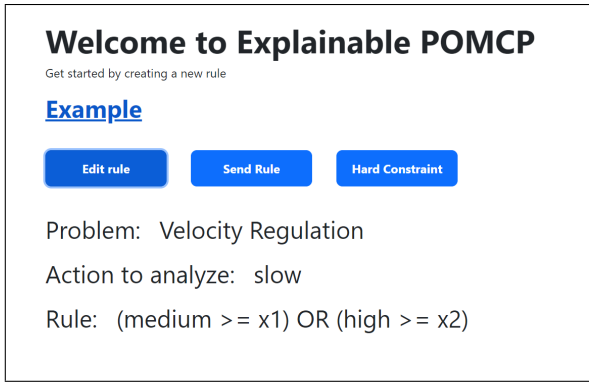


Figure 9: Third template iteration

and the rule (Figure 8):

$$r_f : \text{select } a_{fast} \text{ when} \\ p(low) \geq 0.9011 \vee p(high) \leq 0.0233 \vee \\ (p(low) \geq 0.8489 \wedge p(medium) \geq 0.0936);$$

that only fails to satisfy 4 steps. As in the previous iteration, one of the step (the robot moves at high speed with belief  $[p(low) = 0.747, p(medium) = 0.170, p(high) = 0.084]$ ) is far from the rule. This is a true outlier, a step that behaves unlike all the others explored by the POMCP. In this case, all the other three steps that cannot be satisfied use the same action of the rule, but they are also very close to the final rule. We can further refine the template to capture these other unsatisfiable steps, but we consider this result a good compromise between simplicity and correctness.



Figure 10: Third Rule

**Iteration 3.** We write a template to describe when the robot moves at slow speed. We identify two important situations that can lead the robot to move at slow speed *i*) the

robot is uncertain about the current difficulty (the belief is close to a uniform distribution), *ii*) the robot knows that the current segment is hard. We try to use  $p(medium) \geq \bar{y}_1$  and  $p(high) \geq \bar{y}_2$  this idea. The template is the following (also shown in Figure 9):

$$r_s : \text{select } a_{slow} \text{ when } p(medium) \geq \bar{y}_1 \vee \\ p(high) \geq \bar{y}_2;$$

that yields the rule:

$$r_s : \text{select } a_{slow} \text{ when } p_1 \geq 0.1920 \vee \\ p_2 \geq 0.0791$$

which fail to satisfy 28 out of 340 steps. Notice that the low value for  $\bar{y}_1$  and  $\bar{y}_2$  (i.e., 0.148 and 0.097) describes all the belief close to the uniform distribution. By analyzing the 28 unsatisfiable steps, we notice that 26 of them are situations in which the robot decides to move at medium speed (i.e.,  $a_{medium}$ ) even if the condition for moving at speed  $a_{slow}$  are satisfied (e.g, three of these steps have belief  $[p(clear) = 0.319, p(medium) = 0.342, p(high) = 0.338]$ ,  $[p(low) = 0.345, p(medium) = 0.337, p(high) = 0.318]$ , and  $[p(low) = 0.335, p(medium) = 0.333, p(high) = 0.332]$  respectively). This analysis tells us that the POMCP considers a worthy risk to move at medium speed (i.e., speed 1) even if it does not have a strong understanding of the current difficulty. If we consider this to be a non-acceptable risk, we should modify the design of POMCP, e.g., by increasing the number of particles used in the simulation.

**Iteration 4.** If we want to properly describe the current policy, we can create a rule that groups together action  $a_{slow}$  and  $a_{medium}$ , to convey the idea that there is a significant overlap between the two actions.

$$r_{s,m} : \text{select } a_{slow} \vee a_{medium} \text{ when} \\ p(medium) \geq \bar{y}_1 \vee p(high) \geq \bar{y}_2;$$

that yields the rule

$$r_{s,m} : \text{select } a_{slow} \vee a_{medium} \text{ when} \\ p(medium) \geq 0.124 \vee p(high) \geq 0.029$$

That fails to satisfy 6 steps. Many of which are unsatisfiable using both rules. It is possible to further refine the rule to improve the understanding of the behavior (e.g., to discover if there are some situation in which action  $a_{medium}$  is always preferred to action  $a_{slow}$ , or vice versa), but we consider this a good results that capture most of the behavior of the policy in a compact and readable representation.

## 5 Conclusions and future work

In this work, we present how to use a methodology that combines high-level indications provided by a human expert with an automatic procedure that analyzes execution traces to synthesize key properties of a policy in the form of rules. This work paves the way towards several interesting research directions. Specifically, we aim at improving the expressiveness of the logical formulas used to formalize the indications of the expert (e.g., by employing temporal logic), and to define a generic language that can be used to build new domains for the system.

## References

- Anjomshoe, S.; Najjar, A.; Calvaresi, D.; and Främling, K. 2019. Explainable Agents and Robots: Results from a Systematic Literature Review. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, AAMAS '19, 1078–1088. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems. ISBN 9781450363099.
- Bastani, O.; Pu, Y.; and Solar-Lezama, A. 2018. Verifiable Reinforcement Learning via Policy Extraction. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, 2499–2509. Red Hook, NY, USA: Curran Associates Inc.
- Bjørner, N.; Phan, A.-D.; and Fleckenstein, L. 2015. vZ - An Optimizing SMT Solver. In *Proceedings of the 21st International Conference on Tools and Algorithms for the Construction and Analysis of Systems - Volume 9035*, 194–199. Berlin, Heidelberg: Springer-Verlag. ISBN 9783662466803.
- Bunel, R.; Turkaslan, I.; Torr, P. H. S.; Kohli, P.; and Mudigonda, P. K. 2018. A Unified View of Piecewise Linear Neural Network Verification. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada*, 4795–4804.
- Cashmore, M.; Collins, A.; Krarup, B.; Krivic, S.; Magazzeni, D.; and Smith, D. 2019. Towards Explainable AI Planning as a Service. 2nd ICAPS Workshop on Explainable Planning, XAIP 2019.
- Cashmore, M.; Fox, M.; Long, D.; and Magazzeni, D. 2016. A Compilation of the Full PDDL+ Language into SMT. In *Proceedings of the Twenty-Sixth International Conference on Automated Planning and Scheduling*, ICAPS'16, 79–87. AAAI Press. ISBN 1-57735-757-4, 978-1-57735-757-5.
- Cassandra, A.; Littman, M. L.; and Zhang, N. L. 1997. Incremental Pruning: A Simple, Fast, Exact Method for Partially Observable Markov Decision Processes. In *Proceedings of the Thirteenth Conference on Uncertainty in Artificial Intelligence*, 54–61. Morgan Kaufmann Publishers.
- Castellini, A.; Marchesini, E.; Mazzi, G.; and Farinelli, A. 2020. Explaining the influence of prior knowledge on POMCP policies. In *Proceedings of the 17th European Conference on Multi-Agents Systems*.
- De Moura, L.; and Bjørner, N. 2008. Z3: An Efficient SMT Solver. In *Proceedings of the Theory and Practice of Software, 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, TACAS'08/ETAPS'08, 337–340. Berlin, Heidelberg: Springer-Verlag. ISBN 3540787992.
- Fox, M.; Long, D.; and Magazzeni, D. 2017. Explainable Planning. *CoRR* abs/1709.10256.
- Gunning, D. 2019. DARPA's Explainable Artificial Intelligence (XAI) Program. ii–ii. ISBN 978-1-4503-6272-6.
- Hellinger, E. 1909. Neue begründung der theorie quadratischer formen von unendlichvielen veränderlichen. *Journal für die reine und angewandte Mathematik* 136: 210–271.
- Huang, X.; Kwiatkowska, M.; Wang, S.; and Wu, M. 2017. Safety Verification of Deep Neural Networks. In *Computer Aided Verification - 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part I*, volume 10426 of *Lecture Notes in Computer Science*, 3–29. Springer.
- Kaelbling, L. P.; Littman, M. L.; and Cassandra, A. R. 1998. Planning and Acting in Partially Observable Stochastic Domains. *Artif. Intell.* 101(1–2): 99–134. ISSN 0004-3702.
- Katz, G.; Barrett, C.; Dill, D. L.; Julian, K.; and Kochenderfer, M. J. 2017. Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks. In Majumdar, R.; and Kunčák, V., eds., *Computer Aided Verification*, 97–117. Cham: Springer International Publishing. ISBN k78-3-319-63387-9.
- Kocsis, L.; and Szepesvári, C. 2006. Bandit Based Monte-Carlo Planning. In *Proc. ECML'06*, 282–293. Berlin, Heidelberg: Springer-Verlag.
- Langley, P.; Meadows, B.; Sridharan, M.; and Choi, D. 2017. Explainable Agency for Intelligent Autonomous Systems. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, AAAI'17, 4762–4763. AAAI Press.
- Mazzi, G.; Castellini, A.; and Farinelli, A. 2021a. Identification of Unexpected Decisions in Partially Observable Monte Carlo Planning: A Rule-Based Approach. In *accepted at the 21th International Conference on Autonomous Agents and MultiAgent Systems*, AAMAS '21.
- Mazzi, G.; Castellini, A.; and Farinelli, A. 2021b. Rule-based Shielding for Partially Observable Monte-Carlo Planning. In *Accepted at the 31th International Conference on Automated Planning and Scheduling*, ICAPS '21.
- Norman, G.; Parker, D.; and Zou, X. 2017. Verification and control of partially observable probabilistic systems. *Real-Time Systems* 53(3): 354–402.
- Papadimitriou, C. H.; and Tsitsiklis, J. N. 1987. The Complexity of Markov Decision Processes. *Math. Oper. Res.* 12(3): 441–450. ISSN 0364-765X.
- Setchi, R.; Dehkordi, M. B.; and Khan, J. S. 2020. Explainable Robotics in Human-Robot Interactions. *Procedia Computer Science* 176: 3057–3066.
- Silver, D.; and Veness, J. 2010. Monte-Carlo Planning in Large POMDPs. In Lafferty, J. D.; Williams, C. K. I.; Shawe Taylor, J.; Zemel, R. S.; and Culotta, A., eds., *Advances in Neural Information Processing Systems* 23, 2164–2172. Curran Associates, Inc.
- Wang, Y.; Chaudhuri, S.; and Kavradi, L. E. 2018. Bounded Policy Synthesis for POMDPs with Safe-Reachability Objectives. *ArXiv* abs/1801.09780.
- Zhu, H.; Xiong, Z.; Magill, S.; and Jagannathan, S. 2019. An Inductive Synthesis Framework for Verifiable Reinforcement Learning. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI 2019, 686–701. New York, NY, USA: Association for Computing Machinery. ISBN 9781450367127.

# Combining Temporal and Probabilistic Planning for Robots Operating in Extreme Environments

Jun Hao Alvin Ng<sup>1, 2\*</sup>, Yaniel Carreno<sup>1, 2\*</sup>, Yvan Petillot<sup>1</sup>, Ronald P. A. Petrick<sup>1</sup>

Edinburgh Centre for Robotics

<sup>1</sup>Heriot-Watt University, Edinburgh, EH14 4AS, United Kingdom

<sup>2</sup>University of Edinburgh, Edinburgh, EH8 9AB, United Kingdom  
{alvin.ng, y.carreno, y.r.petillot, r.petrick}@hw.ac.uk

## Abstract

To coordinate robots to achieve common goals requires reasoning about constraints and uncertainty present in complex and non-deterministic environments. In large-scale problems, this is computationally expensive making it impractical for online planning and execution. Furthermore, complete and accurate models are difficult to handcode while approximate models may result in unsound plans. We propose a decoupled framework composed of a centralised goal allocation algorithm and an online learning and planning module. The former reasons about robot capabilities and temporal constraints to coordinate multiple robots. The latter is a reinforcement learning agent that executes the temporal plan when possible; otherwise it follows a policy trained offline with an approximate model. The robot learns over time to improve its performance in future episodes. We demonstrate our approach on a real-world application involving a fleet of heterogeneous robots operating in an offshore energy environment.

## Introduction

In real-world robotics applications, planning provides solutions that enable autonomous and intelligent behaviour for robots to complete their goals in dynamic and uncertain environments. When multiple robots are required to achieve a goal, temporal planning can be used for coordination. Temporal planning considers temporal constraints and durative actions in the decision-making process to synthesise temporal plans comprising of concurrent actions from multiple robots. In addition, temporal planning provides long-horizon strategic planning while managing constrained resources (Bernardini, Fox, and Long 2017).

Most temporal planners consider only a single deterministic outcome of probabilistic actions which results in poor robustness during plan execution. Robotic applications in extreme environments are often probabilistically interesting (Little, Thiebaux et al. 2007)—*a planning problem that includes: (i) multiple goal trajectories; and (ii) at least one pair of distinct goal trajectories share a common sequence of outcomes except in the last state where there are distinct outcomes of the same action*—and reasoning over uncertainties can lead to robust plans. For instance, a robot operating in a hazardous offshore structure (see husky2 in Figure 1)

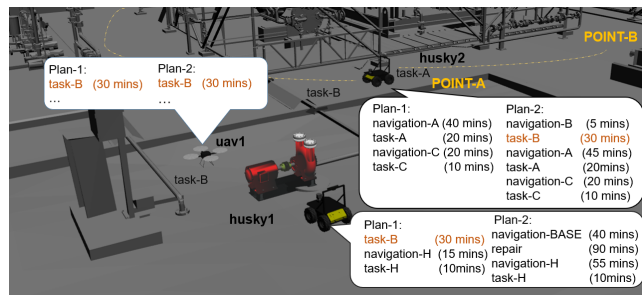


Figure 1: Overview of the mission environment where a fleet of robots executes temporal plans. Mission replanning considers action probability of failure which might force reallocation of goals amongst the fleet.

may choose a longer path to move from *point A* to *point B* if the path poses less risk to its safety. Probabilistic planners reason about probabilistic outcomes of actions or exogenous events but are unable to consider temporal constraints and, hence, are ill-suited for multi-agent planning. Existing work combining temporal planning and reasoning under uncertainty (Schillinger, Bürger, and Dimarogonas 2018; Bernardini, Fox, and Long 2017; Tsamardinos 2002; Zhang et al. 2017) does not fully address the requirements of our application of interest: deploying and coordinating a fleet of heterogeneous robots in an extreme environment with temporal uncertainty and probabilistic outcomes, without the requirement of a complete and accurate domain model.

This paper presents two main contributions. First, we propose a novel hybrid framework, TEMPORALPROB, which combines the advantages of temporal planning and probabilistic planning. A centralised multi-agent planning (MAP) module acts as a strategic long horizon planner coordinating multiple robots, and reasons with temporal constraints and robot capabilities. Each robot is a decentralized reinforcement learning (RL) agent which acts robustly in the face of uncertainty while learning over time to improve reasoning capabilities. The two-tier approach significantly reduces the search space on both levels. This decreases the computational cost, enabling online planning and execution which is essential for robots operating over long horizons in extreme

\*The first two authors have equal contribution.

environments with uncertainty.

The second contribution is the mapping of a temporal MAP problem to a set of single-agent probabilistic planning problems, one for each agent. The former considers temporal information while the latter considers probabilistic outcomes. We model the duration of executing actions as costs, and joint goals, which are goals that require robots’ coordination, as time-constrained goals such that the RL agent can optimise the makespan and reason over coordinated actions. Furthermore, the search spaces of the single-agent planning problems are reduced by considering only a subset of the state-action space of the MAP problem. This aids in reducing the sample complexity of the RL algorithm.

## Related Work

Our work involves temporal planning, probabilistic reasoning, reinforcement learning, and MAP. To the best of our knowledge, our framework is the first of its kind to solve problems in this space: temporal, probabilistic, multi-agent, and without the true model. However, we review related work which covers a combination of these areas.

**Temporal Planning.** Temporal planners (e.g., SGPlan (Hsu and Wah 2008), LPG-TD (Gerevini and Long 2006), and TFD (Eyerich, Mattmüller, and Röger 2009)) reason about action duration and temporal constraints, and can be used to solve MAP problems which require coordination and collaboration among the agents. However, these planners have not been widely introduced in MAP problems due to scalability issues. Some approaches have shown the ability to provide solutions while improving the initial plan output based on their cost function. For instance, POPF (Coles et al. 2010) uses the makespan—the time that elapses from the start of plan implementation to the end—as a cost function and is able to generate plans which coordinate agents to achieve goals. An extension to POPF is OPTIC (Benton, Coles, and Coles 2012) which introduces reasoning about preferences. OPTIC produces better plans by introducing soft deadlines and temporal preferences on plan trajectory. These temporal planners deal with deterministic domains and produce plans which cannot deal with unexpected situations, thus the need for replanning during plan execution arises.

**Temporal and Probabilistic Planning.** Temporal planning problems with uncertainty require probabilistic reasoning for plan optimality. (Schillinger, Bürger, and Dimarogonas 2018) addresses a multi-robot coordination problem to achieve a global goal. A deterministic finite-state automaton is used to model probabilistic outcomes of actions and temporal specifications of the goal. A goal allocation algorithm assigns an option to each robot. Our work considers temporal constraints of both actions and goals, handles multiple goals, and does not require a complete and accurate model. (Bernardini, Fox, and Long 2017) combines Monte Carlo methods, which reason over the probabilistic motion of a target, with temporal planning, which offers long-term strategic planning, to solve a search-and-tracking problem. We consider uncertainty in outcomes of task-level actions rather than in motion, and are interested in multi-agent coordination to complete a set of goals. (Tsamardinos 2002; Zhang

et al. 2017) reason about probabilistic dynamics and temporal uncertainty during plan execution. We deal with temporal uncertainty by learning the duration of actions from observations. (Zhang et al. 2017) coordinates robots to avoid conflicts such that each robot can complete its goal while we are interested in coordinated goals.

**Multi-Agent Planning and Learning.** Multi-agent RL (MARL) algorithms learn in the shared state-action spaces of all agents (Zhang, Yang, and Başar 2019) which are often prohibitively large and do not scale well with the number of agents. Thus, MARL algorithms typically have a high sample complexity. This is evident in work such as (Xinyi et al. 2019) which requires an excessive amount of training data. This is not practical in our application of robots operating in extreme environments as data collection is expensive and potentially unsafe. (Xinyi et al. 2019) trains a neural network with Q-learning to learn and encode a task allocation so that planning is computationally faster with a feed-forward operation. We achieve lower computational cost by decoupling MAP and goal allocation from single-agent planning and learning. We consider single-agent RL and use model-based RL which has a lower sample complexity than model-free methods (Buckman et al. 2018; Deisenroth and Rasmussen 2011; Finn, Abbeel, and Levine 2017).

## Preliminaries

Here we present relevant background on planning, model representations, and reinforcement learning.

**Markov Decision Process (MDP).** MDPs model fully-observable problems with uncertainty. A finite-horizon MDP is a tuple of the form  $\langle S, A, T, R, s_0, H, \gamma \rangle$  where  $S$  is a set of states,  $A$  is the set of actions,  $T : S \times A \times S \rightarrow [0, 1]$  is the transition function,  $R : S \times A \rightarrow \mathbb{R}$  specifies rewards for performing actions,  $s_0$  is the initial state,  $H$  is the planning horizon, and  $\gamma$  is the discount factor. Factored MDPs (Boutilier, Dearden, and Goldszmidt 2000) represent large, structured MDPs compactly. For example,  $T$  can be represented by dynamic Bayesian networks (DBN) which exploit the fact that the transition of a state variable often depends only on a small number of variables.

**Model-Based Reinforcement Learning (MBRL).** MBRL methods use a domain model, which is either known or learned, to approximate a policy  $\pi$  for a MDP. The expected return for  $\pi$  is computed using the Q-function.

**Goal Allocation Problem.** A goal allocation problem is defined by a tuple  $t := \langle R, RC, G, GC, X \rangle$ ; where  $R$  is a set of agents;  $RC$  is a set of capabilities;  $G$  is a set of goals;  $GC$  is a set of capabilities required to implement the goals; and  $X$  is a set of goal coordinates. The capabilities define skills that allow an agent to execute actions. For instance, the capability `canMove` is required to execute the action `navigate`. Each agent  $r \in R$  can have multiple capabilities.

**Planning Domain Definition Language (PDDL).** Our temporal planning problems are modelled using a variant of PDDL (McDermott et al. 1998), the de facto standard representation language supported by most modern planners.

PDDL provides a way of describing the components of a planning problem, i.e., how the world is structured, what actions are available, etc. Given our choice of planner, OPTIC, we will use the temporal constructs supported by PDDL2.1 (Fox and Long 2003) and later.

**Temporal Planning Problem.** Actions change the values of state fluents in a planning problem and are described by their preconditions and effects. For temporal planning, we adopt the full PDDL2.1 variant (Fox and Long 2003) with continuous change. A *temporal planning problem* is a tuple  $P_t := \langle P, V, A, I, G, T \rangle$ , where  $P$  is set of propositions;  $V$  is a set of fluents;  $A$  is a set of instantaneous and durative actions where each  $a_i$  is a tuple  $\langle a_{pre}, a_{eff}, a_{dur} \rangle$ ,  $a_{pre}$  is a set of conditions that must hold for the action to be applicable,  $a_{eff}$  is the set of action effects, and  $a_{dur}$  is a set of duration constraints;  $I$  is the initial state defined by the propositions and fluents ( $P \cup V$ );  $G$  is a set of goals,  $G : P \cup V$ ; and  $T$  is a set of time windows. Each time window is defined using timed initial literals (TILs) which define the time  $t$  at which particular propositions in  $P$  become true/false.

**Relational Dynamic Influence Diagram Language (RDDL).** RDDL (Sanner 2010) is a planning language for describing relational MDPs. Semantically, RDDL describes DBNs extended with an influence diagram. The domain file specifies object types, non-fluents, fluents, conditional probability functions (CPFs), and a reward function. The problem file specifies objects, the initial state, and values of non-fluents.

Our framework performs two levels of reasoning. First, it considers a MAP problem, modelled as a temporal planning problem described in PDDL, and performs goal allocation. Next, it considers a single-agent planning problem, modelled as a MDP and described in RDDL, and learns a policy using MBRL.

### System Framework Definition

Temporal planners based on PDDL2.1, like OPTIC, support the implementation of planning problems with hard constraints and joint goals. Joint goals are achieved by coordinated actions from multiple robots. These planners generate temporal plans which distribute the goals over multiple agents to minimise the makespan. However, the resulting goal allocation quality is often quite poor (Carreno, Petrick, and Petillot 2019). This can be resolved with planners which support preferences to guide the search based on cost functions. However, solutions tend to be domain-dependent which affect the generalisation of the approach. Instead, we address this issue by decoupling the goal allocation process from the planning process. We propose a goal allocation algorithm which maximises the number of completed goals, and minimises a cost function involving the spatial distances between goals, makespan, and aborted goals.

Another limitation of temporal planners is the lack of reasoning over every probable outcome of actions. Current approaches to temporal planning are deterministic and do not consider unexpected situations during plan execution. This necessitates replanning. We address this issue by combining temporal planning, which deals with multi-agent coord-

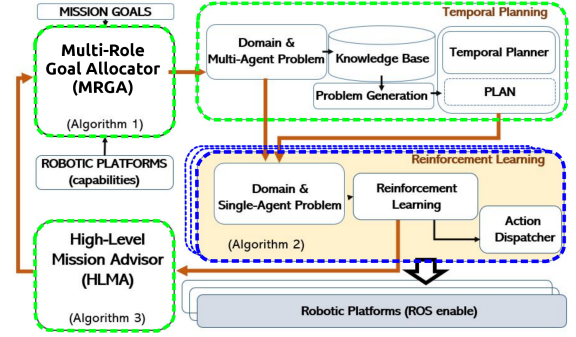


Figure 2: TEMPORALPROB framework for planning, acting, monitoring, and replanning. The modules outlined in green (blue) dotted lines represent high-level (low-level) reasoning. High-level reasoning deals with temporal and multi-agent planning while low-level reasoning deals with online learning and single-agent planning and replanning due to probabilistic uncertainty.

dination, and model-based reinforcement learning (MBRL), which deals with probabilistic dynamics. Since it is difficult to handcode domain models for complex, real-world applications, MBRL is selected over probabilistic planners as it does not require the true model. We use a deterministic high-level (H-L) domain for temporal planning and a probabilistic low-level (L-L) domain for MBRL. We discuss these domains in the Implementation Scenario section.

**TEMPORALPROB Framework.** We introduce a novel framework, TEMPORALPROB, as shown in Figure 2, which integrates a multi-role goal allocator algorithm (MRGA) (Carreno et al. 2020), temporal planning, and reinforcement learning. TEMPORALPROB performs planning, plan execution, replanning, and learning in an integrated cycle where observations from plan execution are utilised to improve planning performances. Firstly, a MAP, described in PDDL, is augmented with information from MRGA which allocates goals to robots by considering the capabilities of the robots. The augmented problem is given to the temporal planner, OPTIC, which synthesises a multi-robot plan. We refer to this phase of planning as the high-level (H-L) phase as it abstracts away probabilistic outcomes which could be encountered during plan execution.

Secondly, the H-L domain, problem, and plan are mapped to L-L domains, problems, and plans, one for each robot. We assume that the true probabilistic dynamics of the environment are unknown and thus rely on handcoded, approximate models which neglect probabilistic dynamics. These approximate models are generative: given a state-action pair  $(s, a)$ , it predicts the successor state  $(s')$  and immediate reward  $(r)$ . Thirdly, each robot would execute its L-L plan. When unexpected events render the L-L plan inapplicable (i.e., the precondition of the action suggested by the L-L plan is not satisfied in the current state), the robot then acts according to a greedy policy based on the Q-function which is trained with simulated observations  $(s, a, r, s')$  produced by the generative domain model. Each robot adapts to unexpected situa-



---

**Algorithm 1:**  $TA(M_m^r, NS, G', R, G, R_c, G_c, R_{cap}, G_{cap})$ 

---

**Input:**  $R$ : Robot Set  
 $G$ : Goal Set  
 $R_c$ : Coordinates of Robot Set  
 $G_c$ : Coordinates of Goal Set  
 $R_{cap}$ : Robots Capability Set  
 $G_{cap}$ : Goal Capability Requirement Set  
 $M_m^r$ : Goal Makespan Set  
 $G'$ : Number of Goals in the Mission  
 $NS$ : Number of Sensor Sets

```
1 begin
2   allocate( $G, R, R_{cap}, G_{cap}$ )
3    $\{GA_{INIT}, RA_{INIT}, C_{sol}, R_{sol}\} \leftarrow \emptyset$ 
4    $[C_{sol}, R_{sol}] = \text{cluster\_cal}(G, R, R_c, G_c)$ 
5    $RA_{INIT} = \max\{\text{weight\_cal}(RG, C_{sol}, R_{sol})\}$ 
6   regions\_dist( $RA_{INIT}, R_c, C_{sol}, G_c$ )
7   allocate\_firstGoal( $GA_{INIT} \leftarrow G$ )
8    $GA_{FINAL}(G) \leftarrow GA_{INIT}$ 
9   allocateGoal( $R, G, NS, G', M_m^r, \text{dist}$ )
10  return  $\text{transformPDDL}(GA_{FINAL})$ 
```

---

tions and attempts to complete as many of its allocated goals as possible within the makespan of the H-L plan.

Lastly, the observations acquired during plan execution are used to improve the L-L domains and to update the expected duration of the actions. At the end of the mission, the robot feeds back information to the High-Level Mission Advisor (HLMA). When feedback from every robot is received, HLMA updates MRGA on the robots' capabilities, observed duration of actions, and status of goals (completed or not). If there are unsatisfied goals, MRGA re-allocates these goals to the robots and the cycle repeats.

TEMPORALPROB reduces the computational costs by restricting the search space. The H-L reasoning deals with the combined state-action spaces of multiple agents and considers multi-agent coordination, temporal constraints, and robot capabilities. However, it neglects the probabilistic nature of actions and only considers their most probable outcomes. The L-L reasoning deals with the state-action spaces of single agents and considers probabilistic outcomes. In essence, we decompose a MAP with temporal constraints into several single-agent probabilistic planning problems. Moreover, we consider the plan execution for each robot. If required, each robot is empowered to deviate from its L-L plan to adapt to unexpected situations rather than replan at the H-L, starting off by having MRGA re-allocate goals, which is potentially impractical. For example, if a robot requests replanning at the H-L while another robot is performing an inspection, this forces the latter to abort the inspection as the H-L plan, and thus the L-L plans, could have changed. The remainder of this section is presented in a manner following the algorithmic workflow of TEMPORALPROB as shown in Figure 2.

**Multi-Role Goal Allocator (MRGA).** We address the goal allocation problem in the context of a fleet of heterogeneous robots considering the approach defined in (Carreno

et al. 2020). We propose a planner agnostic solution, called MRGA, which guides the search by optimising goal allocation. This is outlined in Algorithm 1. MRGA considers the capabilities of each robot and the capabilities required to complete each goal (line 2) to define and allocate the set of goals each robot can complete. Next, the goals are clustered into regions (lines 3 to 4). In line 3,  $GA_{INIT}$  represents the initial set of goals allocated to each robot (one per robot);  $RA_{INIT}$  describes the set of robots allocated to each cluster;  $C_{sol}$  is a set of clusters of goals; and  $R_{sol}$  is a set of integers representing the number of goals each robot can achieve in each cluster in  $C_{sol}$ . The number of regions are constrained to be equal to the number of robots. The approach establishes a rank amongst robots with respect to the number of goals they can implement in each region (line 5).  $RA_{INIT}$  records which robot should be sent to each region (initially) considering the number of goals they can achieved in that region.  $RG$  represents the set of goals each robot can achieve. Each robot is then assigned to a region (line 6) after considering two criteria: (i) the number of goals a robot has the required capabilities to complete, and (ii) the distance between the robot's current position and the closest goal. Then, MRGA allocates a goal in the region of each robot to the robot; this goal has the shortest path with respect to the robot's initial position (line 7). The allocated goals are removed from the set of incomplete goals in  $G$  (line 8) and stored in  $GA_{FINAL}$ . The remaining goals are allocated to robots while considering the total makespan, the robots and goals coordinates, and the redundancy of the capabilities of the robots required to complete the goals (line 9).

The allocation is sequential which allows MRGA to evaluate the cost each robot incurs in achieving a particular goal. Robots are not restricted to a region and can move to other regions if they are required to complete goals in those regions. This is the case when a robot possesses the necessary capability to achieve a goal located in a different region to the one the robot is initially operating in. The goal allocation  $GA_{FINAL}$  is transformed into PDDL fluents (i.e.,  $\text{robot\_can\_act } ?r - \text{robot } ?wp - \text{poi}$ ) (line 10) which are included in the H-L domain and problem files (see the Implementation Scenario section for more details). In the PDDL domain, the fluent  $\text{robot\_can\_act}$  is included in the precondition of actions to constrain the selection and execution of actions for the appropriate robot. Since PDDL planners do not deal with the goal allocation problem directly, MRGA decreases overall planning times and improves the optimality of the resulting plans.

**Temporal Planning.** The H-L domain and problem augmented with the information provided by MRGA are given as inputs to the temporal planner, OPTIC, which produces a temporal H-L plan. The H-L plan specifies the actions each robot executes at a time instance. We decompose the H-L plan into individual plans, one for each robot and map each plan to a L-L plan. The L-L plans are dispatched to the robots which then proceed to execute their L-L plans.

**Online Planning, Execution, and Learning.** During plan execution of the L-L plan, an expected situation could invalidate the plan which necessitates replanning. For example,

the robot’s manipulator arm might become damaged forcing the robot to return to the base for repairs. This leads to two consequences. Firstly, the robot is now at the base and is potentially further away from the goals it has been allocated. Secondly, joint goals which require the coordination of the robot would not be completed unless the repair is done. One solution is to replan at the H-L: MRGA reallocates the goals and OPTIC generates a new H-L plan. However, this could be impractical in real-world applications. If goals are reallocated whenever a robot experiences an unexpected outcome, it would lead to situations where robots are frequently disrupted in the midst of executing some actions and thus never achieve any useful work.

Following this observation, we propose to replan at the L-L where every robot is capable of online replanning. The robot deliberates over whether it should return to the base to repair its manipulator arm. This might be undesirable if the mission time exceeds the start time of a joint goal after the repair. However, if the probability of success with a damaged manipulator arm is low, then the robot might focus on completing other goals which does not require manipulation. Alternatively, it could choose to prioritise joint goals. A probabilistic planner is unable to reason about these as we assume an incomplete and deterministic model is available initially. Hence, a learning approach such as MBRL is used.

We describe our approach in detail with Algorithm 2 which is a MBRL algorithm. It takes as input a generative model  $\mathcal{M}$ , a problem instance which specifies the initial state ( $s_0$ ) and the goals ( $G$ ) allocated to the robot, joint goals ( $\mathcal{G} \subseteq G$ ), and a L-L plan ( $A$ ) which is mapped from the H-L plan given by the temporal planner. The planning problem is represented as an MDP with a continuous time dimension  $t$  for mission time, ( $S, A, T, R, s_0, H, \gamma, t$ ). This shares similarity with time-dependent MDPs (Boyan and Littman 2001) though we are only considering time-dependent rewards for joint goals. Algorithm 2 starts by initialising a parameterised Q-function,  $Q_\theta$ , with simulation training (lines 6-9) using  $\mathcal{M}$  which predicts the successor state, reward ( $r_t$ ), and execution duration ( $\Delta t$ ) (line 7). This is similar to Dyna (Sutton et al. 2008). During simulation training, actions are selected with an epsilon-greedy policy,  $\pi_\theta^{eps}$ . The reward (or cost) of executing an action is  $-\Delta t$ , and a reward of +100 is given for each goal achieved while no reward is given for a joint goal if  $t$  exceeds its start time (or its time constraint).

$Q_\theta$  is approximated as a linear function of the weight vector  $\theta$  and a set of basis functions  $\mathcal{F}$ . The basis function is represented by features which are conjunctive ground state fluents. The initial set of features comprises every state fluent. New features, which are conjunctions of any two existing features, are added incrementally using iFDD+ (Geramifard et al. 2011), an online feature discovery algorithm. The weight vector is updated with Double Q-learning (Hasselt, Guez, and Silver 2016) (line 9). The mission time  $t$  (line 8) is used to determine if any joint goal  $g \in \mathcal{G}$  can no longer be achieved (line 7). This happens when  $t$  exceeds the start time of the joint goal. The simulation training is computationally expensive but can be done offline before the robot starts its mission.

After simulation training, the robot executes  $A$  sequen-

---

**Algorithm 2:** MBRL( $\mathcal{M}, \mathcal{P}, \mathcal{G}, A, H_{sim}, H_{plan}, H_{mk}, N$ )

---

**Input:**  $\mathcal{M}$ : Approximate, Generative Model  
 $\mathcal{P}$ : Problem Instance with initial state  $s_0$  and set of goals  $G$   
 $\mathcal{G}$ : Joint Goals  
 $A$ : Plan  
 $H_{sim}$ : Simulated Horizon  
 $H_{plan}$ : Plan Horizon  
 $H_{mk}$ : Maximum Allowable Makespan  
 $N$ : Number of Simulations

```

1 begin
2    $\theta \leftarrow \mathbf{0}$ 
3    $\mathcal{F} \leftarrow \text{initialize\_features}(\mathcal{M}, \mathcal{P})$ 
4   for  $l$  to  $N$  do
5      $t = 0$ 
6     for  $i = 1$  to  $H_{sim}$  do
7        $s_i, r_i, \Delta t \leftarrow$ 
8          $\text{simulate}(\mathcal{M}, \mathcal{G}, G, s_{i-1}, \pi_\theta^{eps}(s_{i-1}), t)$ 
9        $t \leftarrow t + \Delta t$ 
10       $\mathcal{F}, \theta \leftarrow$ 
11         $\text{update}(\mathcal{F}, \theta, s_{i-1}, \pi_\theta(s_{i-1}), r_i, s_i)$ 
12     $t = 0$ 
13     $\text{Follow} = \top$ 
14    for  $i = 1$  to  $H_{plan}$  do
15      if  $\text{Follow}$  then
16         $a \leftarrow \text{follow\_plan}(A)$ 
17        if  $a$  is not applicable in  $s_{i-1}$  then
18           $\text{Follow} = \perp$ 
19        if  $\neg \text{Follow}$  then
20           $a \leftarrow \pi_\theta^{greedy}(s_{i-1})$ 
21           $s_i, \Delta t \leftarrow \text{act}(s_{i-1}, a)$ 
22           $t \leftarrow t + \Delta t$ 
23           $\mathcal{M} \leftarrow \text{update\_model}(\mathcal{M}, s_{i-1}, a, s_i, \Delta t)$ 
24          if no uncompleted goals or  $t \geq H_{mk}$  then
25            return  $\mathcal{M}, s_i, t$ 
26  return  $\mathcal{M}, s_{H_{plan}}, t$ 

```

---

tially (line 14). If an unexpected outcome occurs which renders the next action in  $A$  to be inapplicable (line 15), the robot then follows a greedy policy based on  $Q_\theta$  (line 18). Exploitation is desired for safety reasons and because  $Q_\theta$  is updated only during simulation training. The advantage of following a policy rather than a plan is the reduced computational time which makes it attractive for online execution. The latter requires replanning whenever an unexpected state is reached. Since the policy is trained with  $\mathcal{M}$  which is often an approximation of the true model, the policy could be suboptimal or unsound in regions of the state spaces where  $\mathcal{M}$  is a poor predictor. This is an issue known as distribution shift. Recent works have proposed solutions to deal with this (Janner et al. 2019; Yu et al. 2020; Wan et al. 2019; Abbeel, Quigley, and Ng 2006). A plausible solution is to terminate the algorithm when a state that is not seen during simulation training is visited. We leave the issue of distribution shift for

---

**Algorithm 3:** ADVISOR( $R, \mathcal{P}, RL, NR, R_{cap}, M_m^r$ )

---

**Input:**  $R$ : Robot Set  
 $\mathcal{P}$ : Initial Problem  
 $RL$ : Reinforcement Learning Advice Set  
 $NR$ : Number of Advice Sets  
 $R_{cap}$ : Robots Capability Set  
 $M_m^r$ : Goal Makespan Set

```
1 begin
2   while not  $NR \leftarrow \emptyset$  do
3     for  $0$  to  $\text{length}(R)$  do
4       acquiredMod(nr):
5          $\text{append\_modifications}(RL, r)$ 
6       modify_robotLoc( $\mathcal{P}, R$ )
7       modify_problemDur( $\mathcal{P}, R$ )
8       modify_Goal( $\mathcal{P}, R$ )
9       reason_Repairs( $\mathcal{P}, R$ )
10    return  $M_m^r, R_{cap}, G$ 
```

---

future work as it is not within the scope of this paper.

The action is executed (line 19) and the resulting observation is used to update  $\mathcal{M}$  to provide more accurate predictions (line 21) and thus resulting in a closer approximation of  $Q_\theta$  to the optimal Q-function. A model-learning algorithm such as LFIT (Martínez et al. 2015) can be used to learn a domain model represented in RDDDL. The current state and mission time are inputs to the temporal planner for replanning, if necessary. The mission is terminated (line 22) if (1) there are no remaining goals, (2)  $H_{plan}$  number of actions executed, or (3) the mission time exceeds  $H_{mk}$ , the makespan of the H-L plan. The last condition is necessary to avoid having robots idle for a long period of time after completing their missions due to a robot's prolonged mission—H-L replanning is done only after all robots have completed or terminated their missions.

**High-Level Mission Advisor (HLMA).** The High-Level Mission Advisor (HLMA) acts as a bridge between MRGA and MBRL. This is detailed in Algorithm 3. The algorithm considers the feedback, or advice, from MBRL and terminates after all advice are considered. Each modification, where  $nr$  represents the modification number, is evaluated over the set of robots  $R$  (line 3) to determine the advice for each robot  $r \in R$  (line 4). Having this information distributed over the fleet of robots, HLMA generates a H-L problem where the locations of robots (line 5), duration of actions (line 6), the status of goals (completed or failed) (line 7), possible reasons for failure to complete a goal, and the additional time required to complete failed goals (line 8) are updated or included in the H-L problem. MRGA then decides if the goal should be allocated to another robot to minimise the makespan of the H-L plan. When all advice are considered, HLMA returns the analysis of the goal makespan set  $M_m^r$ , robot capabilities  $R_{cap}$ , and unsolvable goals  $G$  (line 9). These are used in the next goal allocation using MRGA (Algorithm 1). MRGA updates the capabilities of robots when a goal fails to be achieved by considering the time each robot

requires to achieve an unsatisfied goal. Thus, HLMA forces MRGA to reallocate unsatisfied goals such that the total makespan is minimised. MRGA updates the capabilities,  $cap$ , of a robot  $r$  with the equation  $fk(r, cap) = \min [fd(R_{cap}, r, cap), \max_{(i,d) \in M_R(r)} (t_{min}(g) + d)]$ , where  $fk$  is the set of goals initially allocated to a robot,  $fd$  is the set of unsatisfied goals which can be deleted from  $fk$ , and  $fd(R_{cap}, r, cap)$  is a function which removes capabilities from the set of capabilities  $R_{cap}$  for robot  $r$  when the time  $(t_{min}(g) + d)$  to achieve the goal  $g$  is not the minimum. HLMA utilises the L-L problem to modify/generate a set of instances in the H-L domain. MRGA considers this updated H-L domain to allocate any remaining unsatisfied goals.

**Example for H-L Replanning.** HLMA improves the quality of plans from replanning by considering failures during plan execution that could affect the optimality of goals allocation. Figure 1 shows an example where tasks in an initial H-L plan (Plan-1) is distributed amongst a fleet of three robots. A terrestrial robot (husky1) fails to inspect the condition of an industrial motor (task-B) due to a damaged camera. This task requires coordination with an aerial robot (uav1) which supervises husky1 while it inspects the motor. Replanning is required due to the failure. HLMA considers the additional time required for husky1 to return to the base for repairs, which is predicted by  $\mathcal{M}$ , to advise MRGA on the reallocation of goals. Plan-2 (in Figure 1) shows task-B is reallocated to husky2 as a consequence of HLMA's advice (i.e., the time incurred to repair husky1 is larger than that of having husky2 achieve the goal).

## Implementation Scenario

Our application of interest is the autonomous maintenance of an offshore oil rig using a fleet of robots which operate concurrently (see Figure 1). This requires a considerable number of sensors, actuators, and coordination between robots. For instance, a UAV needs to supervise a husky while the latter is manipulating a valve at an area where there is a high risk of explosion. In this section, we describe the H-L and the L-L domains, the mapping from a H-L domain (problem) to a L-L domain (problem), and the decomposition of a H-L plan to a set of L-L plans. The H-L domain models multiple robots and is used in MRGA and in temporal planning. It is deterministic and does not consider probable outcomes due to probabilistic actions. The L-L domain models probabilistic actions which allows reasoning over every probable outcome during plan execution.

**High-Level (H-L) Domain.** The H-L domain is described in PDDL. In addition to state fluents which describe the state of the robot and the environment, we introduce additional state fluents to define (i) robot capabilities to execute different goals based on their sensory system, (ii) robot availability, and (iii) restrictions associated with the positions the robots can reach in the environment. For example, the fluent  $robot\_can\_act(?r ?wp)$  constrains robot  $?r$  to a set of locations  $?wp$  where it can be. We use functions to represent energy and data resources a robot has, and distances. For example,  $distance\_intime(?wpi ?wpi)$  defines the time robot  $?r$



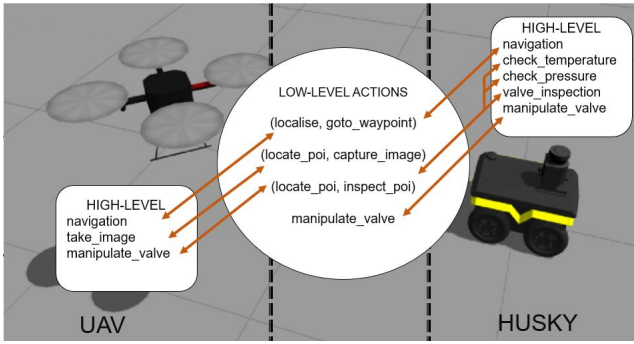


Figure 3: The mapping from a H-L action for a husky or a UAV to a sequence of L-L actions. More than one H-L action can be mapped to a L-L action, however, the grounding of these L-L actions will be different.

at  $?wpi$  takes to reach  $?wpf$ . The values of functions are updated by HLMA when new observations are acquired. There are six H-L actions, all of which are deterministic and durative, for a robot  $?r$ :

- *navigation* ( $?r ?wpi ?wpf$ ):  $?r$  moves from location  $?wpi$  to  $?wpf$ ,
- *take\_image* ( $?r ?s ?wp$ ):  $?r$  captures an image of the location  $?wp$  with its camera  $?s$ ,
- *check\_temperature* ( $?r ?s ?wp$ ):  $?r$  measures the temperature at location  $?wp$  with its sensor  $?s$ ,
- *check\_pressure* ( $?r ?s ?wp$ ):  $?r$  measures the pressure at location  $?wp$  with its pressure sensor  $?s$ ,
- *valve\_inspection* ( $?r ?s ?wp$ ):  $?r$  inspects the valve at location  $?wp$  with its camera  $?s$ , and
- *manipulate\_valve* ( $?r1 ?r2 ?s1 ?s2 ?wp1 ?wp2$ ): a husky  $?r1$ 's manipulator arm  $?s1$  turns the valve at location  $?wp1$  while a UAV  $?r2$  at location  $?wp2$  records the process with its camera  $?s2$ .

Our application involves two heterogeneous robots, the husky and the UAV, which have different sensors onboard. Thus, some PDDL actions and fluents could apply to one type of robot and not the other. The duration of the H-L actions are determined from data collected in past experiments involving real robots.

**Low-Level (L-L) Domain.** The probabilistic dynamics in our application of interest are (1) the robot may lose localisation while going to another waypoint, (2) its camera may lose calibration at any point, and (3) its manipulator arm (for Huskies) might be damaged while navigating to another waypoint due to collision. To model these probabilistic dynamics, we use RDDDL which has a richer representation than Probabilistic PDDL (PPDDL) (Younes and Littman 2004). Both RDDDL and PPDDL cannot represent temporal aspects, hence, we map the H-L actions to non-durative L-L actions, as shown in Figure 3. The L-L actions which a robot  $?r$  can execute are:

- *capture\_image*( $?r ?poi$ ):  $?r$  takes an image of  $?poi$ ,

- *locate\_poi*( $?r ?poi$ ):  $?r$  search for  $?poi$  at its current location,
- *inspect\_poi*( $?r ?poi$ ): a husky  $?r$  inspects the temperature, pressure, or value, or a UAV  $?r$  records a husky manipulating the valve  $?poi$ ,
- *localise*( $?r$ ):  $?r$  localises itself,
- *goto\_waypoint*( $?r ?wpi ?wpf$ ):  $?r$  navigates from location  $?wpi$  to  $?wpf$ ,
- *manipulate\_valve*( $?r ?poi$ ): a husky  $?r$  manipulates the valve  $?poi$ ,
- *repair\_manipulator*( $?r$ ): a husky  $?r$  repairs its manipulator arm, possibly with external aid, and
- *calibrate\_camera*( $?r$ ):  $?r$  calibrates its camera, possibly with external aid.

We introduce the actions *localise*, *repair\_manipulator*, and *calibrate\_camera* to handle unexpected situations. An ill-calibrated camera reduces the probability of success of *inspect\_poi* and *manipulate\_valve*, while a damaged manipulator arm reduces the probability of success of *manipulate\_valve*. A robot can only calibrate its camera or repair its manipulator arm at the base. If it loses localisation, it cannot execute any other actions except *localise*.

We consider and model temporal elements of the H-L domain in the L-L domain. First, the costs of L-L actions represents the duration of executing the actions. A RL algorithm which maximises the expected return will therefore learn policies which minimise the makespan. Second, joint goals are represented as fluents which must be made true by the time instances as defined by the end time of the H-L action *manipulate\_valve* in the temporal plan. Robots fail to complete a joint goal if the mission time exceeds the start time of the joint goal. A joint goal of manipulating a valve is achieved when a husky executes *manipulate\_valve* and a UAV executes *inspect\_poi* concurrently. Either robot can execute the joint action before the start time of the joint goal. When this happens, the robot simply idles till the start time. Replanning at the H-L is required to coordinate robots to complete any remaining joint goal.

**Mapping between H-L and L-L Representations.** We use three variants of the L-L domain in our experiments. The first variant is the true model which is used by the simulator to return a successor state during plan execution. This model is not made known to the planners. The second variant is a deterministic model which is an approximation of the true model and only predicts the most probable outcome of probabilistic actions. Both variants are handcoded. The third variant is a model learned from observations collected previously using the model learner from (Martínez et al. 2015). The L-L problems and L-L plans are mapped from the H-L problem and temporal plan. We illustrate the mapping from a temporal plan for a H-L problem used in our experiments (see Experiments and Results section) to L-L plans.

To generate the L-L plan for *husky2*, we extract H-L actions from the temporal plan which involve *husky2*. Each of these H-L actions is mapped to L-L actions as in Figure 3. The first *navigation* action is mapped to *localise*

H-L	Time: (Action)	[Duration]
	0.00: navigation(h2 wpg1 wpg52)	[166.35]
	0.00: navigation(h1 wpg0 wpg31)	[115.18]
	0.00: navigation(uav1 wpa0 wpa35)	[111.50]
	...	
	166.35: check_pressure(h2 p-sensor1 wpg52)	[20.00]
	268.04: valve_inspection(h2 c.h1 wpg35)	[50.00]
	<b>318.04: man_valve(h2 uav1 c.uav0 wpg35 wpa35)</b>	<b>[30.00]</b>

**L-L (Action) - h2**

```

localise(h2)
goto_waypoint(h2 wpg1 wpg52)
locate_poi(h2 pressure_wpg52)
inspect_poi(h2 pressure_wpg52)
...
manipulate_valve(h2 valve_wpg35)

```

**L-L (Action) - uav1**

```

localise(uav1)
goto_waypoint(uav1 wpa0 wpa35)
...
locate_poi(uav1 valve_wpa35)
inspect_poi(uav1 valve_wpa35)

```

Figure 4: Fragment of a H-L temporal plan for a UAV (uav1) and two Huskies (h1, h2) and the fragments of the L-L plans which are mapped from the H-L plan (the L-L plan for husky1 is not shown).

and *goto\_waypoint* if the robot is not localised in the initial state, and subsequent *navigation* actions are mapped to *goto\_waypoint*. *check\_pressure(husky2 pressure\_analyser1 wpg52)* is mapped to *locate\_poi(husky2 pressure\_wpg52)* and *inspect\_poi(husky2 pressure\_wpg52)*—the robot needs to locate the valve before it can approach the valve for inspection. This low-level detail is abstracted in the H-L domain but is useful for implementation as the actions relate to different actuation commands. *inspect\_poi* does not consider the sensor *pressure\_analyser1* in its arguments since reasoning about robot capabilities is not required at L-L. We introduce a new object *pressure\_wpg52* to represent a *poi* at *wpg52* to avoid ambiguity—the L-L actions *locate\_poi* and *inspect\_poi* relate to several possible H-L actions (i.e., *check\_temperature*, *check\_pressure*, and *valve\_inspection*), and if there are more than one *poi* at the same location, there is a need to differentiate if the robot is inspecting the pressure, temperature, or valve.

The joint goal of manipulating the valve *valve\_wpg35* is allocated to *uav1* and *h2* by MRGA. The actions which achieve this joint goal are highlighted in red in Figure 4. The H-L joint action *manipulate\_valve* (abbreviated as *man\_valve* in the figure) achieves this joint goal and requires the coordination of *uav1* and *h2*. This H-L action is mapped to the L-L actions *manipulate\_valve(h2 valve\_wpg35)* and *inspect\_poi(uav1 valve\_wpa35)*. Both robots have to execute their L-L actions by the mission time  $T = 318.04$  to achieve the joint goal. Thus, a joint goal has a time constraint and is added to  $\mathcal{G}$  in Algorithm 2. If one of the robots attempts to achieve the joint goal before  $T$ , it will have to wait for the other robot. If either robot did not attempt the joint goal (by executing their respective L-L actions) by  $T$ , the joint goal can no longer be achieved until H-L replanning is done. Both

robots will no longer follow their L-L plans and will adapt to the situation by following their L-L policies instead.

The H-L problem involves multiple robots and objects which are decomposed to single-agent problems. The objects in a L-L problem are a subset of the objects in the H-L problem. These objects are in the initial state of the robot, are involved in the goals allocated to the robot, or are in the L-L plan. Following the example in Figure 4, there are six objects for the L-L problem of *husky2*: *husky2*, *wpg1*, *wpg35*, *wpg52*, *pressure\_wpg52*, and *valve\_wpg35*. In contrast, the H-L problem has 68 objects of which 56 are locations. The L-L action *goto\_waypoint* alone would have 3,080 grounded instantiations considering all of the 56 locations whereas our implementation has only six grounded instantiations. By using a subset of objects, the state-action space is reduced which decreases the sample and computational complexities of Algorithm 2. The initial state for the L-L problem takes from the H-L problem the locations of the base and the robot. Information on whether the robots are localised and the conditions of the Huskies’ manipulator arms and cameras are not represented in the H-L problem. This information is provided separately.

## Experiments and Results

We evaluate our work with a mission: two Huskies (*husky1*, *husky2*) and a UAV (*uav1*) are tasked with checking the temperature at a *poi*, checking the pressure at another *poi*, taking images of two other *poi*, and manipulating two valves. These 6 H-L goals are mapped to 8 L-L goals. The manipulation of a valve is a joint goal which requires the coordination of a husky and a UAV to achieve. This is mapped to 2 L-L goals, one for the husky and one for the UAV. We conducted simulated experiments using RDDLSim (Sanner 2010) as the simulator. We added normally-distributed noise to the duration of actions (20% of its mean duration) which is not made known to our algorithms.

In the first round of planning, MRGA allocates the goals to the robots. Each husky is tasked with manipulating a valve which requires the supervision of *uav1* (i.e., *uav1* has two temporally-dependent joint goals since it cannot perform both at the same time). In addition, *uav1* needs to take images of two *poi* and *husky2* needs to check the temperature and pressure of two *poi*. The state-action spaces of the L-L problems for *uav1*, *husky1*, and *husky2* are  $2^{36} \times 45$ ,  $2^{12} \times 12$ , and  $2^{29} \times 41$ , respectively. If we did not eliminate objects which are of no relevance to a robot when mapping from H-L problem to L-L problems, the state-action spaces of the Huskies would increase to  $2^{37} \times 56$  (both Huskies share the same state-action spaces).

We evaluate the utility of learning over time using MBRL. A L-L policy is learned by simulation training using an approximate model (see Algorithm 2). The parameters for the simulation training are  $H_{sim} = 30$  and  $N = 1000$ . We compare two possible scenarios: **Scenario 1**, where no observations are available and a deterministic model is used, and **Scenario 2**, where observations are acquired from previous missions and an approximate probabilistic model is learned. We assume that the preconditions of L-L actions are known

Scenario	Robot	TC <sup>1</sup>	MC <sub>Plan</sub> <sup>1</sup>	MC <sub>Policy</sub> <sup>1</sup>	%TC <sup>2</sup>	%MC <sub>Plan</sub> <sup>2</sup>	%MC <sub>Policy</sub> <sup>2</sup>
1	uav1	245/400	13/100	14/100	64	42	3
2	uav1	255/400	25/100	12/100	81	46	19
1	husky1	55/100	44/100	11/100	89	23	62
2	husky1	66/100	44/100	22/100	88	23	55
1	husky2	234/300	27/100	27/100	93	36	56
2	husky2	252/300	34/100	28/100	97	43	52

Table 1: Comparison in performance using a deterministic model (Scenario 1) and a learned model (Scenario 2) to train L-L policies for 100 independent simulations of a mission. The performance is evaluated by the following metrics:  $TC$  (number of goals completed / total number of goals),  $MC_{Plan}$  (number of missions completed by following the L-L plan / total number of missions), and  $MC_{Policy}$  (number of missions completed by following the L-L policy / total number of missions). The superscript 1 (2) denotes the results obtained in the first (second) round of planning. In the second round, the results are represented in percentages (e.g., a value of 64 for % $TC$  means 64% of the allocated goals are completed).

for both scenarios and that they only differ in their transition functions. As the deterministic model does not consider any probabilistic outcomes, it cannot predict states where the robot loses localisation, its camera loses calibration, or its manipulator arm is damaged. Therefore, these states are never encountered in the simulation training and the L-L policy is suboptimal or even unsound in these regions of the state space. On the other hand, the learned model predicts all probabilistic outcomes, albeit with the wrong probabilities<sup>1</sup>.

Since the environment is probabilistic, we simulated 100 independent runs for each scenario. The performances of both scenarios are shown in Table 1 and are measured by the number of goals completed ( $TC$ ) and the number of missions completed by following the L-L plan ( $MC_{Plan}$ ), or by following the L-L policy ( $MC_{Policy}$ ). A robot’s mission is completed if all of its goals are completed. If no unexpected situations are encountered during plan execution, a robot can complete all of its goals by executing its L-L plan. Otherwise, it has to adapt by following its L-L policy.

Our empirical results show that the learned model leads to improved performances. In general, the robots completed more missions and goals in Scenario 2 than in Scenario 1. Following the L-L policy trained with the learned model, *uav1* completed 255 out of 400 goals (*uav1* is allocated 4 goals, and there are 100 missions), 10 goals more than the deterministic model. The largest performance improvement due to the learned model is demonstrated in the results for *husky1* where *husky1* completed 22 missions out of 100 missions by following the L-L policy as compared to 11 missions using the deterministic model. Due to the proba-

bilistic nature of the environment, only a small number of the missions are completed when the robots follow the L-L plan ( $MC_{Plan}$ ). The utility of adapting to unexpected situations by switching to the L-L policy can be observed by the relatively significant number of missions completed by following the L-L policy ( $MC_{Policy}$ ).

At the end of its mission, the robots feed back to HLMA. If at least one goal is not completed, replanning at the H-L is required. In this second round of planning, a new temporal plan and problem instance are mapped to a set of L-L problems. The robots then continue their mission to complete the remaining goals. We do not re-use the L-L policies from before as the objects of the planning problems and goals have changed (i.e., the state-action spaces have changed and are different across the simulated runs in the second round). Instead, we train new L-L policies with simulation training using the approximate models again. This underlines the advantage of learning models which can generalise over different planning problems. We assume realistically that observations acquired in a mission (the previous mission) are insufficient to improve the model and hence use the same models as before for both scenarios.

Table 1 shows the results for the remaining missions where at least one goal is not achieved (uncompleted missions). Since both scenarios have different numbers of uncompleted missions, results are represented in percentages for ease of comparison. Scenario 2 generally outperforms Scenario 1 though in some cases, Scenario 1 is better (i.e., the  $MC_{policy}$  for *husky1* and *husky2*). In Scenario 2, the robots completed more goals previously as compared to Scenario 1. The uncompleted goals are typically the joint goals. In other words, Scenario 2 has a larger proportion of goals which are joint goals. These are difficult to complete, and sometimes impossible in the current mission (i.e., it might not be possible for a husky to repair a damaged manipulator arm and then manipulate the valve in time).

In Scenario 1 (Scenario 2), 21 (26) missions were completed where every robot completes all of its allocated goals. Following H-L replanning, in Scenario 1 (Scenario 2), 34 (46) more missions were completed. This leaves 45 (28) uncompleted missions with 74 (44) uncompleted goals in Scenario 1 (Scenario 2). This provides evidence that our work

<sup>1</sup>The model used by RDDLSim (the truth model) has a probability of 0.08 for a camera to be damaged and a probability of 0.05 for a manipulator arm to be damaged. If the camera is damaged, the probabilities of success for the L-L actions *locate\_poi*, *inspect\_poi*, and *capture\_image* are 0.2. If the manipulator arm is damaged, the probability of success for the L-L action *manipulate\_valve* is 0.2. The learned model has a probability of 0.42 for a camera to be damaged and a probability of 0.12 for a manipulator arm to be damaged. If the camera is damaged, the probabilities of success for *locate\_poi*, *inspect\_poi*, and *capture\_image* are 0.77, 0.69, 0.24, respectively. If the manipulator arm is damaged, the probability of success for *manipulate\_valve* is 0.74.

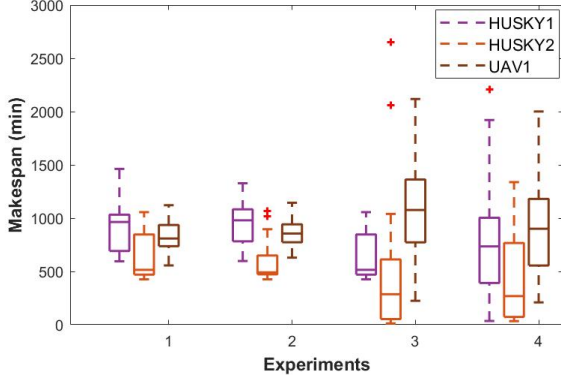


Figure 5: Makespan using a deterministic and a learned model for the first (1 and 2) and second (3 and 4) round of planning respectively for 100 independent simulations.

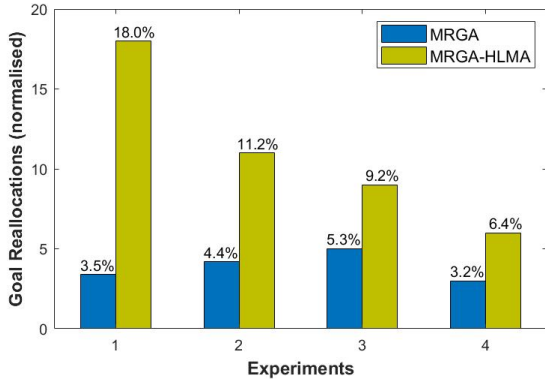


Figure 6: Cumulative reallocation of goals using a deterministic and a learned model for the first (1 and 2) and second (3 and 4) round of planning respectively for 100 independent simulations.

could improve performances over time even with an approximate model by updating the generative model given new observations. Furthermore, the absence of probabilistic reasoning, due to the use of a deterministic model, in Scenario 1 causes the performances to deteriorate.

We also evaluated the interaction of the MBRL approach with temporal planning. Figure 5 shows the statistics for the makespan in the first and second round of planning. In general, the makespans in the second round are longer. This is because the advice from HLMA influences the reallocation of goals such that the risk of failure is reduced during plan execution. Furthermore, the learned model experienced the larger change in makespan as HLMA considers the probabilities of unexpected outcomes. MRGA is thus forced to re-allocate goals less optimally (i.e., trading off makespan for reduced risk). However, this translates to an increase in the number of missions completed for the learned model.

Figure 6 shows the cumulative reallocation of goals in both scenarios and for both rounds of planning. Here, the advice from HLMA influences the number of goals that

MRGA reallocates. HLMA introduces several changes in the duration of actions that affect MRGA’s reasoning about goal distribution. If HLMA is not used, MRGA focuses on reallocation by considering the positions of the robots but does not consider changes in the duration of actions. HLMA also reasons about the trade-off in allocating a goal to a robot that needs to repair its manipulator arm (or calibrate its camera) such that it can achieve a goal with a high probability, versus assigning the goal to an operational robot that may be further away from the goal. Based on this reasoning, HLMA advises MRGA to maintain the capability associated with the goal implementation ( $fk(r, cap)$ ) in the capability set of the robot with the best performance. The number of goals, normalised by the total number of goals, which have to be reallocated is reduced from the first to the second round of planning. Our approach optimises the goal distribution considering learned knowledge from observations. Scenario 2 outperforms Scenario 1 in terms of the number of reallocations required (the fewer the better) in both rounds of planning as a result of using a more accurate model. HLMA allows the approach to reach the best possible plan in comparison to the initial deterministic plan. Therefore, the number of goal reallocations that considers the risk of failure, which is feed back by MBRL, is gradually reduced over subsequent rounds of planning. This explains the similarity of the results (3.2% ~ 6.4%) for the second round of planning in Scenario 2. The goals reallocated after replanning are mostly associated with the positions of the robots when failures occur as HLMA does not provide additional insights about the probability of failures (for the same mission) as they were considered in the previous plan update.

## Conclusions and Future Work

In this paper, we propose a novel hybrid framework, TEMPORALPROB, which decouples multi-goal allocation, multi-agent planning (MAP) under temporal constraints, and model-based single-agent reinforcement learning (MBRL). By considering different levels of model abstractions, we reduce the search space. MAP does not need to reason over all probable outcomes of actions while MBRL only deals with the state-action spaces of a single robot without considering temporal constraints. The reduced computational cost of our framework offers better scalability to large planning problems and is well-suited for online planning and execution. To introduce robustness to uncertainty, feedback from MBRL advises the multi-goal allocation and MAP in their decision-making processes. We demonstrated the applicability of our approach with a fleet of heterogeneous robots in simulated experiments. Results showed that goal-directed performances improved with a model learned from prior observations as compared to an initially assumed deterministic model. For future work, probabilistic reasoning can be used to estimate delays a robot might experience to complete joint goals and provide guidance to a temporal planner to include buffer time. Also, different strategies for goal reallocation can be explored, e.g., when a robot is unlikely to complete a joint goal or a severe unexpected outcome is encountered.

## Acknowledgements

This work was funded and supported by the ORCA Hub ([orcahub.org](http://orcahub.org)), under EPSRC grant EP/R026173/1.

## References

- Abbeel, P.; Quigley, M.; and Ng, A. Y. 2006. Using inaccurate models in reinforcement learning. In *Proceedings of ICML*, 1–8.
- Benton, J.; Coles, A. J.; and Coles, A. 2012. Temporal Planning with Preferences and Time-Dependent Continuous Costs. In *Proceedings of ICAPS*.
- Bernardini, S.; Fox, M.; and Long, D. 2017. Combining temporal planning with probabilistic reasoning for autonomous surveillance missions. *Autonomous Robots* 41(1): 181–203.
- Boutilier, C.; Dearden, R.; and Goldszmidt, M. 2000. Stochastic dynamic programming with factored representations. *Artificial intelligence* 121(1-2): 49–107.
- Boyan, J. A.; and Littman, M. L. 2001. Exact solutions to time-dependent MDPs. In *Advances in Neural Information Processing Systems*, 1026–1032.
- Buckman, J.; Hafner, D.; Tucker, G.; Brevdo, E.; and Lee, H. 2018. Sample-efficient reinforcement learning with stochastic ensemble value expansion. In *Advances in Neural Information Processing Systems*, 8224–8234.
- Carreno, Y.; Pairet, È.; Petillot, Y.; and Petrick, R. P. A. 2020. Task Allocation Strategy for Heterogeneous Robot Teams in Offshore Missions. In *Proceedings of AAMAS*.
- Carreno, Y.; Petrick, R. P. A.; and Petillot, Y. 2019. Multi-agent Strategy for Marine Applications via Temporal Planning. In *IEEE-AIKE*, 243–250. IEEE.
- Coles, A. J.; Coles, A.; Fox, M.; and Long, D. 2010. Forward-Chaining Partial-Order Planning. In *Proceedings of ICAPS*, 42–49.
- Deisenroth, M.; and Rasmussen, C. E. 2011. PILCO: A model-based and data-efficient approach to policy search. In *Proceedings of the 28th International Conference on Machine Learning*, 465–472.
- Eyerich, P.; Mattmüller, R.; and Röger, G. 2009. Using the context-enhanced additive heuristic for temporal and numeric planning. In *ICAPS*.
- Finn, C.; Abbeel, P.; and Levine, S. 2017. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the ICML*.
- Fox, M.; and Long, D. 2003. PDDL2.1: An extension to PDDL for expressing temporal planning domains. *JAIR* 20: 61–124.
- Geramifard, A.; Doshi, F.; Redding, J.; Roy, N.; and How, J. 2011. Online Discovery of Feature Dependencies. In *Proceedings of the ICML*, 881–888.
- Gerevini, A.; and Long, D. 2006. Preferences and soft constraints in PDDL3. In *Proceedings of ICAPS Workshop on Planning with Preferences and Soft Constraints*, 46–53.
- Hasselt, H. v.; Guez, A.; and Silver, D. 2016. Deep Reinforcement Learning with Double Q-Learning. In *Proc. AAAI*, 2094–2100.
- Hsu, C.-W.; and Wah, B. W. 2008. The SGPlan planning system in IPC-6. In *Proceedings of ICAPS*.
- Janner, M.; Fu, J.; Zhang, M.; and Levine, S. 2019. When to trust your model: Model-based policy optimization. In *Advances in Neural Information Processing Systems*, 12519–12530.
- Little, I.; Thiebaux, S.; et al. 2007. Probabilistic planning vs. replanning. In *ICAPS Workshop on IPC: Past, Present and Future*.
- Martínez, D.; Ribeiro, T.; Inoue, K.; Alenyà Ribas, G.; and Torras, C. 2015. Learning probabilistic action models from interpretation transitions. In *Proc. ICLP*, 1–14.
- McDermott, D.; Ghallab, M.; Howe, A.; Knoblock, C.; Ram, A.; Veloso, M.; Weld, D.; and Wilkins, D. 1998. PDDL – The Planning Domain Definition Language (Version 1.2). Technical Report CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control.
- Sanner, S. 2010. Relational dynamic influence diagram language (rddl): Language description. *Unpublished ms. Australian National University* 32: 27.
- Schillinger, P.; Bürger, M.; and Dimarogonas, D. V. 2018. Auctioning over Probabilistic Options for Temporal Logic-Based Multi-Robot Cooperation Under Uncertainty. In *IEEE-ICRA*, 7330–7337.
- Sutton, R. S.; Szepesvári, C.; Geramifard, A.; and Bowling, M. H. 2008. Dyna-style planning with linear function approximation and prioritized sweeping. In *Proceedings of UAI*.
- Tsamardinos, I. 2002. A Probabilistic Approach to Robust Execution of Temporal Plans with Uncertainty. In *SETN*.
- Wan, Y.; Zaheer, M.; White, A.; White, M.; and Sutton, R. S. 2019. Planning with expectation models. In *Proceedings of IJCAI*, 3649–3655.
- Xinyi, Z.; Qun, Z.; Bailing, T.; Boyuan, Z.; and Ming, Y. 2019. Fast task allocation for heterogeneous unmanned aerial vehicles through reinforcement learning. *Aerospace Science and Technology* 92: 588 – 594. ISSN 1270-9638. doi:<https://doi.org/10.1016/j.ast.2019.06.024>. URL <http://www.sciencedirect.com/science/article/pii/S1270963818318704>.
- Younes, H. L.; and Littman, M. L. 2004. PPDDL1. 0: The language for the probabilistic part of IPC-4. In *Proceedings of ICAPS*.
- Yu, T.; Thomas, G.; Yu, L.; Ermon, S.; Zou, J.; Levine, S.; Finn, C.; and Ma, T. 2020. MOPO: Model-based Offline Policy Optimization. *arXiv preprint arXiv:2005.13239*.
- Zhang, K.; Yang, Z.; and Başar, T. 2019. Multi-Agent Reinforcement Learning: A Selective Overview of Theories and Algorithms.
- Zhang, S.; Jiang, Y.; Sharon, G.; and Stone, P. 2017. Multi-robot Symbolic Planning under Temporal Uncertainty. In *Proceedings of AAMAS*, 501–510.



# Construction Site Automation: Open Challenges for Planning and Robotics

**Paolo Forte, Anna Mannucci, Henrik Andreasson, Federico Pecora**

Center for Applied Autonomous Sensor Systems (AASS), Örebro University  
Fakultetsgatan 1, 702 81 Örebro - Sweden  
<name>.<surname>@oru.se

## Abstract

This paper describes open challenges in deploying fleets of autonomous vehicles for material handling in construction sites. First, we review the level of automation of current applications, identifying the technological barriers which are currently limiting automation. Second, we describe the material flow coordination problem in terms of sequences of elementary tasks. Finally, we discuss discrete and continuous-time formulations to solve the integrated task and motion planning, coordination and control problems. We also discuss a potential architecture for closing the sense-plan-act loop in construction site automation.

## 1 Introduction

In the last decades, interest in automating earth-moving processes has grown significantly in view of its potential to improve productivity, reduce labour force, and create a safer work environment. This interest is also driven by the fact that construction operations, such as loading, unloading and moving material, are repetitive, and therefore suitable for automation (Xu and García de Soto 2020; Jayaraj and Divakar 2018). Moreover, robots could potentially perform construction tasks where human presence is undesirable, unsafe, or impossible, e.g., in military applications (Ha, Yen, and Balaguer 2019), in mines, or in space (where teleoperation is not possible due to long time delays).

Today, construction sites are only partially automated (Davila Delgado et al. 2019; Dadhich, Bodin, and Andersson 2016), if at all. Indeed, achieving reliable autonomous navigation in dynamic, harsh, unstructured environments like construction sites is still an open challenge. Weather conditions and the progress of construction activities constantly change the environment. Therefore, offline plans of robot actions often need to be revised at run-time to account for unplanned contingencies. However, online re-planning is difficult due to constraints on maneuverability deriving from the nontrivial kinematic models of most construction machines and from the size, shape and nature of the carried payloads. Operations in the construction site also require interactions between different robots: excavators may be used to transport materials but have limited payload, hence they are less effective than haulers over long distances. Optimizing fleet performance is nontrivial since it depends on both individual

robot capabilities and how shared resources are contended among robots, i.e., interference costs (Nam and Shell 2015).

This paper focuses on automation of material flow in construction sites, that is, the problem of finding a feasible way of displacing material using a fleet of autonomous wheel loaders, excavators and bulldozers. The main contributions of the paper are:

1. In Section 2, we review the current state of automation in construction sites, summarizing the material flow problem and the level of automation currently achieved. In doing so, we outline the importance of 3D graphical tools as a flexible and intuitive way to convey high-level plans.
2. Furthermore, we define a taxonomy of construction-site tasks and how they relate to each other, such that complex tasks can be specified as sequences of elementary tasks.
3. In Section 3, we discuss two solutions for material flow planning: the former combines elements of the Sokoban game (Junghanns and Schaeffer 1997) and of the blocks-world domain (Gupta and Nau 1992); the latter poses the problem as an optimization problem with both continuous and discrete variables.

Section 4 discusses a possible architecture, and relevant work is summarized in Section 5.

## 2 Motivating application scenario

Earth-moving operations are construction processes that consist of moving soil from one position to another via activities such as excavating, loading, hauling, and dumping material. As detailed in the following sections, there are three key process elements: earth-moving robots (agents executing actions), environment model (where actions are performed) and construction tasks (what, when and how).

### 2.1 Earth-Moving Robots

Construction operations usually require the cooperation of different robots. These include excavators (Fig. 1 (left)), commonly used to load material onto haulers (Fig. 1 (center)). Correctly operating these machines manually requires more than 10 years of experience (Hishimoto et al. 2020) due to their non-trivial kinodynamics. Moreover, these machines are characterized by significant purchasing/leasing prices, high operating and maintenance costs. Safe, autonomous and accurate motion planning and control, as well



Fig. 1: Earth-moving robots: an excavator (left), a hauler (center), and a bulldozer (right).

as efficient fleet composition and task allocation may reduce operating costs and increase safety.

At a high level of abstraction, the status of these machines can be described as follows (Peurifoy et al. 2018). During the operational process, an excavator can be: busy (either performing a task or traveling) or idle. An excavator can perform tasks on its own (e.g., soil preparation and excavation) or interact with haulers (i.e., soil-loading), and it can move to new working areas or to parking stations. Once the material is loaded, haulers move the material to a desired position. Similarly to excavators, haulers are characterized by two states: busy or idle, and in the busy state a hauler can be stationary (if it needs to wait for an excavator that loads the material) or mobile. The operations of haulers are composed of specific actions such as filling, travelling, and dumping.

## 2.2 Environmental Model

Online reasoning, activity identification and monitoring are essential steps to control the performance of earth-moving operations (e.g., an excavator should have finished loading the material before the hauler starts moving). In addition, planned earth-moving operations and external factors (such as rain) may affect the traversability and therefore the ability to execute operations. However, maintaining an effective representation of the construction site is challenging. Traversability is indeed not trivial to model, as environmental conditions may have different effects on different robots, e.g., mud may slow down an excavator but render the area inaccessible by a hauler.

A related problem is that of obtaining knowledge of the current situation from observations. Many vision-based methods are in use in on-site construction management applications for, e.g., safety assessment (Chi and Caldas 2012), progress monitoring (Ahmed, Haas, and Haas 2012), and productivity analysis (Kim et al. 2018). These methods use image processing and supervised learning to identify workers, equipment, materials, and structures (Golparvar-Fard, Heydarian, and Niebles 2013). Hence, they rely on the availability of a sufficient amount of labelled training data to recognize construction resources. Data of the construction site can be collected using unmanned aerial vehicles (UAV), and used to create a 3D model of the site (Bang et al. 2020; Halbach and Halme 2013). The model would ideally be used in a 3D graphical tool which allows users to drag and drop materials in the environment, thereby specifying goals for the fleet, as shown in Fig. 2.

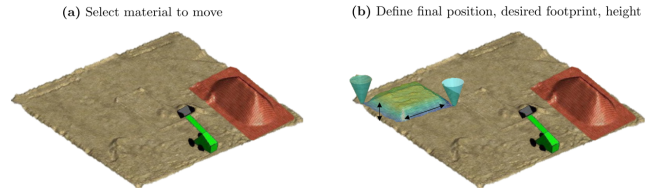


Fig. 2: Example of a goal specification via drag and drop operations, source: (Halbach and Halme 2013).

## 2.3 Construction Site Tasks

A goal of construction site automation is to synthesize construction tasks (e.g., reach a pile, load the material and dump it at dump locations, etc.) from high-level specifications (e.g., drag-and-drop amounts of material into desired locations). This subsumes the problems of synthesizing feasible motions and controls for the robots (that is, solving the motion planning, coordination and control problems), and maintaining feasibility in the face of contingencies. To the best of our knowledge, a classification of tasks in construction sites that is adequate for automated planning has never been specified. We therefore propose the following (see also Fig. 3):

**Navigating:** all required operations to move between locations. This task may be performed by all the robots shown in Fig. 1 with different costs/efficiency in terms of carried payload per trip. Reliable and safe autonomous navigation requires accurate localization and motion control, multi-machine coordination, online monitoring, obstacle detection and avoidance.

**Material handling:** all operations to load, unload or flatten the material. This task may be performed by excavators or bulldozers (Fig. 1).

**Spreading/compacting:** when navigating and handling the material, these tasks are performed concurrently.

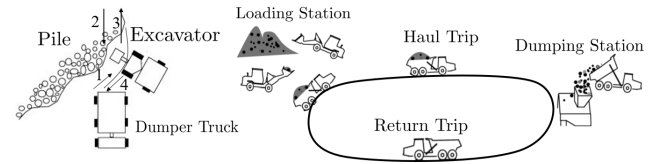


Fig. 3: Construction site tasks: loading (left), navigating and material handling (right), source: (Halbach and Halme 2013).

The main challenge in plan synthesis is to solve the task assignment and sequencing, motion planning and control problems jointly. Typically, in order to diminish computational overhead, these problems are solved in a decoupled manner, often neglecting constraints imposed by other sub-problems. However, the higher the level of abstraction at the task level (e.g., assuming holonomic kinodynamics), the higher the probability that the resulting plan will be unfeasible or sub-optimal. How to effectively reason about kinodynamic and environmental constraints is indeed an open problem (Erdem, Patoglu, and Schüller 2016).

### 3 Task Planning: Problem Formulation

Some task features can be described accurately in discretized space (e.g., the type and sequence of actions to be performed), while others require a continuous-space representation (e.g., the spatial distribution and quantity of granular materials). To understand the complexity vs. performance trade-off while discretizing some of these quantities, we here investigate two formulations of the task planning problem: a discrete one, based on a combination of the Sokoban and Blocks-world domains, and a continuous formulation, where the problem is posed as an Optimal Assignment Problem (OAP).

#### 3.1 Discrete Formulation

In this representation we discretize the following: 1) the quantity of material into fixed volumes (blocks); 2) the environment into sectors – each corresponding to a load or unload location and having a fixed block capacity– where each sector may be either free (no material) or occupied according to the quantity of material stored in it; 3) the set of traversable paths using a roadmap, whose nodes corresponds to fixed locations (one per sector), and whose edges represent the existence of kinematically-feasible paths between nodes. Similarly to Sokoban, blocks can be pushed; similarly to the blocks-world, they can be picked, stacked and unstacked. The volume of material that can be moved by each robot in one trip depends on the robot and the material types. Some earth-moving processes, e.g., building a road, may require different layers of material to be displaced in a predefined order (as shown in the example in Fig. 4). We

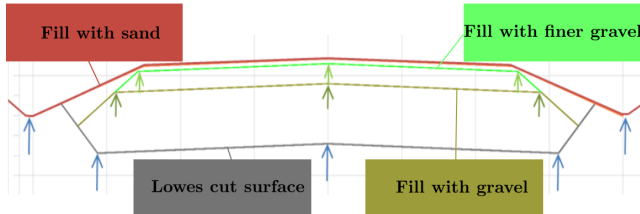


Fig. 4: Layers description to build a road.

include elements of the blocks-world domain to account for these ordering constraints among tasks. The full PDDL formulation of the domain is available online in two variants<sup>1</sup> (with and without durative actions) and has been tested with both total and partial-order planners<sup>2</sup>. Note that in this formulation: 1) fixed sectors and fixed positions of piles inside sectors are assumed; 2) plans are computed and executed synchronously among all robots (leading to useless waiting times); 3) multi-robot interference must be accounted for separately, e.g., by considering intersections among trajectories (hence with polynomial cost in the number of edges of the roadmap); 4) specifying the problem is not straightforward as it requires good knowledge of the PDDL domain,

<sup>1</sup>See <http://bit.ly/38gc4HA> for details.

<sup>2</sup>A selection of the tested planners is reported at <https://planning.wiki/ref/planners>. In future work we will investigate the use of the multi-agent version of PDDL (MA-PDDL).

rendering automatic/easy specification of the problem (as would be required for real deployment) problematic.

#### 3.2 Continuous Formulation

Continuous optimization may be exploited to formulate the task planning problem while avoiding the discretization into sectors and allowing arbitrary locations of piles. In future work we will investigate a Single Task, Multi Robot, Time-extended Assignment (ST-MR-TA) Multi-Robot Task Assignment (MRTA) (Gerkey and Mataric 2004) formulation, where tasks (which can be cooperative) are *asynchronously* assigned to idle robots while considering task scheduling. To account for multi-robot interference, we will combine interference-free costs (e.g., the capabilities of different robots) and interference costs (e.g., overlapping trajectories) in the objective function of the OAP as done in (Forte et al. 2021). We expect this to yield an intractable Mixed Integer Linear Programming (MILP) problem (Nam and Shell 2015). We specifically aim to investigate the interactive variables to define/tune the complexity and scalability of this continuous formulation, e.g., the maximum number of alternative paths for each pair of nodes in the roadmap.

### 4 Towards an Architecture

As we have discussed, the earth-moving problem subsumes solving task planning, as well as other important sub-problems, such as:

**Tasks allocation and sequencing**, e.g., integrating one of the solutions proposed in Section 3. This module outputs an allocation of robots to sequences of tasks which are subject to temporal and resource constraints.

**Motion planning**, which either synchronously (Discrete Formulation) or asynchronously (Continuous Formulation) generates the set of kinematically feasible paths.

**Coordination**, which is responsible of defining and refining over time the set of constraints on the robot trajectories to: 1) avoid inter-robot collisions (safety); 2) ensure all robots will accomplish their tasks (liveness). For this purpose, we will leverage our previous work (Pecora, Cirillo, and Dimitrov 2012) on heuristically-driven precedence constraints over shared regions of the robots’ paths.

**Perception**, which is responsible for 1) constructing collections of spatial constraints for the robot motions and obstacle detection, and 2) monitoring the progress/outcome of construction tasks. This may close the system control loop to handle contingencies and failures.

**Localization**, which provides the positions of robots in the site (e.g., by combining data from GPS, on-board range devices and IMUs (Saarinen et al. 2013)).

**Control**, which computes robot commands satisfying all constraints posed by the previous modules.

Task planning will hence be integrated into an architecture including each of these processes (see Fig. 5).

We believe that a shared representation and online reasoning are keys to effectively close the overall sense-plan-act loop in dynamic and unpredictable environments as construction sites.



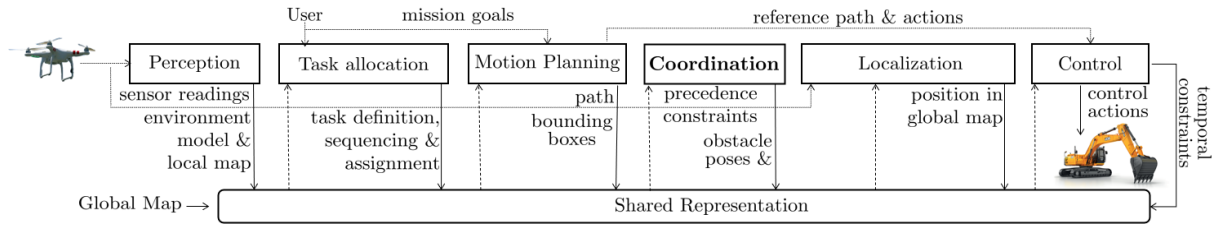


Fig. 5: A possible modular architecture.

## 5 Related Work

Interesting work in the literature which shares some similarities with the earth-moving problem is reported below.

### Integrated task and motion planning architectures.

Several results in hybrid planning (i.e., integrating high-level planning with low-level robot control) have been proposed over the years to solve complex hybrid planning problems, with different outcomes according to the level of integration (Erdem, Patoglu, and Schüller 2016). Both Cambon, Alami, and Gravot (2009) (multi-robot in a fixed and known environment) and Lozano-Perez and Kaelbling (2014) (single robot) combine symbolic search for sequences of (discrete) tasks and reasoning about (continuous) geometric constraint to defer the specification of high-level plans (in a discrete space of actions) to low-level planners (which account for geometric constraints and kinematic feasibility). Extensions toward dynamics and uncertainties in the environment, online re-planning and multi-robot cooperation are required to apply these techniques in construction sites. Conversely, only a subset of earth-moving problems are addressed by Multi-Agent Task Assignment and Path Finding methods (Ma et al. 2017), in which dealing with asynchronous assignment, with uncertainties, online planning and replanning under kinodynamic constraints are current open issues.

**Motion planning in dynamic environments.** Mansouri, Lagriffoul, and Pecora (2017) proposes a variant of the multi-vehicle routing problem with homogeneous fleets which accounts for non-holonomic constraints and dense, dynamic obstacles related to a real-world mining application. Similarly to the problem described in this paper, the authors highlight advantages and limitations of formalizing their problem as a variant of the Sokoban game. However, their tasks (drilling blast holes, which create obstacles and hence change the traversability of the environment) do not require interaction with the material, nor the cooperation of different machines.

**Construction site inspection and monitoring.** Unmanned aerial systems (UASs) and Unmanned ground vehicles (UGVs) have recently been utilized for safety inspection (Martinez et al. 2021), to detect unsafe conditions and protect workers from potential injuries and fatal accidents (Park, Kim, and Cho 2017), and to detect cracks on buildings, e.g., bridges (Morgenthal et al. 2019; Lim, La, and Sheng 2014), or houses after natural disasters (Torok, Golparvar-Fard, and Kochersberger 2014). Tools for productivity analysis of earth moving processes based

on image data have shown promise in terms of economic yield, higher efficiency and reduction of costs (Kim et al. 2018; Kim, Chi, and Seo 2018). Zhu, Ren, and Chen (2016) proposes a particle filtering approach which mitigates occlusion problems in tracking construction workers and equipment; Park and Brilakis (2016) presents an integrated detection and tracking method to localize construction objects in consecutive video frames. Kim, Kim, and Kim (2016) presents a tracking method for moving objects in construction sites using the Gaussian mixture model and morphological processing, and a construction equipment tracking method based on real-time learning from an automatically developed training database (Kim and Chi 2017).

## 6 Conclusion

On-site autonomous construction robots promise to improve productivity and working conditions for humans. Yet automating the earth-moving process poses difficulties in both planning and robotics, due to the presence of dynamic obstacles, heterogeneous robots, and the complexity of describing and tracking the displacement of material in the construction site. In this paper, we have outlined some of the resulting open challenges, with the aim to elicit a suitable problem formulation for practical applications. As a next step, we will extend, simulate and compare discrete and continuous formulations of the problem, and integrate them within a modular architecture with a shared representation for perception, planning, coordination and control.

## References

- Ahmed, M.; Haas, C.; and Haas, R. 2012. Using digital photogrammetry for pipe-works progress tracking. *Canadian J. Civil Eng.* 39(9): 1062–1071.
- Bang, S.; Baek, F.; Park, S.; Kim, W.; and Kim, H. 2020. Image augmentation to improve construction resource detection using generative adversarial networks, cut-and-paste, and image transformation techniques. *Automat. Construct.* 115: 103198. ISSN 0926-5805.
- Cambon, S.; Alami, R.; and Gravot, F. 2009. A Hybrid Approach to Intricate Motion, Manipulation and Task Planning. *The Intern. J. Robot. Research* 28: 104 – 126.
- Chi, S.; and Caldas, C. H. 2012. Image-Based Safety Assessment: Automated Spatial Safety Risk Identification of Earthmoving and Surface Mining Activities. *J. Construction Eng. Manage.* 138(3): 341–351.

- Dadhich, S.; Bodin, U.; and Andersson, U. 2016. Key challenges in automation of earth-moving machines. *Automat. in Construction* 68.
- Davila Delgado, J. M.; Oyedele, L.; Ajayi, A.; Akanbi, L.; Akinade, O.; Bilal, M.; and Owolabi, H. 2019. Robotics and automated systems in construction: Understanding industry-specific challenges for adoption. *J. Build. Eng.* 26: 100868.
- Erdem, E.; Patoglu, V.; and Schüller, P. 2016. A systematic analysis of levels of integration between high-level task planning and low-level feasibility checks. *AI Commun.* 29: 319–349.
- Forte, P.; Mannucci, A.; Andreasson, H.; and Pecora, F. 2021. Online Task Assignment and Coordination in Multi-Robot Fleets. *IEEE Trans. Robot. Autom.* 6: 4584–4591.
- Gerkey, B. P.; and Mataric, M. J. 2004. A formal analysis and taxonomy of task allocation in multi-robot systems. *The Intern. J. Robot. Research* 23(9): 939–954.
- Golparvar-Fard, M.; Heydarian, A.; and Niebles, J. C. 2013. Vision-based action recognition of earthmoving equipment using spatio-temporal features and support vector machine classifiers. *Adv. Eng. Inform.* 27(4): 652–663. ISSN 1474-0346.
- Gupta, N.; and Nau, D. 1992. On the Complexity of Blocks-World Planning. *Artif. Intell.* 56: 223–254.
- Ha, Q.; Yen, L.; and Balaguer, C. 2019. Robotic autonomous systems for earthmoving in military applications. *Automat. Construct.* 107: 102934. ISSN 0926-5805.
- Halbach, E.; and Halme, A. 2013. Job planning and supervisory control for automated earthmoving using 3D graphical tools. *Automat. Construct.* 32: 145–160. ISSN 0926-5805.
- Hishimoto, T.; Yamada, M.; Yamauchi, G.; Nitta, Y.; and Yuta, S. 2020. Proposal for Automation System Diagram and Automation Levels for Earthmoving Machinery. In *Proc. 37th Int. Symp. Automat. Robot. CConstruct. (ISARC)*, 347–352. Kitakyushu, Japan: Int. Ass. Automat. & Robot. Construct. (IAARC).
- Jayaraj, A.; and Divakar, H. 2018. Robotics in Construction Industry. *IOP Conf. Materials Sci. & Eng.* 376: 012114.
- Junghanns, A.; and Schaeffer, J. 1997. Sokoban: A Challenging Single-Agent Search Problem. In *IJCAI 1997*.
- Kim, H.; Bang, S.; Jeong, H.; Ham, Y.; and Kim, H. 2018. Analyzing context and productivity of tunnel earthmoving processes using imaging and simulation. *Automat. Construct.* 92: 188–198. ISSN 0926-5805.
- Kim, H.; Kim, K.; and Kim, H. 2016. Vision-Based Object-Centric Safety Assessment Using Fuzzy Inference: Monitoring Struck-By Accidents with Moving Objects. *J. Comput. Civ. Eng.* 30(4): 04015075.
- Kim, J.; and Chi, S. 2017. Adaptive Detector and Tracker on Construction Sites Using Functional Integration and Online Learning. *J. Comput. Civ. Eng.* 31(5): 04017026.
- Kim, J.; Chi, S.; and Seo, J. 2018. Interaction analysis for vision-based activity identification of earthmoving excavators and dump trucks. *Automat. in Construction* 87: 297–308. ISSN 0926-5805.
- Lim, R.; La, H.; and Sheng, W. 2014. A Robotic Crack Inspection and Mapping System for Bridge Deck Maintenance. *Trans. Automat. Sci. Eng.* 11: 367–378.
- Lozano-Perez, T.; and Kaelbling, L. 2014. A constraint-based method for solving sequential manipulation planning problems. *2014 IEEE/RSJ Int. Conf. Intell. Robots Syst.* 3684–3691.
- Ma, H.; Koenig, S.; Ayanian, N.; Cohen, L.; Hönig, W.; Kumar, T. K. S.; Uras, T.; Xu, H.; Tovey, C.; and Sharon, G. 2017. Overview: Generalizations of Multi-Agent Path Finding to Real-World Scenarios. *ArXiv abs/1702.05515*.
- Mansouri, M.; Lagriffoul, F.; and Pecora, F. 2017. Multi vehicle routing with nonholonomic constraints and dense dynamic obstacles. In *2017 IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, 3522–3529.
- Martinez, J. G.; Albeaino, G.; Gheisari, M.; Issa, R. R.; and Alarcón, L. F. 2021. iSafeUAS: An unmanned aerial system for construction safety inspection. *Automat. in Construction* 125: 103595. ISSN 0926-5805.
- Morgenthal, G.; Hallermann, N.; Kersten, J.; Taraben, J.; Debus, P.; Helmrich, M.; and Rodehorst, V. 2019. Framework for automated UAS-based structural condition assessment of bridges. *Automat. in Construction* 97: 77–95. ISSN 0926-5805.
- Nam, C.; and Shell, D. A. 2015. Assignment algorithms for modeling resource contention in multirobot task allocation. *IEEE Trans. Automat. Sci. Eng.* 12(3): 889–900.
- Park, J.; Kim, K.; and Cho, Y. K. 2017. Framework of Automated Construction-Safety Monitoring Using Cloud-Enabled BIM and BLE Mobile Tracking Sensors. *J. Construction Eng. Manage.* 143(2): 05016019.
- Park, M.-W.; and Brilakis, I. 2016. Continuous localization of construction workers via integration of detection and tracking. *Automat. Construct.* 72: 129–142. ISSN 0926-5805.
- Pecora, F.; Cirillo, M.; and Dimitrov, D. 2012. On Mission-Dependent Coordination of Multiple Vehicles under Spatial and Temporal Constraints. *IEEE Int. Conf. Intell. Robots Syst.* .
- Peurifoy, R.; Schexnayder, C.; Schmitt, R.; and Shapira, A. 2018. *Construc. Plann. Equip. Meth. (9th Ed.)*.
- Saarienen, J.; Andreasson, H.; Stoyanov, T.; and Lilienthal, A. J. 2013. Normal distributions transform Monte-Carlo localization (NDT-MCL). In *2013 IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 382–389. IEEE.
- Torok, M. M.; Golparvar-Fard, M.; and Kochersberger, K. B. 2014. Image-Based Automated 3D Crack Detection for Post-disaster Building Assessment. *J. Computing in Civil Eng.* 28(5): A4014004.
- Xu, X.; and García de Soto, B. 2020. On-site Autonomous Construction Robots: A review of Research Areas, Technologies, and Suggestions for Advancement.
- Zhu, Z.; Ren, X.; and Chen, Z. 2016. Visual Tracking of Construction Jobsite Workforce and Equipment with Particle Filtering. *J. Computing Civ. Eng.* 30: 04016023.

# Probabilistic Plan Legibility with Off-the-shelf Planners

Anonymous Authors

## Abstract

Legible planning is the creation of plans that best disambiguate their goals from a set of other candidates from an observer’s perspective. In this paper we propose a method for legible planning for arbitrary PDDL domains, by extending previous research on legibility to classical planning without requiring to construct ad-hoc planners. We also discuss how the observer perspective may be estimated through a second order theory of mind that connects the planner’s and the observer’s task spaces. Our solution can for example be deployed in human-robot teaming scenarios, where an autonomous robot in a team can implicitly communicate its goal by producing legible plans. We present benchmark results on several PDDL planning domains. Our results generally show that plan legibility is a trade-off with plan efficiency, however, not all planning domains allows to increase legibility in the same way and a regularizing factor to balance legibility and efficiency was proved necessary.

## Introduction

A main effort in recent research in Artificial Intelligence is to provide intelligent decision-making algorithms the ability to produce explanations, and to make their decisions understandable. Three aspects have been identified as crucial to allow understandability: their ability to produce trust, to allow interaction with their decisions, and to be transparent in their decision making processes (Fox, Long, and Magazzeni 2017). Even though such guidelines for *Explainable AI* are mostly tailored for recommender systems, they can be applied to most AI systems, and especially for algorithms producing plans of actions, for example to execute on an embodied agent such as a robot. In fact, a planner (robot) collaborating with humans should indeed propose useful plans, allow to mediate such plans, and be transparent in its actions. The concept *understandable robots* (Hellström and Bensch 2018) is often used in this context.

In this paper we focus on the *transparency* part of Explainable AI for planning algorithms, which broadly translates into the field of interpretable planning, i.e. the generation of plans that are understandable from a human point of view (Chakraborti et al. 2017; 2019; MacNally et al. 2018; Kulkarni, Srivastava, and Kambhampati 2019). Planning while keeping a human observer perspective into consideration is desired in most human-robot interaction, where the

collaboration with a robot may become difficult whenever its observed actions and behaviors are not directly understandable by the human collaborator (Chakraborti et al. 2019).

A robot’s behavior can be hard to understand for reasons such as model discrepancies (Chakraborti et al. 2017), where the robot uses a task model different than the human’s, or an asymmetry of information between human and robot, e.g. differences in their beliefs. If not dealt with, such divergencies may lead to decreased interpretability of the robot’s behavior, ultimately leading to a loss of trust in the robotic system (Hellström and Bensch 2018).

Interpretability in planning has been addressed using several terms with subtle yet relevant differences, such as *explainable* (Kulkarni et al. 2019; Zhang et al. 2017), *predictable* (Zhang et al. 2017) and *legible* planning (Dragan, Lee, and Srinivasa 2013; MacNally et al. 2018). In this paper we use legibility, which focuses on reducing ambiguity over other possible goals (Chakraborti et al. 2019). Legible planning requires definition of optimality criteria for plans that are not exclusively based on their execution cost, but rather on how the plan’s goal can be successfully identified. In general, this requires to explicitly model the observer as an agent performing inference over the planner’s actions and weighting the observations towards a set of possible goals.

The contributions in this paper extend the formalism for plan legibility (especially (Dragan, Lee, and Srinivasa 2013; Dragan and Srinivasa 2014)) to classical planning methods such as PDDL (McDermott 1998) without requiring the construction of ad-hoc planners. We further provide discussion on how legible planning involves the creation of plans that are legible inside the model used by the observer to evaluate goals, hence requiring to take into account the differences between this model and the task model being internally utilized by the planner. This difference between models has been addressed in adjacent topics such as model reconciliation (Chakraborti et al. 2017), but has, to the best of our knowledge, never been thoroughly put side-to-side with interpretable planning methods where planner and observer either share the same task model (e.g. trajectories in the Cartesian space) or have a different observation space. We argue that legible planning requires a second order theory of mind, which allows the planner to evaluate its own plans from a perspective of an observer that uses a possibly inaccurate mental model of the planner. Theory of mind acts as

the glue that connects their two task spaces.

We further provide an algorithm to produce legible plans using off-the-shelf PDDL planners, and measure its performance over several standard domains. Since PDDL is easy to use and fairly well known by the planning and robotics communities, an algorithm for legible planning based on PDDL can easily find practical implementations for robotic scenarios, in fields such as human-robot interaction and human-robot teaming, or for algorithms exploring theory of mind in the domain of planning.

The rest of the paper is structured as follows. In Section 2 we provide some brief relevant background on legible planning. In Section 3 we introduce a working definition of plan legibility in a probabilistic setting. Theory of mind is contextualized as a function which transforms the planner domain to the observer’s perspective. In Section 4 we propose a procedure to compute legible plans using off-the-shelf PDDL planners. In Section 5 and 6 we provide an illustrative example and performance measures over a set of planning domains. In Section 7 we provide some conclusive remarks.

## Background

Plan legibility for robotic manipulators was investigated in (Dragan, Lee, and Srinivasa 2013; Dragan and Srinivasa 2013; 2014). The authors proposed trajectories that are skewed in a way that avoids ambiguity of goal locations. The increased legibility of these trajectories was opposed to optimality in terms of travelled distance. The observer and the robot were respectively evaluating and producing legible plans in the Cartesian space of the manipulator. The same authors investigated how a manipulator’s legibility affects human-robot interaction in a task where experimenters are asked to act according to the robot’s predicted goal (Dragan et al. 2015). Their results showed that legible trajectories positively affected the perceived interpretability of the robot, increased interaction fluency, and overall induced a greater sense of collaboration during the interaction. They further showed, through a user study, that there was a significant preference for legible plans rather than functional trajectories (avoiding obstacles), or predictable ones (with minimal cost).

In (MacNally et al. 2018), the authors use POMDPs to model how the beliefs of an observer, in particular inferred goals, are affected by different action selection policies. Since sequences of actions performed by an actor (e.g. a robot) leads to inferred goal distributions in an observer’s beliefs, the authors discuss how actions can be used to communicate what the goal being pursued is, hence allowing an *implicit* communication of the actor’s goal that was highlighted as relevant whenever no explicit communication mechanisms between actor and observer exist. This implicit communication of the goal by acting in a way recognizable by observers was named transparent planning, which is equivalent to the notion of legible planning used by other authors. Legibility as an implicit communication mechanism is also explored in (Miura and Zilberstein 2020). Through the utilization of *Legible Markov Decision Processes* (MDPs) the authors show how it is possible to in-

tegrate an observer’s beliefs into the actor’s beliefs. An important point discussed by the authors is that legibility can be arbitrarily increased through a trade-off with planning cost, and a balance between the two is almost always required. In both (MacNally et al. 2018) and (Miura and Zilberstein 2020) there is no disambiguation between the planner’s and the observer’s task space, but rather they use the same underlying MDP.

Plan legibility is similarly discussed as a planning problem with controlled observability. By controlling how observations are made, an observer can be made to recognize the correct goal. Adversarial and cooperative observers could be respectively misled or informed. For example, (Kulkarni, Srivastava, and Kambhampati 2019) propose a controlled observability planning algorithm that can create plans that are more legible or obfuscating depending on the employed heuristic to search the state space. The authors propose for every goal to have a heuristic attracting the search, and by averaging these heuristics with appropriate weights they obtain plans that are more legible. The same authors extend this line of research by allowing for both cooperative and adversarial observers to be simultaneously present (Kulkarni, Srivastava, and Kambhampati 2020). In both papers, planner and observer have different task spaces where the ground truth observations, found by an observation model, are respectively mapped.

## Probabilistic Plan Legibility

We make the assumption that discrimination of goals is performed by an observer while observing the planner performing a task. Hence, the legibility of a plan is based on the observations that the plans lead to and prior beliefs over the task, e.g. the state of the task prior to the attempt of recognizing its goal. Therefore, legible planning should produce observations in the observer perspective that, together with prior beliefs, lead to the best discrimination of the plan’s true goal.

Taking a probabilistic approach, at any moment there is a set  $G = \{\hat{g}, g_0, g_1, \dots, g_n\}$  of possible goals the planner can aspire to. To be legible, a plan must produce observations that make its true goal  $\hat{g}$  easily discernible from the other goals in  $G$ . Therefore, generating legible plans is equivalent to finding the plan that minimizes the difference between the true goal distribution that the planner has, and a goal distribution that the observer infers from the observations. In this setting, the observer can be modeled as a probabilistic model for goal recognition  $P(G|\Pi)P(\Pi)$ , with  $\Pi$  being the set of possible plans,  $P(\Pi)$  the prior probability distribution of observing sequences of actions (plans) while  $P(G|\Pi)$  is the distribution of the goals given observed plans. Hence, the observer is modeled as estimating the planner’s goal distribution by observing its sequences of actions. For a goal  $g \in G$  we define the legibility of a plan  $\pi$  in this plan space as:

$$\text{legibility}(\pi, g) = \mathcal{H}(P_g(G), P(G|\pi)P(\pi)), \quad (1)$$

where  $\mathcal{H}$  is a similarity function of two probability distributions.  $P_g(G)$  is the goal probability distribution for the goal,

with  $P(G = g) = 1$  and  $P(G) = 0$  for all other possible goals.

The most legible plan is denoted  $\pi_{\text{legible}}$  and defined as follows:

$$\pi_{\text{legible}} = \underset{\pi \in \Pi_{\hat{g}}}{\operatorname{argmax}} \operatorname{legibility}(\pi, \hat{g}), \quad (2)$$

where we consider only the plans  $\Pi_{\hat{g}}$  that achieve the true goal  $\hat{g}$  from a given starting condition  $I$  which is the same for every goal in  $G$ . Whether  $\hat{g}$  is effectively best discerned from the other candidate goals is not assured, as it in some cases may be impossible to generate sufficiently legible plans due to constraints in the task space, or it might even be impossible to generate a plan at all.

Since Eq. 1 considers only complete plans, we further introduce the notion of  $n$ -legibility, indicating the legibility of a plan after  $n$  steps:

$$n\text{-legibility}(\pi, g) = \operatorname{legibility}(\pi_{1..n}, g) \quad (3)$$

where  $\pi_{1..n}$  is the plan comprising the  $n$  first steps of  $\pi$ .

We finally propose a generalized measure  $\hat{n}$ -legibility as a weighted average of the legibility of all of the plan prefixes  $\pi_1, \pi_{1..2}, \dots, \pi_{1..n}$ :

$$\begin{aligned} \hat{n}\text{-legibility}(\pi, g) &= \sum_{i \in 1..n} w_i \cdot \operatorname{legibility}(\pi_{1..i}, g) = \\ &\sum_{i \in 1..n} w_i \cdot \mathcal{H}(P_g(G), P(G|\pi_{1..i})P(\pi_{1..i})), \quad (4) \\ &\sum_i w_i = 1. \end{aligned}$$

From this definition, the legibility of any step  $j$  can be obtained by setting  $w_j = 1$ . Other customized averages may, for example, give a greater importance of legibility during the early steps of plans.

## Theory of Mind

Legible planning requires the planner to estimate the probability distribution over the possible goals, computed from the observer's perspective. While in previous discussion we implicitly assumed that both planner and observer share the same observation model, we now remove this assumption by introducing a second-order theory of mind (Meijering et al. 2011; Devin and Alami 2016).

Theory of mind relates to the ability of agents to attribute mental states and beliefs to themselves or other agents, and of creating a point of view of a situation in terms of beliefs, goals and intentions that is different from their own but rather belonging others. A first order theory of mind is expressed in the sentence "Bob thinks that Alice thinks X", or in other words Bob has an estimate of Alice's mental state, believing she's thinking X. Higher order theories deepen these levels of reasoning by extending the thinking chain. A second order reasoning would be "Carl thinks that [Bob thinks that Alice thinks X]"—with parenthesis added to highlight the recursion. In this case Carl holds an estimate of Bob's mental state. Arbitrary higher orders of reasoning follow the same incremental structure.

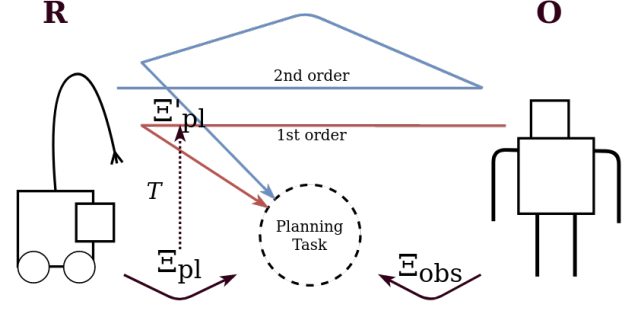


Figure 1: Legible planning requires a second order theory of mind, by which the planner, working e.g. on a mobile robot  $R$ , estimates how the observer  $O$  interprets the robot's actions by using a first order theory of mind of the robot. Legibility of plans is computed inside  $\Xi'_{pl}$  that is  $O$ 's model of  $R$ .

In this paper we propose that legible planning requires a second order theory of mind as a model of how the observer infers the planner's goal. Consider the statement: " $O$  thinks that  $R$ 's goal is  $G$ ", where  $O$  is the observer,  $R$  is the robot,  $G$  is the robot's goal. The statement describes  $O$ 's first-order theory of mind. At this level the belief about  $R$ 's goal belongs to  $O$ , and is not accessible by  $R$ . A second order chain of reasoning can be described as " $R$  thinks that [ $O$  thinks that  $R$ 's goal is  $G$ ]", where at the first level  $O$  uses her first order theory of mind, while at the second level the planner estimates the result of this inference by using its second order theory of mind. For  $R$ 's legible planning, the second order reasoning can be reformulated as " $R$ 's goal is that [ $O$  thinks that  $R$ 's goal is  $G$ ]". The planner's action are legible not in the observer's task model (as was also proposed in previous literature), but rather inside the model that the observer uses to evaluate the planner's actions. This model, when implemented by  $R$ , effectively allows it to access  $O$ 's beliefs about him.

Following this reasoning in the setting of planning domains we define a second order theory of mind as a function  $T$  which allows to transform the plan instances  $\Xi_{pl}$  utilized by the planner to the observer perspective of the same  $\Xi'_{pl}$ :

$$T = T_{pl} \circ T_{obs} : \Xi_{pl} \rightarrow \Xi_{obs} \rightarrow \Xi'_{pl} \quad (5)$$

such that for a specific instance  $\Xi_{pl}$  we can get a corresponding instance as  $T(\Xi_{pl}) = \Xi'_{pl}$ .  $T$  can also be seen as the composition of the first order theories of mind  $T_{pl}$  and  $T_{obs}$ , through which the planner makes an estimate of the observer's first order model, hence obtaining a second order model. This procedure is illustrated in Figure 1.

In PDDL,  $T$  could transform domains from  $\Xi_{pl}$  by modifying its operators and predicates with corresponding operators and predicates taken from  $\Xi'_{pl}$ , and an associated problem instance by describing the initial and goal conditions in terms of truth values from  $\Xi'_{pl}$ . In this setting, plans produced in  $\Xi_{pl}$  induce corresponding plans in  $\Xi'_{pl}$ .

To compute legibility the domains  $\Xi'_{pl}$  are utilized, as they model how the observer perceives and integrate in its

beliefs the produced observations in the form of a plan. Notice that this model is possessed locally by the planner by its second order theory of mind, and once implemented does not require additional external input to execute a task legibly. Nevertheless, even after deployment, if  $\Xi_{\text{pl}}$  and  $\Xi'_{\text{pl}}$  result being sufficiently different from one another the observer can judge the plans as non legible. In these cases it may be necessary for the robot to produce explanations to make the domains compatible again (Chakraborti et al. 2017). Since model reconciliation is outside of the scope of legible planning, we will not cover the cases in which the two planning models require reconciliation, and to avoid reconciliation scenarios we introduce the hypothesis that every plan instance computed in  $\Xi_{\text{pl}}$  has exactly one valid description also in  $\Xi'_{\text{pl}}$ .

### Goal Recognition using PDDL

As previously discussed, we model the observer as a probabilistic goal recognizer  $P(G|\Pi)P(\Pi)$  which infers the probability distribution over the possible goals given sequences of observed actions, possibly also integrating the observations with contextual information from its beliefs.

Though it is possible to implement it in many ways, we realized it by a probabilistic model based on the Planning Domain Description Language (PDDL). The provided formulation for goal recognition is based on (Persiani and Hellström 2020), and is flexible in how observed actions can appear in the plans  $\Pi_{\Xi'}$ , which, as we will later discuss, allows us to easily define a set of theory of mind models as functions which drop parts of the observations. In the following we give some technical implementation details.

PDDL (McDermott 1998) is a standard language to specify planning domains for what is usually referred to as *classical planning*. It is based on the STRIPS syntax and uses predicate logic to describe the current task state. In PDDL, a planning domain is specified by the tuple  $\langle \mathcal{P}, \mathcal{A} \rangle$ , where  $\mathcal{P}$  is the set of possible truth predicates describing a state and  $\mathcal{A}$  a set of operators that allow to transition between states. Every operator is defined by the triple  $\langle \text{pre}(a), \text{eff}^-(a), \text{eff}^+(a) \rangle$ .  $\text{pre}(a)$  is a list of predicates that must be true in a given state for applying  $a$  to it,  $\text{eff}^-(a)$  and  $\text{eff}^+(a)$  are two lists of negative and positive effects which describe how the state is modified by  $a$ . For a specific planning domain  $\langle \mathcal{P}, \mathcal{A} \rangle$ , a derived planning instance is obtained by specifying the tuple  $\Xi = \langle \mathcal{P}, \mathcal{A}, I, \mathcal{G} \rangle$ . Where  $I \subseteq \mathcal{P}$  is the initial state,  $\mathcal{G} \subseteq \mathcal{P}$  is the target goal state. The goal of a planner is to find a valid sequence of operators  $\pi \in \Pi$  that from  $I$  reaches  $\mathcal{G}$  while incurring the least cost. From a planning domain  $\langle \mathcal{P}, \mathcal{A} \rangle$  and a sequence of observations  $\pi = \{o_0, \dots, o_n\} \in \Pi$ , goal recognition can be performed by providing a model for

$$P(G|\Pi)P(\Pi) = \beta P(\Pi|G)P(G) \quad (6)$$

$\Pi$  is the set of valid partial plans inside  $\Xi$ , while  $P(G)$  is the explicit prior probability of the goals. In this setting a possible way to realize  $P(\Pi|G)$  is by computing, for every goal being considered, the cost of two optimal plans obtained from the planning instances  $\Xi = \langle \mathcal{P}, \mathcal{A}, I, \mathcal{G} \rangle$  and  $\Xi' = \langle \mathcal{P}, \mathcal{A}', I, \mathcal{G}' \rangle$ , where:

- $\mathcal{A}' = \mathcal{A}$  with action effects modified as:  
 $\forall a' \in \mathcal{A}'$ 
  - $\text{pre}(a') = \text{pre}(a) \wedge p_a$
  - $p_a = \wedge_i (x_{ai} = \text{arg}_{ai})$  if  $\text{arg}_{ai}$  is specified for action  $i$
  - $\text{eff}^+(a') = \text{eff}^+(a) \wedge e_0$  if  $a \in \pi$  and is the first of the observations (i.e.  $n = 0$ )
  - $\text{eff}^+(a') = \text{eff}^+(a) \wedge e_{n-1} \rightarrow e_n$  if  $a \in \pi$  and  $n \geq 1$
  - $\text{eff}^+(a') = \text{eff}^+(a)$  otherwise.
- $\mathcal{G}' = \mathcal{G} \cup e_n$ , where  $e_n$  is the effect predicate of the last action in  $\pi$ .

The latter planning instance achieves  $\mathcal{G}$  by producing a plan  $\pi_{\Xi'}$  which is constrained to contain the observations  $\pi$ , the former instead achieves  $\mathcal{G}$  by the means of an optimal plan  $\pi_{\Xi}$ . These two instances are used to evaluate the degree of rationality  $R(\pi, \mathcal{G})$  that the observations have towards the possible goals  $\mathcal{G}$ , as computed by the formula:

$$R(\pi, \mathcal{G}) = \frac{|\pi_{\Xi}|}{|\pi_{\Xi'}|} \quad (7)$$

$|\pi_{\Xi'}|$  is the cost of the optimal plan achieving  $\mathcal{G}$  using  $\Xi'$ , while  $|\pi_{\Xi}|$  is the cost of the optimal plan using  $\Xi$ . Given the hypothesis that an optimal plan  $\pi_{\Xi}$  exists, the following holds true:

$$|\pi_{\Xi}| \leq |\pi_{\Xi'}|, 0 \leq \frac{|\pi_{\Xi}|}{|\pi_{\Xi'}|} \leq 1 \quad (8)$$

All sequences of observations that induce plans with minimum cost have  $R(\pi, \mathcal{G}) = 1$ , while sub-optimal observation sequences towards  $\mathcal{G}$  have  $0 \leq R(\pi, \mathcal{G}) < 1$ . Hence, the lower  $R(\pi, \mathcal{G})$  is, the more sub-optimal it is to achieve that goal while being consistent with the observations.  $R(\pi, \mathcal{G})$  captures a measure of rationality of the observed actions in the sense that it evaluates whether and how their investment of resources is efficient towards the available goals. By this definitions optimal plans are the also the most rational (rationality is here equivalent as *predictability* in (Dragan and Srinivasa 2013) if we implicitly assume that the observer evaluates rational actions as predictable).

The probabilistic model for sequences of actions given the possible goals,  $P(\Pi|G)$ , can be finally obtained as a function of  $R$ , such as through a Boltzmann distribution:

$$\forall g \in G P(\pi|g) = \alpha e^{\frac{R(\pi, g)}{\tau}} \quad (9)$$

where  $\alpha$  is the normalizing factor which marginalizes over all of the possible candidate goals,  $\tau > 0$  is the distribution's temperature parameter. Given a sequence of observations, this model returns high probability for the goals that are rational to pursue, low probability otherwise.

### Production of legible plans with off-the-shelf planners

As discussed in Section 3, legible planning means to find a plan  $\pi$  such that its goal is easily discernible from a set of other candidates in an observer perspective. This is obtained by making the observer model  $P(G|\Pi)P(\Pi)$  to provide a



probability distribution over the goals as similar as possible to the one of the real goal with  $P(G = \hat{g}) = 1$ . With the introduction of the theory of mind  $T$ , Eq. 1 (and similarly Eq. 3,4) can be rewritten as:

$$\begin{aligned} \text{legibility}_T(\pi, g) &= \mathcal{H}(P_{g'}(G'), P(G'|\pi')P(\pi')) \\ g' &= T(g), G' = T(G), \pi' = T(\pi) \\ \{\pi, G\} &\in \Xi_{pl}, \{\pi', G'\} \in \Xi'_{pl}, \end{aligned} \quad (10)$$

where now the goal recognition is done utilizing the planning instance  $\Xi'_{pl}$  used by the observer to evaluate goals, which is obtained by applying  $T$  to  $\Xi_{pl}$ . Once inside  $\Xi'_{pl}$  goal recognition can be performed as previously described without further modifications. With a slight abuse of notation we use  $T$  to transform a planning instance in the broad sense and in particular also the plans and goals it yields.

Since plan legibility is non-monotonic (i.e.  $n$ -legibility( $\pi, g$ )  $\geq (n+1)$ -legibility( $\pi, g$ )) in the general case,  $n$ -legibility requires to exhaustively search the plan space up to a depth of  $n$  to then compute the legibility at that level. In order to leverage already existing planners, we propose to utilize diverse planning techniques (Katz and Sohrabi 2020). Diverse planning is the task of finding  $k$  plans that achieve a target goal which are evaluated based on their cost and diversity. We use diverse planning as an alternative to search the plan space and use the parameter  $k$  to define its size. After that, the  $\hat{n}$ -legibility of a plan is found by iteratively computing its  $i$ -legibility values for  $i = 1..n$ , averaging the results to get the final value. Based on diverse planning and Eq. 10, our proposed algorithm to find the legible plan is shown in Algorithm 1.

---

**Algorithm 1**  $\hat{n}$ -legible planning.

---

```

1: procedure LEGIBLE-PLANNING( $\Xi, T, G, \hat{g}, \gamma, k$ )
2:    $\Pi_{\hat{g}} \leftarrow \text{DIVERSE-PLAN}(\Xi, \hat{g}, k)$ 
3:    $\Xi' \leftarrow T(\Xi)$ 
4:    $\hat{g}' \leftarrow T(\hat{g})$ 
5:    $G' \leftarrow T(G)$ 
6:    $\bar{\pi} \leftarrow \underset{\pi \in \Pi_{\hat{g}}}{\text{argmax}} |T(\hat{\pi})| \text{-legibility} - \gamma|\pi|$ 
7: end procedure
```

---

Algorithm 1 performs three main operations: first it computes the set of plans  $\Pi_{\hat{g}}$  achieving the target goal in the planner's perspective, then it transforms the planning instance into its correspondings using the observer's model through the theory of mind  $T$ , finally, it finds the  $|\hat{\pi}|$ -legible plan also regularizing the result towards cheap plans, such that the efficiency of the returned plan can also be maintained. Legibility is computed in the observer's evaluated plan space  $\Pi'_{\hat{g}}$ , while the plan cost using the planner's task model.

### Illustrative Example

To illustrate the production of legible plans we propose the following example in the Human-Robot teaming scenario

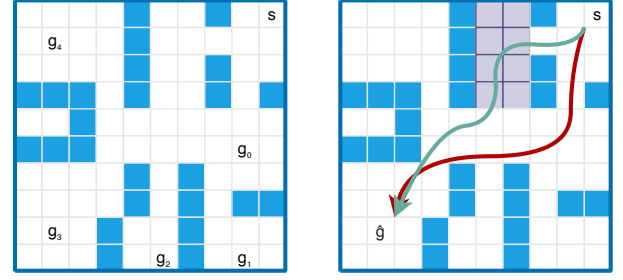


Figure 2: Legible planning in the rescue scenario. Left side: world grid that the commander believes the robot is planning for. This plan instance is estimated internally by the robot through a theory of mind, and doesn't have any obstacles. Right side: real world grid computed by the robot, which has the real goal and contains detected obstacles.

shown in Figure 2. It is a rescue scenario where an autonomous scout robot is moving in a dangerous environment (e.g. a laboratory filled with toxic gases) searching for people, while a commander supervises the operations on its computer interface from outside of the building. Both have access to a map of the environment in the form of a grid-world on which the robot's position is being tracked. We assume the commander doesn't have direct access to the internal goal of the autonomous robot, nor of other map features such as obstacles or rough terrain, which instead are detectable by the robot through its sensors. When the mission starts the robot communicates its goal explicitly to the commander through e.g. a speech interface, but in order to continuously and implicitly better communicate its goal, the autonomous robot estimates the commander's other expected possible goals, and produces a legible plan such that its true goal is best discriminated.

To make its plan more legible from the commander perspective, the robot should evaluate the possible candidate goals that its actions might communicate to the commander as being pursued (left side of Figure 2). In this case a possible estimate is that the commander thinks that all of the rooms of the building are candidate destinations for the plans. In this setting, the right side of Figure 2 compares the legible plan (green) with the optimal plan (red) for the same true goal  $g_3$ , computed in the robot perspective. The legible plan is more expensive, as it passes over some obstacles (purple), yet since it avoids as much as possible going towards other candidate goals, it results being more legible by the commander. It better communicates that  $g_3$  is the true pursued goal, and symmetrically, that the other candidate goals are not the ones being pursued.

The commander's perspective is estimated by the scout robot through its theory of mind. In this particular scenario the robot estimates that the commander believes the robot can move freely and that the environment is unobstructed, which is plausible in this scenario as only the robot can map the real environment. Since everything is computed locally to the robot, the commander is not required to perform any operation to communicate its mental state in the form of an

Domain	$ \mathcal{A} $	$ \mathcal{P} $	$ I $	$ G $	$ \bar{\pi} $
<i>intrusion</i>	9.0	11.0	1.0	4.75	17.15
<i>kitchen</i>	29.0	23.0	2.0	1.0	10.6
<i>satellite</i>	5.0	12.0	62.8	6.8	16.55
<i>campus</i>	22.0	12.0	1.0	2.75	4.925
<i>blocks-world</i>	4.0	5.0	14.4	4.95	15.25
<i>logistics</i>	6.0	3.0	22.7	2.3	31.25
<i>easy-ipc-grid</i>	3.0	8.0	227.4	1.0	17.2
<i>miconic</i>	4.0	8.0	518.6	6.6	24.85
<i>ferry</i>	3.0	7.0	99.3	8.9	28.27

Table 1: Average instance measures over the tested planning domains. The columns, from left to right are: number of operators, number of predicates, size of the initial state, size of the goal, length of optimal plan.

expected planning instance, such as through annotating the map.

## Evaluation

In order to benchmark our method on PDDL domains, we run Algorithm 1 on the following planning domains: *logistics*, *blocks-world*, *intrusion-detection*, *kitchen*, *campus*, *satellite*, *easy-ipc-grid*, *miconic*, *ferry* (Pereira, Oren, and Meneguzzi 2017)<sup>1</sup>, selecting for every domain 10 random planning instances. Table 1 shows average relevant metrics of the instances. To compute plans and perform goal recognition we utilize the *Forbid-Iterative*<sup>2</sup> (Katz and Sohrabi 2020) planner which is based on *Fast-Downward* (Helmert 2006) and performs diverse planning by iteratively creating plan instances forbidding previously found plans. The distance of the true goal distribution and the observer’s predicted distribution is realized through the cross-entropy function.

For every planning instance we applied Algorithm 1 by the following procedure: we selected one goal randomly chosen amongst a set of 4 candidates as true goal for that instance, and used diverse planning to compute  $k = 50, 100, 200$  plans toward it. For every plan its resulting legibility value is found by averaging the legibility values for all of its steps.

In the general case, planner and observer have two disjoint task models which makes their theory of mind non trivial and for which a qualitative evaluation would require an initial learning phase and model reconciliation to align them. However, since we want to test many PDDL domains, we want to evaluate many possible theory of minds statistically rather than qualitatively. In this setting we utilize a subset of all of the possible theory of minds that is easy to randomly sample. We focus more in particular on the theory of minds which treat the transformation between planning instances  $T$  as a function dropping parts of the actions or parameters. This corresponds in estimating the observer as

<sup>1</sup><https://github.com/pucrs-automated-planning/goal-plan-recognition-dataset>. Accessed March 20, 2021.

<sup>2</sup><https://github.com/IBM/forbiditerative>. Accessed March 20, 2021.

having strictly less or equal information as the planner when inferring goals.

More precisely we don’t assume, in the inferred plans, for the observations to appear in the exact succession in which they were gathered, but other actions are allowed to appear in between them. All of the observations must nevertheless appear in the observed order in their corresponding inferred plans. This simulates the observer not being able to recognize all of the actions. Additionally, we allow for observations to be partially instantiated, i.e. not all of their parameters are required to be specified, with missing parameters being rather inferred during plan recognition. This simulates how the observer doesn’t have access to all of the information contained in the observed actions and infers the missing parts. Even though we find it plausible also in real implementations, we selected this family of theory of mind because it is easy to simulate and sample, which each sampled  $T$  randomly dropping  $d\%$  of the plans actions and parameters. The tested values of  $d\%$  are 0, 20, 40, 60, 80.

For every planning instance the main gathered measures are the *legibility gain* and the *cost gain*. The legibility gain  $\mathcal{L}_{gain}$  corresponds to the ratio between the legibility of the optimal plan  $\bar{\pi}$  and the one of the most legible plan among the  $k$  generated plans,  $\pi_{\text{legible}}$ .

$$\mathcal{L}_{gain} = \frac{\hat{n}\text{-legibility}(\pi_{\text{legible}})}{\hat{n}\text{-legibility}(\bar{\pi})} \quad (11)$$

The legibility gain provides indication on how advantageous it is, in terms of legibility, to follow a legible plan rather than the optimal plan. The total legibility of a plan is computed by averaging all of the legibilities of its steps with equal weight. The other relevant indicator is the cost gain  $\mathcal{C}_{gain}$ , which instead indicates how costly (in plan length) the legible plan is when compared with the optimal plan.

$$\mathcal{C}_{gain} = \frac{|\pi_{\text{legible}}|}{|\bar{\pi}|} \quad (12)$$

These two measures together show the trade-off between plan legibility and cost. Table 2 illustrates the measures gathered on the tested domains. The obtained values for a domain are the averages of all the 10 instances belonging to it.

## Discussion

Our measurements show a positive correlation between legibility and cost: it was always possible to increase legibility in exchange of making plans more lengthy (see Table 3). The measured gains seems however to be strongly dependent on the tested domain. This is better highlighted in Figure 3, where legibility is compared with the percentage of random items being dropped from the plans. For example, in *logistics* the legibility peaks when none of the observations are dropped, while for *intrusion-detection* legibility increases linearly in the opposite direction. Dropping parts of observed plans relaxes plan recognition in the sense that more possible plans fits the same observations, which translates in making the sequences of observations more probable

d%	domain	$\gamma$									
		0		0.001		0.01		0.1			
		$\mathcal{L}_{gain}$	$\mathcal{C}_{gain}$	$\mathcal{L}_{gain}$	$\mathcal{C}_{gain}$	$\mathcal{L}_{gain}$	$\mathcal{C}_{gain}$	$\mathcal{L}_{gain}$	$\mathcal{C}_{gain}$		
0	campus	1.32	1.1	1.31	1.1	1.26	1.1	1.08	1.03		
	kitchen	1.43	1.26	1.37	1.26	1.16	1.11	1.0	1.0		
	satellite	1.38	1.34	1.35	1.32	1.2	1.2	1.01	1.01		
	intrusion	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0		
	ferry	1.58	1.3	1.48	1.29	1.16	1.08	1.02	1.02		
	logistics	3.08	3.21	2.33	2.39	1.43	1.17	1.06	1.01		
	blocks	2.44	6.24	1.87	6.24	1.24	2.54	1.0	1.0		
	grid	1.03	1.35	1.02	1.29	1.0	1.22	1.0	1.0		
	miconic	1.08	1.2	1.07	1.13	1.04	1.05	1.01	1.0		
20	campus	1.25	1.05	1.25	1.05	1.21	1.05	1.08	1.02		
	kitchen	1.58	1.13	1.52	1.11	1.29	1.06	1.02	1.05		
	satellite	1.28	1.29	1.26	1.26	1.15	1.19	1.01	1.01		
	intrusion	1.07	1.04	1.07	1.04	1.04	1.04	1.0	1.0		
	ferry	1.99	1.25	1.89	1.25	1.45	1.15	1.04	1.05		
	logistics	1.7	1.44	1.64	1.42	1.33	1.25	1.04	1.01		
	blocks	3.03	5.84	2.24	5.04	1.33	2.39	1.02	1.0		
	grid	1.05	1.41	1.04	1.38	1.02	1.27	1.0	1.0		
	miconic	1.13	1.16	1.12	1.16	1.07	1.03	1.02	1.0		
40	campus	1.19	1.15	1.18	1.15	1.15	1.08	1.04	1.02		
	kitchen	1.51	1.11	1.47	1.08	1.28	1.06	1.02	1.05		
	satellite	1.19	1.3	1.17	1.24	1.1	1.13	1.01	1.0		
	intrusion	1.12	1.08	1.12	1.08	1.07	1.06	1.0	1.01		
	ferry	1.44	1.45	1.4	1.42	1.2	1.17	1.02	1.0		
	logistics	1.53	1.54	1.47	1.49	1.24	1.27	1.04	1.01		
	blocks	2.91	5.75	2.2	5.13	1.3	2.02	1.03	1.0		
	grid	1.09	1.5	1.08	1.5	1.04	1.3	1.0	1.0		
	miconic	1.12	1.19	1.11	1.19	1.07	1.03	1.02	1.0		
60	campus	1.2	1.15	1.2	1.15	1.16	1.13	1.06	1.01		
	kitchen	1.63	1.16	1.59	1.16	1.36	1.11	1.05	1.05		
	satellite	1.14	1.25	1.13	1.19	1.08	1.13	1.01	1.0		
	intrusion	1.17	1.35	1.15	1.28	1.08	1.07	1.0	1.01		
	ferry	1.23	1.41	1.2	1.41	1.1	1.13	1.01	1.0		
	logistics	1.31	1.59	1.28	1.44	1.14	1.17	1.01	1.0		
	blocks	2.14	5.76	1.76	4.95	1.17	1.82	1.02	1.0		
	grid	1.11	1.87	1.1	1.82	1.04	1.49	1.0	1.0		
	miconic	1.06	1.19	1.05	1.16	1.03	1.02	1.01	1.0		
80	campus	1.2	1.17	1.2	1.17	1.16	1.17	1.05	1.01		
	kitchen	1.47	1.15	1.44	1.15	1.3	1.11	1.05	1.05		
	satellite	1.1	1.27	1.09	1.27	1.05	1.08	1.01	1.0		
	intrusion	1.16	1.37	1.14	1.35	1.07	1.11	1.0	1.01		
	ferry	1.09	1.37	1.08	1.23	1.02	1.09	1.0	1.0		
	logistics	1.15	1.44	1.13	1.32	1.06	1.1	1.01	1.0		
	blocks	1.59	5.83	1.43	4.49	1.11	1.65	1.02	1.0		
	grid	1.12	2.59	1.1	2.31	1.04	1.47	1.0	1.0		
	miconic	1.06	1.15	1.05	1.14	1.03	1.0	1.01	1.0		

Table 2: Computed plan legibility and cost measures.  $\mathcal{L}_{gain}$ : cost gain,  $\mathcal{C}_{gain}$ : legibility gain,  $d\%$ : percentage of dropped observations,  $\gamma$ : regularization factor towards cheap plans. Values are for  $k = 200$ .

toward all goals simultaneously. Our hypothesis is that planning domains with goal-specific actions will benefit more from dropping part of the observations rather than domains with universally applicable actions, as in the former case relaxed plans are still bound by the goal-specific actions. This currently gathered data indicates that it is difficult to predict a priori how legibility will behave in experiments on real data.

The results further show that the regularization factor  $\gamma$  plays an important role. In domains such as *blocks-world* or *logistics* some action loops can arbitrarily increase the legibility towards the goals. This happens to the expense of a huge drop of performance of the plans. In those domains, without regularization legible plans could reach a length up to 3-6 times higher than the optimal plans for the same instances.

## Conclusions

In this paper we proposed a procedure to compute legible plans using off-the-shelf PDDL planners. Our formulation is based on a probabilistic formulation of goal recognition

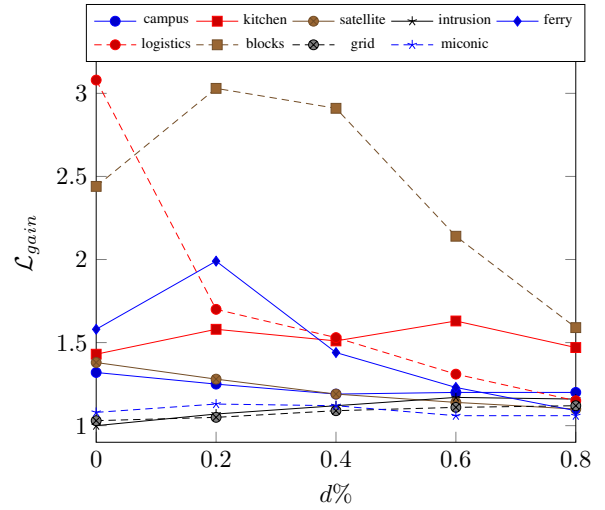


Figure 3: Legibility gain for various percentages of dropped parameters.

	$k$	50	100	200
$\gamma = 0$	$\mathcal{L}_{gain}$	1.31	1.43	1.54
	$\mathcal{C}_{gain}$	1.36	1.47	2.09
$\gamma = 0.01$	$\mathcal{L}_{gain}$	1.13	1.18	1.16
	$\mathcal{C}_{gain}$	1.18	1.21	1.36
$\gamma = 0.1$	$\mathcal{L}_{gain}$	1.01	1.02	1.01
	$\mathcal{C}_{gain}$	1	1	1

Table 3: Average over all of the domains of legibility and cost gains for increasing values of  $k$ .

and models the observer’s task model as being estimated by the planner. The integration of these two task models is obtained by a procedure based on theory of mind which transforms the planner’s task model into the same from the observer’s perspective. The introduction of theory of mind for legibility is crucial as it describes how planner and observer tasks models are connected, and makes the planner to know how its actions are perceived.

We further proposed an illustrative example based on a rescue scenario with a robot and a human commander explaining how legibility behaves in this context. In Human-Robot Teaming contexts we would like to highlight that since legibility leverages the theory of mind owned by the robot, it doesn’t require inputs by the commander such as labeling of the scene. This is particularly interesting in the context of highly autonomous robots. Additionally, a statistical evaluation over several planning domains showed how the proposed algorithm successfully generates plans that are more legible than the optimal plans, also indicating how legibility is a trade-off with plan cost. Our investigations show that this relation between legibility and cost is strongly dependent on the domain and theory of mind being utilized. In our statistical experiments we tested the family of theory of minds that strictly reduce the amount of information for goal recognition.

Apart from illustrating the broad possibility of increasing legibility of plans, these current tests cannot reach further conclusions a priori valid on every possible domain, which rather require investigation on the specific domain and theory of mind being utilized.

Future work regards the investigation of legibility as perceived by humans in real scenarios. In particular on whether the proposed model for plan legibility, which is based on the concept of rationality, effectively capture human judgement of legibility in domains specific for human-robot interaction.

## References

- Chakraborti, T.; Sreedharan, S.; Zhang, Y.; and Kambhampati, S. 2017. Plan explanations as model reconciliation: Moving beyond explanation as soliloquy. *arXiv preprint arXiv:1701.08317*.
- Chakraborti, T.; Kulkarni, A.; Sreedharan, S.; Smith, D. E.; and Kambhampati, S. 2019. Explicability? legibility? predictability? transparency? privacy? security? the emerging landscape of interpretable agent behavior. In *Proceedings of the international conference on automated planning and scheduling*, volume 29, 86–96.
- Devin, S., and Alami, R. 2016. An implemented theory of mind to improve human-robot shared plans execution. In *2016 11th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, 319–326. IEEE.
- Dragan, A., and Srinivasa, S. 2013. Generating legible motion.
- Dragan, A., and Srinivasa, S. 2014. Integrating human observer inferences into robot motion planning. *Autonomous Robots* 37(4):351–368.
- Dragan, A. D.; Bauman, S.; Forlizzi, J.; and Srinivasa, S. S. 2015. Effects of robot motion on human-robot collaboration. In *2015 10th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, 51–58. IEEE.
- Dragan, A. D.; Lee, K. C.; and Srinivasa, S. S. 2013. Legibility and predictability of robot motion. In *2013 8th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, 301–308. IEEE.
- Fox, M.; Long, D.; and Magazzeni, D. 2017. Explainable planning. *arXiv preprint arXiv:1709.10256*.
- Hellström, T., and Bensch, S. 2018. Understandable robots - What, Why, and How. *Paladyn, Journal of Behavioral Robotics* 9(1):110–123.
- Helmert, M. 2006. The fast downward planning system. *Journal of Artificial Intelligence Research* 26:191–246.
- Katz, M., and Sohrabi, S. 2020. Reshaping diverse planning. In *AAAI*, 9892–9899.
- Kulkarni, A.; Zha, Y.; Chakraborti, T.; Vadlamudi, S. G.; Zhang, Y.; and Kambhampati, S. 2019. Explicable planning as minimizing distance from expected behavior. In *AAMAS*, 2075–2077.
- Kulkarni, A.; Srivastava, S.; and Kambhampati, S. 2019. A unified framework for planning in adversarial and cooperative environments. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, 2479–2487.
- Kulkarni, A.; Srivastava, S.; and Kambhampati, S. 2020. Signaling friends and head-faking enemies simultaneously: Balancing goal obfuscation and goal legibility. In *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems*, 1889–1891.
- MacNally, A. M.; Lipovetzky, N.; Ramirez, M.; and Pearce, A. R. 2018. Action selection for transparent planning. In *AAMAS*, 1327–1335.
- McDermott, D. 1998. Pddl-the planning domain definition language.
- Meijering, B.; Van Rijn, H.; Taatgen, N.; and Verbrugge, R. 2011. I do know what you think i think: Second-order theory of mind in strategic games is not that difficult. In *Proceedings of the Annual Meeting of the Cognitive Science Society*, volume 33.
- Miura, S., and Zilberstein, S. 2020. Maximizing plan legibility in stochastic environments. In *Proceedings of the 19th International Conference on Autonomous Agents and Multi-Agent Systems*, 1931–1933.
- Pereira, R. F.; Oren, N.; and Meneguzzi, F. 2017. Landmark-based heuristics for goal recognition. In *Thirty-First AAAI Conference on Artificial Intelligence*.
- Persiani, M., and Hellström, T. 2020. Intent recognition from speech and plan recognition. In *International Conference on Practical Applications of Agents and Multi-Agent Systems*, 212–223. Springer.
- Zhang, Y.; Sreedharan, S.; Kulkarni, A.; Chakraborti, T.; Zhuo, H. H.; and Kambhampati, S. 2017. Plan explicability and predictability for robot task planning. In *2017 IEEE international conference on robotics and automation (ICRA)*, 1313–1320. IEEE.

# A Sampling-Based Optimization Approach to Handling Environmental Uncertainty for a Planetary Lander

Connor Basich,<sup>1,2</sup> Daniel Wang,<sup>1</sup> Joseph A. Russino,<sup>1</sup> Steve Chien,<sup>1</sup> and Shlomo Zilberstein<sup>2</sup>

<sup>1</sup>Jet Propulsion Laboratory, California Institute of Technology, Pasadena, California, {firstname.lastname}@jpl.nasa.gov

<sup>2</sup>University of Massachusetts Amherst, Amherst, Massachusetts, {cbasich, shlomo}@cs.umass.edu

## Abstract

Planning for autonomous operation in unknown environments poses a number of technical challenges. The agent must ensure robustness to unknown phenomena, unpredictable variation in execution, and uncertain resources, all while maximizing its objective. These challenges are exacerbated in the context of space missions where uncertainty is often higher, long communication delays necessitate robust autonomous execution, and severely constrained computational resources limit the scope of planning techniques that can be used. We examine this problem in the context of a Europa Lander concept mission where an autonomous lander must collect valuable data and communicate that data back to Earth. We model the problem as a hierarchical task network, framing it as a utility maximization problem constrained by a monotonically decreasing energy resource. We propose a novel deterministic planning framework that uses periodic replanning and sampling-based optimization to better handle model uncertainty and execution variation, while remaining computationally tractable. We demonstrate the efficacy of our framework through simulations of a Europa Lander concept mission in which our approach outperforms several baselines in utility maximization and robustness.

## Introduction

Planning in domains with large uncertainty and very low margins for error has long been a challenge in the AI planning community. While numerous techniques have been developed over the years, many are rendered impractical or infeasible by the technical constraints that many physical robotic systems face. In space-based scenarios, the uncertainty is often higher, the margins for error lower, and the constraints more severe. Traditional approaches to planning in space-based domains have consequently utilized deterministic or sampling-based planning methods (Chi et al. 2019; Chi, Chien, and Agrawal 2020) which are fast and computationally inexpensive, and can be very effective when paired with a priori expert domain knowledge. Recent work has investigated how periodic replanning, flexible execution, and online model updates can be used in conjunction with search-based deterministic planning to improve the efficacy and robustness of the overall plans generated and executed by a space-based robotic system (Wang et al. 2020).

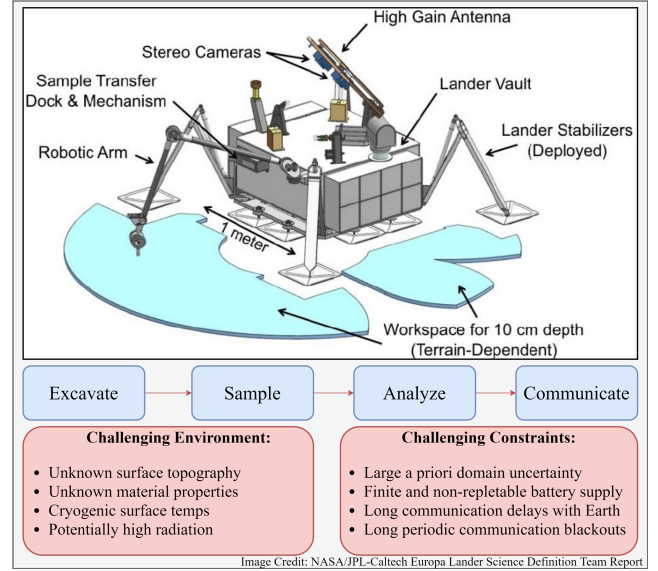


Figure 1: An illustration of the Europa Lander concept mission in which an autonomous lander is tasked with performing *in-situ* analysis of sampled surface material and communicating the collected data to Earth.

While often effective in practice, these approaches do not actively consider model uncertainty and off-nominal behavior during plan generation, and are hence *reactive* in how they handle the uncertainty faced by the system over its environment. In this work, we propose a planning framework that extends the algorithm from (Wang et al. 2020) by *proactively* anticipating deviations from nominal execution by incorporating domain uncertainty into the plan generation, creating plans that are more robust to these deviations without sacrificing solution quality in the nominal case. We examine the efficacy of our approach in the context of a proposed concept mission in which a lander is tasked with analyzing surface material to acquire valuable scientific data by performing *in-situ* analysis of samples excavated from the surface of the Jovian moon Europa, and communicating that data back to Earth (Hand 2017).

This concept mission entails several challenges that differentiate it from prior missions. First, a priori knowledge of the environment is severely limited and uncertain. Second,

the system’s battery supply is finite and non-repletable (i.e. there is no possible power generation). Third, communication with Earth is constrained by two factors: (1) due to the large distance to Earth, there are long communication delays when communicating with Earth. As a result, ground-in-the-loop operations cannot be relied on as the system will be losing battery while waiting for communications and hence the lander must be capable of operating fully autonomously. And (2), due to Jovian occlusion, the lander will be faced with long periodic communication blackouts (roughly 42 out of every 84 hours) which constrain when the lander is capable of downlinking the data it has collected to Earth.

As utility is only assigned to data that is acquired and *successfully downlinked to Earth*, and none for data collected but not downlinked, in order for the Lander to be successful it needs to carefully manage the trade-off between data acquisition and communication. Furthermore, it must do this while constrained by a finite and monotonically decreasing battery supply, limited knowledge of its environment, and limited communication with Earth. As a result, for the mission to be successful, the system requires an autonomous planning and execution framework that is (1) computationally efficient; (2) robust to unprecedented levels of uncertainty; but still (3) capable of maximizing overall utility.

We model the problem as a *hierarchical task network* (HTN) (Nau et al. 2003) due to the structured nature of the tasks that the lander can perform, and consequently use an anytime heuristic-search algorithm designed for solving HTNs (Wang et al. 2020) as the primary subroutine of the proposed approach. Our approach is based on principles from Hindsight Optimization (HOP) (Chong, Givan, and Chang 2000) and works by hypothesizing a set of sampled scenarios that the system may face, planning for each scenario, and evaluating each of the sampled plans’ performances across all scenarios. The plan with the highest weighted value is selected (this can be viewed as an approximation to maximizing expected utility). As the planner is deterministic, we also perform periodic replanning to ensure that the system’s performance does not deviate too far from expectation. This approach has similarities with determinization-based methods that have been highly successful in solving large Markov decision processes (MDPs) (Yoon, Fern, and Givan 2007; Yoon et al. 2008; Pineda and Zilberstein 2014). While MDPs and other stochastic sequential decision making models have been had success in many settings, they are computationally expensive and ill-suited for domains with concurrent actions and continuous states such as that which is considered in this paper.

We empirically validate our approach in a simulated Europa-like concept mission. The execution system we use in our simulations is MEXEC, an integrated planner and executive first built for NASA’s Europa Clipper mission (Verma et al. 2017), to better react to variations in both environment and execution. Finally, to compensate for both uncertain model priors and the deterministic nature of the planner, the framework replans on a periodic basis. We present empirical results against two base-line approaches similar to those used in prior missions: a greedy planner with replanning and an anytime heuristic-search based plan-

ner with replanning. We show that the proposed planning framework, **HTNSearch-PHRA**, is more effective on average and more robust to uncertainty across five different mission scenarios. In addition, we analyze the effect of increasing the size of the hypothesized scenario set by comparing the performance of the algorithm with four different sets of hypothesized scenarios.

## Problem Description

### Domain Overview

The primary goal of the Europa Lander concept mission is to excavate and sample the moon’s surface, analyze the sampled material for signs of biosignatures, and communicate that data back to Earth (Hand 2017). Additionally, there are secondary objectives to take panoramic imagery of the European surface and collect seismographic data. Lander operations are therefore limited to primary objective tasks, secondary objective tasks, and data communication. This provides significant structure to the problem, since the concept mission clearly defines the sequence of actions required to achieve these goals. Figure 2 displays an example of a potential execution trace of tasks that satisfies the minimum requirements of the mission, and illustrates the inherent structure of the concept mission amongst the possible tasks.

As a minimum requirement, the lander should excavate a trench in the European surface, collect three samples from that site, analyze those samples, and communicate that data to Earth. The basic requirements of a mission would require only a single site to be excavated. However, there is value in excavating additional sites, because the material at different sites may possess different properties. In addition, the lander may choose to resample the same location to, for example, verify the discovery of a biosignature at that location. In the baseline concept mission, all three of the lander’s samples are chosen from the same target. Note that after the first site is excavated, no further excavations are needed to sample from that trench; all three sampling activities can share a single excavation site. After excavation and sample collection, samples must be transferred into scientific instruments that analyze the material and produce data products. Then, for a mission to achieve any actual utility, those data products must be communicated back to Earth. Because communication is difficult and energy intensive, the lander may choose to compress data lossily which reduces both the energy required to communicate the data and its utility, if the expected utility of this action is higher.

In addition to sampling tasks, the lander may engage in seismographic data collection and period panoramic imagery tasks. These are considered lesser goals, with lower utility associated with their completion. As such, the data products that these tasks generate are considered to have lower value. However, these tasks also involve no surface interaction, and have less uncertainty associated with them.

It is important to note that utility is only achieved when data is downlinked back to Earth. This is true for both the sampling and seismograph/panorama tasks. Some excavation sites or sampling targets may provide more utility than others if, for example, one of those targets has a positive



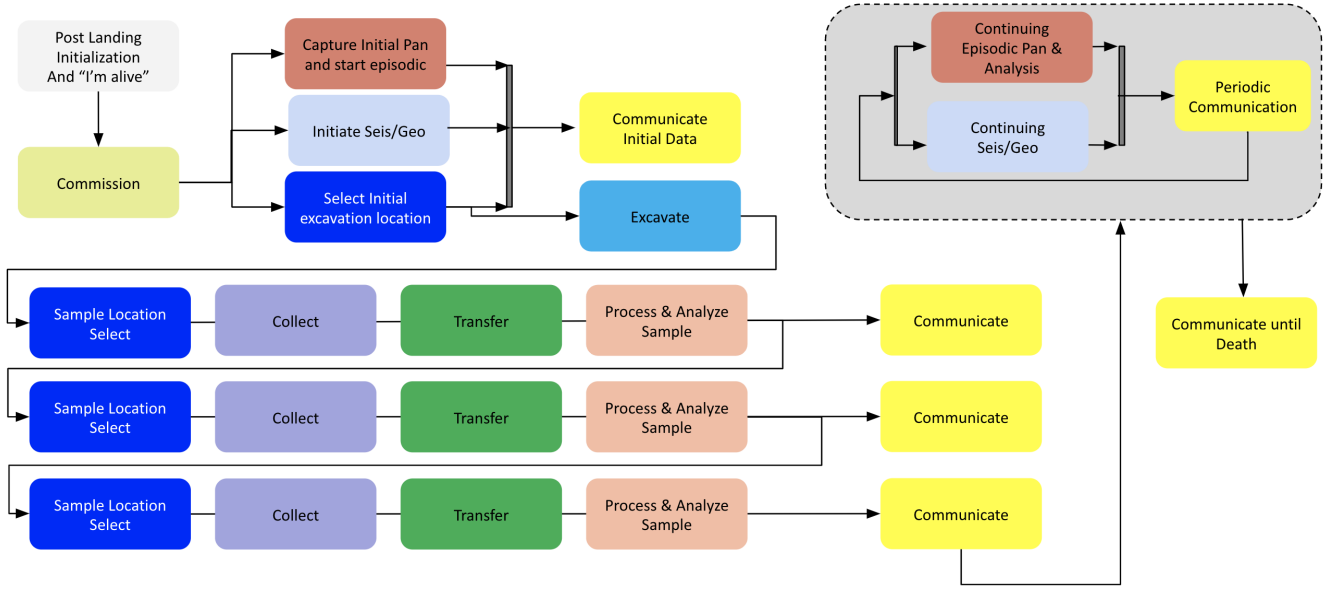


Figure 2: An illustration of a potential execution trace of an example task network for the Europa Lander concept mission that satisfies the minimum requirements of the mission.

biosignature and the other does not. However, regardless of the quality of the material that the lander samples, no utility is achieved unless that data is communicated. This dynamic means that while potential utility is generated during the sampling and analysis phases, it is only realized by completing relevant communication tasks.

The Europa Lander concept mission is also constrained by a finite battery that cannot be recharged. Battery life is a depletable resource, and the algorithm must plan accordingly to maximize utility in the face of this constraint. In addition to this challenge, the surface characteristics of Europa are uncertain, and any prior mission model that is generated before landing are assumed to be inaccurate. In particular, the energy consumption of the excavation and sample collection tasks is largely unknown. There is also significant variation in the utility of any given sample, since the value of sampling a given target on Europa depends on whether the material is scientifically interesting, e.g. whether a biosignature is present.

## Problem Formulation

We model this problem using a hierarchical task network (HTN) to compile the domain-specific knowledge of the dependency structure into the task network. HTNs have been used successfully in industrial and other real-world applications to improve the tractability of planning problems in systems such as SHOP2 (Nau et al. 2003) and SHOP3 (Goldman and Kuter 2019). In an HTN, hierarchical tasks are decomposed into a set of subtasks. We refer to the higher-level tasks as *parent tasks*, and refer to their children as *subtasks*. Parent tasks may decompose into a number of different partially ordered sets of subtasks; we refer to each of these

sets as a potential *decomposition* of that parent task. Finally, we refer to tasks with no decompositions as *primitive tasks*. These primitive tasks represent tasks that the lander can be directly commanded to perform. Decompositions enable us to significantly reduce the planning search space as we can treat all subtasks of a parent task as a singular block during the planning process; for example, the model treats “excavate, sample, transfer, analyze” as a single unit and schedules the subtasks accordingly.

Formally, an input to the problem is a set  $\mathcal{T} = \{T_1, \dots, T_n\}$  and a system state  $S$ . Each  $T_i$  is a task that is represented by the tuple  $\langle p_i, d_i, e_i, u_i, s_i, C_i, W_i, D_i \rangle$  where  $p_i$  is the priority of the task,  $d_i$  is the expected duration of the task,  $e_i$  is the expected rate of energy usage by the task,  $u_i$  is the expected utility of the task,  $s_i$  is the preferred start time of the task,  $C_i$  is the set of constraints that must be satisfied for the task to be scheduled,  $W_i = \{[t_{i1}, t_{i2}], \dots, [t_{in-1}, t_{in}]\}$  is the set of time windows that the task can be scheduled in, and  $D_i = \{T_{i1}, \dots, T_{im}\}$  is the partially ordered set of decompositions of the task, which can be empty if  $T_i$  is a primitive task. The state  $S$  is represented by a collection of continuous and discrete features including the remaining battery supply, the current data load, the current time, and all features required to model task constraints.

There are four main parent task types in the mission model. The first is a *Preamble* which consists of post-landing initialization and other one-time initialization tasks, and must be executed immediately upon landing. Second are data acquisition tasks which consist of excavation, sample collection, sample transfer, and sample analysis tasks. Excavation can take place in one of three excavation sites, and may be skipped if a site has previously been excavated. For collection tasks, the lander may choose between one of sev-

eral collection targets at any given excavation site (repeated sampling of the same target is allowed with no penalty). The analysis task returns the dataload acquired from a given sample upon completion; dataload represents the maximum potential utility that the acquired data provides upon successful downlink to Earth. Third, there are Seismograph/Panorama tasks which consist of seismographic data collection and panoramic image data collection; these tasks provide less data but are more reliable in their execution. Fourth is the communication task which decomposes into either a single, or a sequence of two, communication(s), either of which can be of raw data or compressed data. In this problem, we assign utility solely to the successful completion of a communication task, where communicating raw data provides greater utility but consumes more energy than communicating compressed data; note that both communication tasks consume the same amount of dataload. Utility is assigned to tasks *a priori* but we note that in practice may likely be updated online as new information is obtained by the system.

## Approach

The underlying planning method employed in this approach relies on heuristic search. Search-based planning algorithms have been popular for a number of years as they (1) do not require that the full state space is evaluated to produce a solution (Hansen and Zilberstein 2001), (2) are often anytime algorithms that can return a solution at any point during runtime (Zilberstein 1996), and (3) can easily leverage heuristics to reduce the computational burden while still achieving high performance (Bonet and Geffner 2001; Hoffmann and Nebel 2001; Korf 1990). In this case, all three of these properties are highly desirable, and influenced our decision to utilize a heuristic search-based planning algorithm.

### Heuristic HTN Search

The main planning algorithm, Algorithm 1, relies on the heuristic search approach for solving HTNs described in (Wang et al. 2020) – which we henceforth refer to as HTNSearch – as its primary subroutine. We offer a brief discussion of their algorithm.

HTNSearch first performs a pre-processing step in which all task decompositions are flattened into a single layer so that parent tasks are simply linear chains of primitive tasks. By flattening the decompositions, we can assign specific utility and energy values to each parent task as there is no ambiguity over which decomposition it is represented by. Note that this step can be performed offline and only needs to be performed once.

Next, HTNSearch initializes the search graph on the newly flattened task network, where nodes hold partial plans and edges hold (flattened) task decompositions or primitive tasks. As the algorithm is deterministic, both the cost and utility associated with any node is the sum over the tasks in the partial plan for the respective value. Finally, the algorithm performs a heuristic branch and bound search procedure over the search graph where the search is bounded by both the feasibility of partial plans (their total energy cost

cannot exceed the current battery supply, and the plan adheres to inter-task constraints), and optional computational constraints on the number of explorable nodes. For any plan and decomposition pair,  $(P, d)$ , a density based heuristic value of  $utility(P) + \frac{utility(d)}{cost(d)}$  is used and ties are broken in favor of lower cost.

The main limitation of this approach, which drives the motivation for Algorithm 1, is that it is a deterministic algorithm for a non-deterministic domain. In other words, the plan that is produced assumes that the future will operate exactly according to expectation. (Wang et al. 2020) address this issue primarily through the use of online model updates and frequent replanning to ‘course-correct’ the system online. This idea is similar to that of determinization-based approaches for solving very large Markov decision processes, such as FF-Replan (Yoon, Fern, and Givan 2007), which have been shown to perform well in the MDP setting, particularly in the infinite horizon case, but are not robust to dead-ends. Hence, we propose a planning algorithm that *proactively* considers off-nominal behavior and execution during plan time, rather than just *reactively* responding to off-nominal behavior and execution. We demonstrate through empirical evaluations that our approach is indeed more robust to off-nominal scenarios than the standard heuristic search, performing as well or better in both positive and negative scenarios.

### HTNSearch with Post Hoc Robustness Analysis

The proposed planning algorithm, the pseudocode for which can be seen in Algorithm 1, is intuitively straightforward. The algorithm takes in as input an instance of the Europa Lander problem modeled as a hierarchical task network,  $\mathcal{T}$ , and begins by producing a set of hypothesized scenarios via the subroutine *hypothesizeScenarios* (line 3). The function *hypothesizeScenarios* takes in the current task network that is being planned on and the system’s current state, and returns a set of hypothesized scenarios  $\mathcal{H}$ . A scenario,  $h \in \mathcal{H}$ , is comprised of an initial state and an instantiation of the parameterized domain. This function is left general as its implementation will be both domain and purpose specific. For example, in our simulations we hypothesize multiple scenarios where the current battery life of the lander is varied up or down to represent the uncertainty over the “true” battery life of the lander. However, other parameters may be varied such as the time or energy to perform various tasks, the likelihood of positive data from different samples, or the possibility of excavated sites collapsing. These scenarios may be developed using expert knowledge *a priori*, or may be generated online by drawing from distributions that parameterize the domain model.

For each hypothesized scenario  $h \in \mathcal{H}$ , the algorithm first instantiates  $h$  by updating the relevant parameters of  $\mathcal{T}$ , and calls HTNSearch on the newly instantiated task network to produce the best-found plan  $\mathcal{P}$  (lines 5-7) within the computational constraint. The plan  $\mathcal{P}$  is evaluated on each scenario  $h' \in \mathcal{H}$  by computing the expected utility following  $\mathcal{P}$ ,  $V^{\mathcal{P}}(\mathcal{T}, h')$  (line 10) using a stochastic execution graph subroutine. The observed expected utility is weighted by a

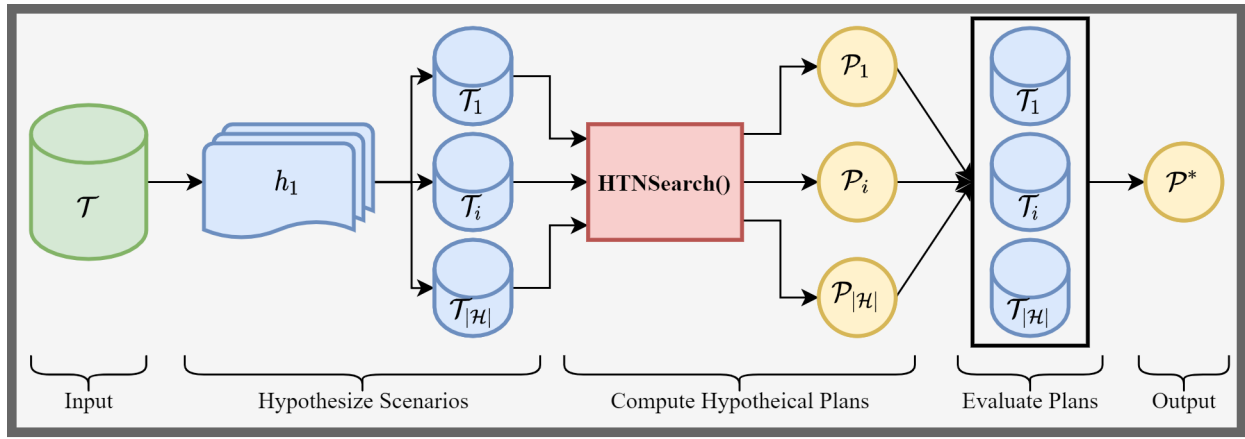


Figure 3: An illustration of the HTNSearch-PHRA algorithm. A tasknet,  $\mathcal{T}$ , and a set of hypothesized scenarios,  $h_1, \dots, h_{|\mathcal{H}|}$ , produce set of instantiated task networks,  $\mathcal{T}_1, \dots, \mathcal{T}_{|\mathcal{H}|}$ . Each  $\mathcal{T}_i$  is comprised of the input task network with certain parameters modified by the scenario; for instance the current state of the system or the cost and utility of various tasks. The planning algorithm, HTNSearch, is ran on each  $\mathcal{T}_i$  to produce a plan,  $\mathcal{P}_i$  that is the best plan found for that instantiated task network. Each plan,  $\mathcal{P}_i$  is evaluated on all  $\mathcal{T}_i$  to produce a cumulative score that is comprised of a weighted sum of expected utilities, where the weights are determined by the likelihood of the hypothesized scenario. The plan that has the highest score,  $\mathcal{P}^*$ , is returned.

function *getScenarioWeight* and the weighted value is added to the total score of the plan,  $\mu^{\mathcal{P}}$  (9-10). Finally, the plan that had the highest total score is returned.

The function *getScenarioWeight* returns a real valued number in  $[0, 1]$ , given an HTN and a scenario. In our case, we compute the Legendre-Gauss quadrature weights assuming that the parameters relevant to the hypothesized scenarios are normally distributed. In general we believe that any relative likelihood-based weighting scheme will work, however we note that offline optimization of these weights may be worthwhile particularly when the scenarios considered are over multiple different task network parameters.

We compute the expected utility of the plan  $\mathcal{P}$  using a stochastic subroutine that builds the non-deterministic execution graph of  $\mathcal{P}$  given the distributions which parameterize the task network (energy cost, duration, data, and utility). Computing the expected utility of a deterministic plan in a stochastic domain is significantly cheaper than computing a fully stochastic policy in the first place, and still allows us to observe a more accurate evaluation of each plan.

Finally, as the HTN search algorithm dominates the other subroutines in terms of runtime complexity, the runtime of Algorithm 1 is  $\sim |\mathcal{H}|$  times the runtime of HTNSearch when  $|\mathcal{H}|$  is small. However, as  $|\mathcal{H}|$  grows, the computation spent evaluating plans grows at a rate of  $O(|\mathcal{H}|^2)$ . Furthermore, there are diminishing returns to increasing the number of hypothesized scenarios considered, particularly when adding very low likelihood scenarios to  $\mathcal{H}$ . An analysis of this can be found further in the paper. Ultimately, the problem of determining which, and how many, scenarios to include in  $\mathcal{H}$  is an important element in balancing the trade off between efficiency and effectiveness.

---

#### Algorithm 1: HTNSearch-PHRA

---

**Input:** A hierarchical task network  $\mathcal{T}$  and state  $s$

**Result:** A plan  $\mathcal{P}^*$

---

```

1  $\mathcal{P}^* \leftarrow \text{None}$ 
2  $\mu^* \leftarrow -\infty$ 
3  $\mathcal{H} \leftarrow \text{hypothesizeScenarios}(\mathcal{T}, s)$ 
4 for  $h \in \mathcal{H}$  do
5    $\hat{\mathcal{T}} \leftarrow \text{instantiateScenario}(\mathcal{T}, h)$ 
6    $\mathcal{P} \leftarrow \text{HTNSearch}(\hat{\mathcal{T}})$ 
7    $\mu^{\mathcal{P}} \leftarrow 0$ 
8   for  $h' \in \mathcal{H}$  do
9      $\gamma \leftarrow \text{getScenarioWeight}(\mathcal{T}, h')$ 
10     $\mu^{\mathcal{P}} \leftarrow \mu^{\mathcal{P}} + \gamma V^{\mathcal{P}}(\mathcal{T}, h')$ 
11  end
12  if  $\mu^{\mathcal{P}} > \mu^*$  then
13     $\mathcal{P}^* \leftarrow \mathcal{P}$ 
14     $\mu^* \leftarrow \mu^{\mathcal{P}}$ 
15  end
16 end
17 return  $\mathcal{P}^*$ 

```

---

## Experiments

### Experimental Setting

To evaluate the performance of Algorithm 1, we compared against two baselines: HTNSearch and GREEDY. In GREEDY, priorities were assigned a priori to each task decomposition, and the planner greedily attempts to schedule tasks in order of priority at each planning cycle, skipping over tasks if they cannot be scheduled due to conflicts or violated constraints. Priorities are assigned offline using a com-

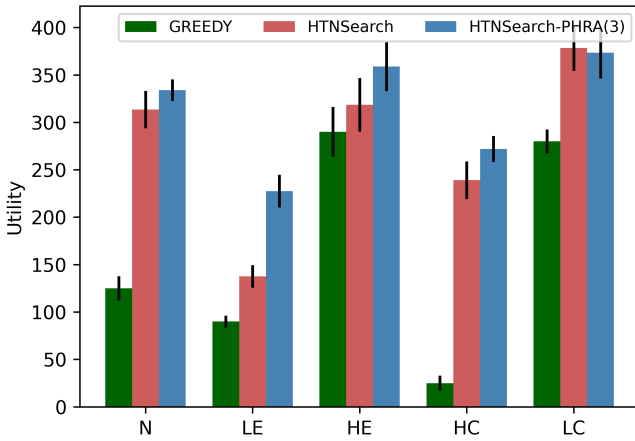


Figure 4: Utility achieved by each planning algorithm on all five scenarios. Values shown are mean and standard error over 10 trials.

bination of hard-coded domain knowledge (e.g. the Preamble must have the highest priority) and Monte carlo trials on sampled priority orderings across the input task.

The domain used for our simulations is described in Section 2. We recall a few key points here. First, the system is tasked with excavating the surface of the moon, collecting samples to analyze which produces data, and then communicating that data back to Earth. Only data that has been successfully communicated provides utility. Second, the agent’s battery supply is consumptive and *non-rechargeable*, and the system maintains a base Hotel load – a constant energy usage – even when sleeping. Third, communication with earth is only possible in cycles due to Jovian occlusion (every other 250 time units in our simulations). Consequently, the system must be able to effectively manage a monotonically decreasing battery supply and fixed time windows for communication, while still performing tasks that produce data, to actually receive any utility. Furthermore, it must execute these tasks in a domain with large a priori uncertainty.

We therefore considered 5 different stochastic scenarios: (1) Nominal, (2) Low Energy, (3) High Energy, (4) High Consumption, (5) Low Consumption. In scenario (1) there are no off-nominal, or unexpected, events or behaviors that occur during simulation. In scenarios (2) and (3) there is a sudden change ( $\pm 20\%$ ) in remaining battery life that occurs 500 time units into the simulation as the battery recalibrates. In scenarios (4) and (5) energy is stochastically drained or added to the observed remaining battery supply at every state update. In all scenarios, task parameters such as energy rates, duration, and data load are all drawn from low variance Gaussians centered around pre-determined means at runtime. Each scenario was simulated 10 times for each planning algorithm to account for this execution variation.

In these experiments, we specifically focused on off-nominal variations on remaining battery for three reasons. The first is that, historically, battery measurements have come with large uncertainty. The second is that battery is the most valuable resource in this domain, as time only mat-

ters in that there is a constant minimum Hotel load, and zero remaining battery supply is a terminal absorbing state. The third reason is that deviations in battery supply or energy consumption act as direct proxies for most other off-nominal behavior (extra time to complete a task, getting stuck, failing to perform a task requiring it to be repeated, etc.). However, the algorithm presented is not relegated to such a constraint, and in general can capture arbitrary scenarios.

## Experimental Results

The main results of our experiments can be seen in Figures 4 and 5. The first experiment compares the performance of three algorithms: GREEDY, HTNSearch, and HTNSearch-PHRA (3). The second experiment compares the performance of four variations of HTNSearch-PHRA, where the size of the hypothesized scenario set is increased: HTNSearch-PHRA (1), HTNSearch-PHRA (3), HTNSearch-PHRA (5), and HTNSearch-PHRA (7).

**Cross-Method Comparison** Figure 4 shows the mean and standard error of the utility achieved by each planning algorithm across the five planning scenarios. GREEDY performs well when the scenario is an optimistic scenario as the system ends up with enough battery to perform all of the high priority tasks without issue; in particular this means sampling more than the other approaches as GREEDY does not perform the more conservative actions such as Seis/Pan which produce less data but are more stable tasks. However, in the pessimistic scenarios, where the battery supply ends up being less than expected, GREEDY performs extremely poorly, using up too much energy early on in the mission sampling new targets to collect data, leaving insufficient energy to communicate all of the data back. This demonstrates the brittleness of a greedy approach in the context of a domain with large uncertainty.

HTNSearch performs well overall, and notably better than Greedy, as expected based on the results from Wang et al. (Wang et al. 2020). However, because its plans are always conditioned on nominal behavior and a lack of unexpected events, it is always reacting to off-nominal deviations, leading to poorer results in both the Low Energy (LE) scenario and the High Consumption (HC) scenario. Notably, the performance is better in the High Consumption scenario, where the negative impact on the battery supply is constant but small, than in the Low Energy scenario where the impact is large, sudden, and unexpected. The former scenario allows for constant reactive replanning to work as a viable strategy for managing the off-nominal performance, but if too much energy has been spent prior to encountering the sudden drop in battery supply in the latter scenario, there is no recourse for the system. The performance in the optimistic scenarios, High Energy and Low Consumption, indicate that the system is able to communicate an extra dataload on average in the low consumption case. Similar to above, we believe that the reason for this is that the system can constantly react to the small increases in battery supply in the Low Consumption scenario, but cannot plan to take advantage of the “excess” energy it encounters in the High Energy case.

On the other hand, HTNSearch-PHRA, which proac-

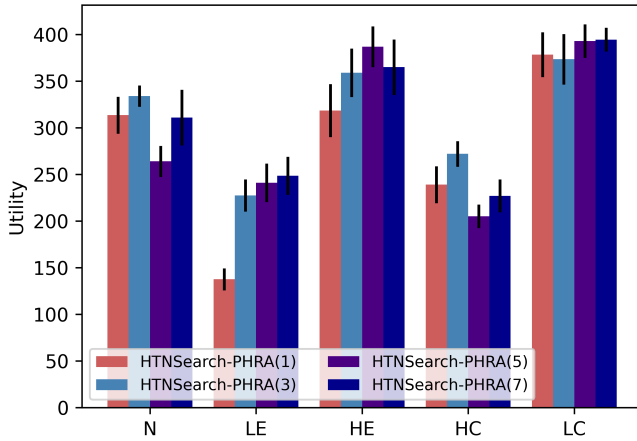


Figure 5: Utility achieved by Algorithm 1 with  $|\mathcal{H}| = 3, 5$ , and 7 respectively, on all five scenarios. Values shown are mean and standard error over 10 trials.

tively selects plans that are robust to low-energy hypothesized scenarios, is more robust to the negative conditions, producing more utility on average in both the Low Energy and High Consumption scenarios. The difference here is more significant in the Low Energy scenario as HTNSearch-PHRA can proactively plan for having less energy, and does not just react to the latest observations and state updates. However, the performance is still greater in the High Consumption scenario because it can constantly respond to the small off-nominal deviations in behavior, and though it selects plans that are robust to high energy scenarios, it does not know that such a scenario will occur until it has. In the optimistic scenarios where energy is more abundant than expected, both HTNSearch and HTNSearch-PHRA perform comparably as they are both able to make use of the excess battery supply, having achieved higher utility than in the Nominal scenario. As above, however, our algorithm performs better in the High Energy case as it proactively creates plans robust to similar situations. Finally, it is worth emphasizing that HTNSearch-PHRA also performed comparably – up to noise – to HTNSearch in the Nominal scenario where the base task network that is used by HTNSearch is correct (up to stochasticity). This means that our approach, although sensitive to off-nominal scenarios, does not sacrifice performance quality in the nominal case as well.

Overall, these results demonstrate that the proposed algorithm performs comparably or better to each baseline approach in all scenarios tested, but the benefits are most notable in scenarios where the deviations are large and sudden, rather than small and frequent, as both search-based algorithms have the ability to respond to the latter events via replanning in order to “course correct”, and in pessimistic scenarios where off-nominal behavior hurts performance rather than aids. This is because the *reactive* approach can always benefit from extra battery supply as the excess is observed, but can not always bounce back from energy deficits. However, proactively producing plans that are sen-

sitive to both these scenarios ensures that the system follows a plan that never performs too poorly in any hypothesized scenario, while also retaining the benefits of reactively course-correcting.

**Analysis of  $\mathcal{H}$  on the Quality of HTNSearch-PHRA** If we consider the results from Figure 5, we observe that increasing the size of the hypothesized scenario set can lead to improved performance, but not always and not to a large extent (up to noise). We suggest that the reason for this is that hypothetical scenarios with low likelihood – particularly in this case where none of the scenarios consider critical failures – impact the score of each generated plan to a sufficiently low degree that plans generated for low likelihood scenarios are never actually selected to be scheduled. This issue may be compounded by the fact that each hypothesized scenario alters the same parameter, namely battery supply.

The reason that HTNSearch-PHRA(5) and HTNSearch-PHRA(7) perform better than HTNSearch-PHRA(3) in the energy based scenarios compared to the consumption based scenarios is likely that the ability to constantly course correct in response to small observed deviations dominates the effect of considering low-likelihood scenarios during the planning phase. However, if we consider scenarios (N) and (HC), HTNSearch-PHRA(5) and HTNSearch-PHRA(7) actually perform worse than the other two. The reason for this is that the plans selected are too conservative – on account of being scored on low likelihood negative outcomes – and end up costing utility over the course of the full problem horizon.

In the future, we plan to analyze in greater depth whether it is possible that using a more diverse portfolio of hypothetical scenarios could lead to overall improved results, and if so, whether only scenarios of sufficient likelihood need even be included in  $\mathcal{H}$  to have a meaningful impact. As the runtime of the algorithm scales with the size of  $\mathcal{H}$ , ensuring that  $|\mathcal{H}|$  is small while still covering a sufficient set of off-nominal scenarios is important for an effective mission.

## Related Work

Onboard planning and execution are of great interest to the space domain. The *Remote Agent* was an architecture for onboard planning and execution addressing remote autonomous operation with deadlines, resource constraints, and concurrent activities (Muscettola et al. 1998). The Remote Agent flew for 48h in 1999 on the Deep Space One spacecraft using a batch planner that took hours on a RAD6000 CPU to generate a temporally flexible plan that was then used by a reactive executive controller (Pell et al. 1997) to provide robust plan execution. The planner used a refinement search paradigm (Jónsson et al. 2000) to construct a temporally flexible plan but did not consider utility in plan generation and did not perform continuous replanning due to the computational expense and long planning time (indeed the replans were scheduled in the prior plan).

The Earth Observing One (EO-1) spacecraft (Chien et al. 2005), which flew for over 12 years from 2004-2017, was designed specifically to react to dynamic scientific events.



Planning was performed by the CASPER planning software (Chien et al. 2000), which took on the order of 10s of minutes to replan but did not produce temporally flexible plans. To address this, the onboard executive (SCL) was able to flexibly interpret the *execution* of a plan to handle minor execution runtime variations. The flight and ground planners (Chien et al. 2010) both used a domain specific search algorithm that enforced a strict priority model over observations for limited model of utility. This scenario is similar to that proposed in this paper, in which the lander must react to dynamic events and observations in order to maximize its utility, while still adhering to both mission and spacecraft constraints. Recently, the Intelligent Payload Experiment (IPEX) also successfully used the CASPER planning software to achieve its mission objective, further validating the efficacy of using onboard replanning to handle dynamic events and observations during operation even when the plans are not temporally flexible (Chien et al. 2017).

The M2020 Perseverance rover also flies an onboard planner (Rabideau and Benowitz 2017) to reduce lost productivity from following fixed time conservative plans (Gaines et al. 2016). The M2020 planning architecture relies on rescheduling and flexible execution (Chi et al. 2018), ground-based compilation (Chi et al. 2019), heuristics (Chi, Chien, and Agrawal 2020), and very limited handling of planning contingencies (Agrawal et al. 2019). However, many characteristics of the M2020 mission are fundamentally different from the concept mission we consider here, such as the lack of reliable a priori model parameters, the non-repletable battery, and the long communications blackout time windows incentivizing greater mission autonomy.

Due to the presence of continuous state variables and the necessity of modeling concurrent actions, we find that stochastic planning models such as MDPs do not support our problem domain well while still being efficient to solve. Instead, as our problem has additional structure in how tasks are conditioned, we represent our model as a hierarchical task network (HTN). Hierarchical task networks have been extensively studied over the last several years as efficient models for planning in highly structured domains where expert knowledge can be embedded directly into the planner (Kuter et al. 2009; Macedo and Cardoso 2004).

Several planning algorithms have been proposed for solving HTNs (Erol, Hendler, and Nau 1994; Nau et al. 2003; Kuter et al. 2005). The subroutine that is used in Algorithm 1, HTNSearch, is most similar to SHOP2 (Nau et al. 2003). However, while SHOP2 selects task nondeterministically from the available task at each iteration of the planning loop, HTNSearch does not commit to a task but instead heuristically searches the tree of (partial) plans and deterministically selects the highest utility node found.

While the motivations and ideas of our approach are similar to the area of robust optimization and uncertainty sets, our work differs from prior work (Ben-Tal, El Ghaoui, and Nemirovski 2009; Bertsimas and Brown 2009) in two key aspects. First, robust optimization is a method for avoiding solutions to convex optimization problems that end up being infeasible in practice due to the realizations of uncertain parameters. However, as our problem is an indefinite plan-

ning problem (and replanning is not modeled as part of the planning problem), formulating it as a convex optimization problem is not straightforward. Second, uncertainty sets are often built from sampled data in the absence of well-defined priors; however, in our case, we assume the existence of well-defined priors over the uncertain parameters (e.g. battery life). We do not use these distributions during planning as full stochastic planning is intractable given the computational resources of the lander.

## Conclusion

Planning and scheduling tasks in the presence of large a priori uncertainty is a challenging problem for space-based missions. The plans need to be robust and effective while not risking compromising system safety or mission success even in the face of domain uncertainty and severe computational constraints. These issues are exacerbated in the context of the Europa Lander concept mission where there is a monotonically decreasing battery supply and large windows of communication blackouts. In this work, we have presented a deterministic planning algorithm – HTNSearch-PHRA – that functions by creating a set of hypothesized scenarios, running an efficient HTN heuristic search planner for each scenario to produce a set of plans that are then each evaluated across all hypothesized scenarios, and returning the plan that performed the best overall. We validated the approach empirically on a simulated Europa Lander domain where we compared it against existing baselines across five different stochastic mission scenarios. We demonstrated that the approach is more robust to off-nominal deviations and unexpected scenarios than the existing baselines, having consistently better performance while still being computationally efficient (running on the order of a few seconds).

There are several topics of consideration for future work. So far, we have tested HTNSearch-PHRA only in the context of varying a single scenario parameter: battery supply. The natural next step is to observe how our algorithm performs when varying other parameters such as task failures, sample target data loads, communication efficiency, task utilities, and catastrophic events. Second, we provided empirical evidence that increasing the number of hypothesized scenarios, at least when only a single parameter is varied, has diminishing returns with respect to utility but grows quickly in runtime. In the future, we would like to perform a more rigorous analysis of the conditions under which increasing the set of hypothesized scenarios will be beneficial, or identify if there are conditions under which new hypothesized scenarios will not impact the results of the algorithm. Finally, we would like to perform more empirical evaluations of our algorithm on a wider set of mission scenarios where the system can perform online model updates as it makes observations that enable it to update its model priors.

## Acknowledgments

This research was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration (80NM0018D0004).



## References

- Agrawal, J.; Chi, W.; Chien, S.; Rabideau, G.; Kuhn, S.; and Gaines, D. 2019. Enabling Limited Resource-Bounded Disjunction in Scheduling. In *IWPSS*.
- Ben-Tal, A.; El Ghaoui, L.; and Nemirovski, A. 2009. *Robust Optimization*. Princeton University Press.
- Bertsimas, D.; and Brown, D. B. 2009. Constructing Uncertainty Sets for Robust Linear Optimization. *Operations Research*.
- Bonet, B.; and Geffner, H. 2001. Planning as heuristic search. *Artificial Intelligence*.
- Chi, W.; Agrawal, J.; Chien, S.; Fosse, E.; and Guduri, U. 2019. Optimizing Parameters for Uncertain Execution and Rescheduling Robustness. In *ICAPS*.
- Chi, W.; Chien, S.; and Agrawal, J. 2020. Scheduling with Complex Consumptive Resources for a Planetary Rover. In *ICAPS*.
- Chi, W.; Chien, S.; Agrawal, J.; Rabideau, G.; Benowitz, E.; Gaines, D.; Fosse, E.; Kuhn, S.; and Biehl, J. 2018. Embedding a Scheduler in Execution for a Planetary Rover. In *ICAPS*.
- Chien, S.; Doubleday, J.; Thompson, D. R.; Wagstaff, K. L.; Bellardo, J.; Francis, C.; Baumgarten, E.; Williams, A.; Yee, E.; Stanton, E.; et al. 2017. Onboard autonomy on the intelligent payload experiment cubesat mission. *Journal of Aerospace Information Systems*.
- Chien, S.; Sherwood, R.; Tran, D.; Cichy, B.; Rabideau, G.; Castano, R.; Davis, A.; Mandl, D.; Frye, S.; Trout, B.; et al. 2005. Using autonomy flight software to improve science return on Earth Observing One. *Journal of Aerospace Computing, Information, and Communication*.
- Chien, S.; Tran, D.; Rabideau, G.; Schaffer, S.; Mandl, D.; and Frye, S. 2010. Timeline-based space operations scheduling with external constraints. In *ICAPS*.
- Chien, S. A.; Knight, R.; Stechert, A.; Sherwood, R.; and Rabideau, G. 2000. Using Iterative Repair to Improve the Responsiveness of Planning and Scheduling. In *ICAIPS*.
- Chong, E. K.; Givan, R. L.; and Chang, H. S. 2000. A framework for simulation-based network control via hindsight optimization. In *CDC*. IEEE.
- Erol, K.; Hendler, J. A.; and Nau, D. S. 1994. UMCP: A Sound and complete procedure for hierarchical task-network planning. In *ICAIPS*.
- Gaines, D.; Anderson, R.; Doran, G.; Huffman, W.; Justice, H.; Mackey, R.; Rabideau, G.; Vasavada, A.; Verma, V.; Estlin, T.; et al. 2016. Productivity challenges for Mars rover operations. In *ICAPS Workshop on Planning and Robotics*. London, UK.
- Goldman, R. P.; and Kuter, U. 2019. Hierarchical Task Network Planning in Common Lisp: the case of SHOP3. In *European Lisp Symposium*. Zenodo.
- Hand, K. P. 2017. *Report of the Europa Lander science definition team*. National Aeronautics and Space Administration.
- Hansen, E. A.; and Zilberstein, S. 2001. LAO: A heuristic search algorithm that finds solutions with loops. *Artificial Intelligence*.
- Hoffmann, J.; and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *JAIR*.
- Jónsson, A. K.; Morris, P. H.; Muscettola, N.; Rajan, K.; and Smith, B. D. 2000. Planning in Interplanetary Space. In *ICAIPS*.
- Korf, R. E. 1990. Real-time heuristic search. *Artificial intelligence*.
- Kuter, U.; Nau, D.; Pistore, M.; and Traverso, P. 2009. Task decomposition on abstract states, for planning under nondeterminism. *Artificial Intelligence*.
- Kuter, U.; Nau, D. S.; Pistore, M.; and Traverso, P. 2005. A Hierarchical Task-Network Planner based on Symbolic Model Checking. In *ICAPS*.
- Macedo, L.; and Cardoso, A. 2004. Case-based, decision-theoretic, HTN planning. In *ECCBR*. Springer.
- Muscettola, N.; Nayak, P. P.; Pell, B.; and Williams, B. C. 1998. Remote agent: To boldly go where no AI system has gone before. *Artificial intelligence*.
- Nau, D. S.; Au, T. C.; Ilghami, O.; Kuter, U.; Murdock, J. W.; Wu, D.; and Yaman, F. 2003. SHOP2: An HTN Planning System. *JAIR*.
- Pell, B.; Gat, E.; Keesing, R.; Muscettola, N.; and Smith, B. 1997. Robust periodic planning and execution for autonomous spacecraft. In *IJCAI*.
- Pineda, L. E.; and Zilberstein, S. 2014. Planning under uncertainty using reduced models: Revisiting determinization. In *ICAPS*.
- Rabideau, G.; and Benowitz, E. 2017. Prototyping an On-board Scheduler for the Mars 2020 Rover. In *IWPSS*.
- Verma, V.; Gaines, D.; Rabideau, G.; Schaffer, S.; and Joshi, R. 2017. Autonomous Science Restart for the Planned Europa Mission with Lightweight Planning and Execution. In *IWPSS*.
- Wang, D.; Russino, J. A.; Basich, C.; and Chien, S. 2020. Using Flexible Execution, Replanning, and Model Parameter Updates to Address Environmental Uncertainty for a Planetary Lander. In *I-SAIRAS*.
- Yoon, S. W.; Fern, A.; and Givan, R. 2007. FF-Replan: A baseline for probabilistic planning. In *ICAPS*.
- Yoon, S. W.; Fern, A.; Givan, R.; and Kambhampati, S. 2008. Probabilistic Planning via Determinization in Hindsight. In *AAAI*.
- Zilberstein, S. 1996. Using anytime algorithms in intelligent systems. *AI magazine*.

# Real-time Planning and Execution for Industrial Operations

Filip Dvorak

Schlumberger  
555 Industrial Blvd  
Sugar Land, TX 77478, United States  
fdvorak@slb.com

## Abstract

Real-time planning systems in industrial environments are faced with challenges of reasoning about how to find the optimal plan of actions to achieve a goal, dispatch and monitor the execution of actions, act on state information, diagnose failures and learn to adapt to unexpected events. In this paper, we formalize planning and execution architecture that encapsulates hierarchical deliberative planning, reactive planning and failure management within behavior trees, discussing experience from practical deployments of the architecture in the oil and gas industry.

We are living in times of continuous evolution of automation, where a broad range of systems is becoming directly connected to the internet, operational context of simple devices grows wider and complexity of autonomous behaviors keeps increasing. The automation challenges consist of having all actors in the system to **connect**, providing their real-time operational data, which we need to **collect**, and **reason** how to **control** the system to achieve the goals efficiently and reliably. Leaving aside the challenges of real-time connectivity and data collection, reasoning about complex dynamic systems requires capturing tacit and explicit knowledge often spread across multitude of sources including different forms of documentation, historical operational data, human domain experts and models tailored for certain configurations and subsystems.

This paper addresses the challenge of heterogeneously spread domain knowledge in real-world problems with a hierarchical autonomous control architecture, combining well-known techniques from satisfiability theory, automated planning, automated diagnosis, constraint programming and behavior trees into a single formal model. In the following sections we first build up representation and the necessary reasoning techniques for planning and diagnosis. Then we build an execution model on top of the behavior trees, which will encapsulate the reasoning techniques into tree nodes, and define semantics of composing execution models by attaching behavior trees to each other. Finally, we discuss the experience from practical deployments of the architecture in the field.

## Representation

The representation of the world state consists of a set of first-order variables, upon which we build quantified boolean for-

mulas.

**Definition 1.** For a finite set of objects  $C$ , a finite set of types  $T = \{T_i \subseteq C\}$  and a set of  $m$   $n$ -ary variables  $V = \{v(c_1 \in T_1, \dots, c_n \in T_n) \mid \text{dom}(v) = \{1, 0\} \vee \text{dom}(v) = \mathbb{Z} \vee \text{dom}(v) = \mathbb{R}\}$ ,  $\alpha_V = (u_1, \dots, u_m)$  denotes an assignment where  $u_i \in \text{dom}(v_i)$ ,  $s_{\alpha_V}$  denotes the state of the world and  $S_V$  denotes the state space as a set of states produced by all possible assignments to variables  $V$ .

Given state representation enables capturing propositional variables together with numeric variables and while it leads to infinite state space, we are interested in practical (finite) sizes of problems, and we will discretize the numeric variables with numeric comparison operators. The following definition will enable us to capture structures in the states space.

**Definition 2.** For a state  $s_{\alpha_V}$ ,  $F$  is a formula grammar  $F \rightarrow$

1.  $\forall x(F) \mid \exists x(F)$ , where  $x \in T_i$
2.  $(F) \odot (F)$ , where  $\odot \in \{\vee, \wedge, \Rightarrow, \Leftarrow\}$
3.  $v_i \odot c$ , where  $v_i \in V, c \in \mathbb{R}, \odot \in \{<, >, =\}$
4.  $v_i \mid \neg v_i$ , where  $v_i \in V, \text{dom}(v_i) = \{0, 1\}$

For a formula  $f$  accepted by grammar  $F$  and a state  $s_{\alpha_V}$ ,  $f(s_{\alpha_V})$  and  $f(s)$  denote a formula whose truth value is its evaluation.

With regard to evaluation, formulas can be treated as a quantified boolean formula (QBF) since line 3. of the grammar collapses all numeric variables into boolean variables, becoming equivalent to line 4. QBF can be evaluated using a simple recursive algorithm and while it is PSPACE-hard in general, we restrict ourselves to easy instances representing pieces of real-world knowledge, e.g.  $\forall d \in \text{devices} (\text{noise}(d) < 60\text{dB})$ .

## Diagnosis

Real-world systems naturally include scenarios when an unexpected event leads to a failure that requires either automated or human-assisted recovery. A common approach to model the consistency of a system is to define a set of constraints that encode the atomic pieces of knowledge about the system - rules that need to be satisfied. Verifying that all rules are satisfied, e.g. in CP or SAT, then tells us that the system is consistent, however, an inconsistent system can

manifest an exponential number of conflicts causing the failure. We can gain more understanding of a failure by introducing *minimal diagnosis* and *minimal conflict set*.

**Definition 3.** For a state  $S$  and a set of formulas  $A = \{f_1, \dots, f_n\}$ , we define:

- A conflict set  $C \subseteq A$ , where  $C_S$  is inconsistent, and minimal conflict set  $C_S^{min}$ , where  $\neg \exists x \in C_S^{min} : C_S^{min} \setminus x$  is inconsistent.
- A diagnosis  $D \subseteq A$ , where  $A_S \setminus D_S$  is consistent, and minimal diagnosis  $D_S^{min}$ , where  $\neg \exists \text{ diagnosis } D_s \subseteq A : |D_s| < |D_S^{min}|$ .

We can compute the minimal conflict set using the QuickXPlain algorithm (Junker 2004) requiring  $O(|C_S^{min}| \log(|S| |C_S^{min}|))$  consistency checks and the minimal diagnosis requiring  $O(|D_S^{min}| \log(|S| |D_S^{min}|))$  consistency checks. While the consistency check is PSPACE-hard (formula is QBF), we can expand the formulas into propositional formulas (exponential growth in number of quantifiers), and consistency checking then becomes NP-complete (constraint satisfaction problem). The main advantage of expanding the formulas is to capture finer diagnosis and conflict sets, e.g. invalidated  $\forall o \in \text{operations } \text{ready}(o) \rightarrow \text{prepared}(o)$  after expansions tells the exact operation causing the inconsistency.

## Planning

Definitions above gave us the state of the world, formulas to find structures in the world, capability to identify if the world is inconsistent and how to explain the inconsistency. Now we can start interacting with the world using a deliberative process of choosing and organizing actions by their expected outcomes - planning (Nau, Ghallab, and Traverso 2004).

**Definition 4.** For a state space  $S$  we define a temporal numeric planning problem  $P = (S, O, s_i, s_g)$ , where:

- $O$  is a set of operators  $o = (f_c, f_p, \alpha_e, d, p)$ , where  $f_c$  and  $f_p$  are formulas representing preconditions and persistent conditions,  $\alpha_e$  represents effects as a partial parameterized assignment,  $d : S \rightarrow \mathbb{R}$  is a parameterized duration function and  $p$  is a set of type assignments to all free variables (parameters) in  $f_c, f_p, \alpha_e$  and  $d$ .
- $s_i$  is an initial state of the world.
- $s_g$  is a partially assigned goal state.

For a planning problem  $P$ ,  $\pi = \{a_o(b, p_\alpha)\}$  is a set of actions, where  $o \in O$  is an operator,  $b \in \mathbb{R}$  is the action start and  $p_\alpha$  is an assignment of objects to free variables in  $f_c, f_p, \alpha_e$  and  $d$ .

$\pi$  is a plan for the planning problem  $P$  iff all the actions can be started at their start times running for their duration, all conditions and persistent conditions are satisfied, all action effects are applied and after applying effects of the latest actions, and we reach a state that satisfies the goal  $s_g$ .

For a planning problem  $P$ ,  $\Pi$  is the set of all possible plans, and we say that  $\pi^* \in \Pi$  is makespan-optimal iff there does not exist a plan with a shorter duration between the earliest action start and latest action end.

While classical propositional planning is PSPACE-hard (Nau, Ghallab, and Traverso 2004), allowing temporal concurrent actions shifts the difficulty of finding a plan to EXPSPACE-hardness (Rintanen 2007) and allowing numeric effects makes the problem undecidable (Helmert 2002).

## Execution Model

In this section, we focus on defining the building blocks of a hierarchical execution model using the behavior trees and within these blocks, we unify reactive and deliberative planning (Ghallab, Nau, and Traverso 2016).

Since we integrate multiple sources of knowledge and models together, the capability to reflectively reason about the execution and structure of the model itself becomes practical, and we capture it in *reflective predicates* following on Definition 1.

**Definition 5.** For a set of objects  $C$ , types  $T$ , variables  $V$  and a planning problem  $P = (S, O, s_i, s_g)$ , we define predicates:

- $b, e : O \times \mathbb{N} \times \mathbb{R} \times T_1 \times \dots \times T_n \rightarrow \{0, 1\}$ , where  $b(o, i, t, c_1, \dots, c_n)$  represents the start of execution of action specified by operator  $o(c_1, \dots, c_n)$  at time  $t$ , and  $e(o, i, t, c_1, \dots, c_n)$  represents the end of execution of the action, and  $i$  represents a unique identifier of the action instance.
- $a : \mathbb{N} \times \mathbb{N} \times T_j \rightarrow \{0, 1\}$ , where  $a(v, n, c)$  represents an assignment of value  $c$  to free variable  $v$  for a unique instance of execution of behavior tree node  $n$ .
- $x : \mathbb{N} \times \mathbb{N} \times \mathbb{R} \rightarrow 0, 1$ , where  $x(D_s^{min}, C_s^{min}, t)$  representing explanation of a diagnosed failure at time  $t$ .

Reflective predicates allow recording an actual schedule of actions executions, passing of assignments to free variables between nodes of the behavior tree, and explanations of diagnosed failures, assuming we have a mapping between variable names and  $\mathbb{N}$  and formulas and  $\mathbb{N}$ .

## Behavior Trees

A behavior tree is a directed acyclic graph with a single root node and a single component. The execution of a behavior tree is execution context passing concept, similar to finite state machines, which naturally supports hierarchical compositions of trees.

The execution of a behavior tree starts from the root which sends tick signal that passes execution context to a child. When the execution context is passed to a node, it returns to the parent status *running* if its execution has not finished yet, *success* if it has achieved its goal, or *failure* otherwise. When the execution context is passed to a node for the first time, it will *initialize* and once it returns success of failure it will *reset*. When a parent node passes the context to a child for the first time, it updates the current state  $s$  with assignments  $a(v, n, c)$  for all variable assignments it has done and received with the identifier  $n$  of the child node. We further assume that all the nodes have access to the current state of the world  $s$ . We distinguish the following control nodes:

- Selector node executes the first child that does not fail. It return success or running when one of its children returns success or running and returns failure if all the children failed.
- Sequence node executes the first child that has not yet succeeded. A sequence returns failure or running when one of its children returns failure or running and returns success if all of children succeeded.
- Flow node executes all its children in a sequence (even if they previously succeeded) until all return success. It returns the status of a first child in the sequence that does not return success.
- Parallel node executes all its children and returns failure if at least one child has failed. It returns running if at least one child has returned running and none has returned failure. It returns success if all children have returned success.
- Branch node is parameterized by formula  $f$  representing the branching schema. When initialized, it finds all possible instantiations  $I$  of the free variables in  $f$  such that  $f_s$  is true for the current state  $s$  and then it duplicates each of its children for each instantiation  $i \in I$ , assigning free variables  $a$  in the created nodes according to  $i$ . The branch node then behaves exactly the same as the Parallel node and when reset, it removes all duplicate child nodes it has created during initialization.

The children of selector and sequence nodes are totally ordered, which represents the priority of alternatives in selector node and progression in sequence nodes. We assume that ordering of children in parallel node does not matter. Branch node can be seen as a multi-threading pool that lunches independently in parallel its subtrees for a given formula, e.g.  $o \in O : ready(o) \wedge \neg done(o)$  will duplicate and execute its subtrees for each operation that is ready but not done yet.

We assume that control nodes do not appear at the leaves of the tree and we define four parameterized leaf nodes:

- Infer node is parameterized by a formula  $f$ . It returns success if formula  $f_s$  is true and failure otherwise. It never returns running.
- Execute is parameterized by formulas  $f^c$ ,  $f^p$ ,  $f^e$  and an operator  $o$ . The node returns failure if  $f_s^c$  does not hold when initialized or there is a time when  $f_s^p$  does not hold. When initialized, the node updates  $s$  with predicate  $b(o, i, t, c_1, \dots, c_n)$  and it returns success if  $f_s^e$  holds, otherwise it returns running.
- Diagnose is parameterized by a set of formulas  $A$ , it return running until  $D_s^{min}$  and  $C_s^{min}$  are found. It returns success if  $D_s^{min} = \emptyset$ , otherwise it sets  $x(D_s^{min}, C_s^{min}, t)$  and returns failure.
- Plan is parameterized by a temporal *PDDL domain* (Fox and Long 2003) with numeric-fluents  $D$  and *PDDL problem template*  $P$ , where  $P$  is instantiated by current state  $s$  and its free variables are assigned by predicate  $a$  for this node, denoted as  $P_s^i$ . When initialized, the node creates a planning problem  $(D, P_s^i) = (S, O, s_i, s_g)$  and runs a planner. It returns running while the planner is running

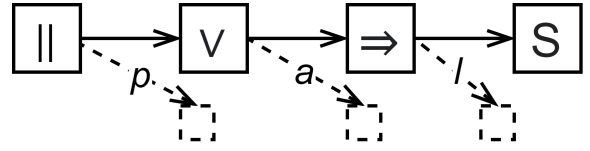


Figure 1: Depicts a behavior tree structure of a parallel node followed by selector and flow nodes with insertion points  $p$ ,  $a$  and  $l$

and returns failure if the planner does not find a plan or runs out of time without finding a single plan. Once a plan is found, the node starts to dispatch action  $a_o(t, p_\alpha)$  by setting predicate  $b(o, i, t, c_1, \dots, c_n)$  if  $f_s^c$  is satisfied and all preceding actions in the plan have finished dispatching. It finishes dispatching the action by setting the predicate  $e(o, i, t, c_1, \dots, c_n)$  if  $f_s^e$ . The node returns success if all planned actions finished dispatching and it returns failure if the persistent conditions  $f_s^p$  did not hold for dispatching of an action.

Infer node is performing a consistency check of (typically) simple formula, while diagnose node builds on infer node and it can work with  $10^5$  simple formulas after expansion providing the explanation of inconsistency. Diagnose node is typically positioned just before the plan node to validate that the whole system is consistent and the planner is expected to find a plan or it can run in parallel with execution and keep checking consistency of every state encountered. Execute node can be seen as a dispatching of a single parameterized operator, while plan node finds and dispatches hundreds of actions according to their partial ordering.

In the further text execution model  $\mathcal{M}$  represents a behavior tree build from leaf and control nodes such that control nodes have at least one child and leaf nodes have none.

## Architecture

Encapsulating reasoning and action execution into nodes of a behavior tree provides the grounds to define a global automated planning and execution system in terms of tree composition.

## Composition

We assume that every execution model  $\mathcal{M}$  has the structure depicted in Figure 1. For models  $\mathcal{M}'$  and  $\mathcal{M}$  we define compositions:

- $\mathcal{M}' \xrightarrow{p} \mathcal{M}$  attaches the root node of  $\mathcal{M}'$  as child of root node  $||$  in  $\mathcal{M}$ .
- $\mathcal{M}' \xrightarrow{a} \mathcal{M}$  attaches the root node of  $\mathcal{M}'$  as a first child of selector node  $\vee$  in  $\mathcal{M}$ , which allows to completely alternate execution of subtree  $S$ .
- $\mathcal{M}' \xrightarrow{l} \mathcal{M}$  attaches the root node of  $\mathcal{M}'$  as a first child of the flow node  $\Rightarrow$  in  $\mathcal{M}$ , which enforce the execution of  $\mathcal{M}'$  to precede the execution of subtree  $S$ .

We can observe that the product of all execution model compositions is an execution model and the composition tree is a deterministic well-defined structure.

## Orchestration

Orchestration is a process of discovering the *execution domains* of its *deployment*, composing them into a single execution model and executing the model. We define a deployment as a set of domains  $\mathcal{D} = \{(R, \mathcal{M}, C)\}$ , where  $R$  is the runtime sensory processing function that interprets raw observations of the world into the state space of variables  $S_V$ ,  $\mathcal{M}$  is an execution model and  $C$  is a set of compositions  $\{\mathcal{M} \odot \mathcal{M}' | (R', \mathcal{M}', C') \in \mathcal{D}, \odot \in \{\overset{p}{\rightarrow}, \overset{a}{\rightarrow}, \overset{1}{\rightarrow}\}\}$ .

We say the deployment  $\mathcal{D}$  is valid iff union of runtimes does not have a conflict (assigning different values to the same variable) and the compositions defined by the domains form a composition tree. Then orchestration of a valid deployment  $\mathcal{D}$  provides autonomous control of the world interpreted by the union of its runtimes and modeled by the execution model composed from execution models of its domains.

## Application

For the last two years and as a continuous effort we have been extending the coverage of automation by adding new domains, extending execution models for heavy machinery, and running end-to-end automation deployments in the field. In this paper, we have formalized the high-level approach for modeling and executing large autonomous systems, yet the practical deployments had to be enabled by the work of dozens of embedded software engineers hidden in lower-level automation, e.g. deep learning classifiers for vibrations, sound, and visual input, which is beyond the scope of this paper.

A typical deployment has one top-level PDDL model with externally provided goals, a proprietary temporal PDDL2.1 planner based on POPF (Coles et al. 2010) then finds close-to-optimal plan with hundreds of actions spanning over several days with a 10s timeout, which upon human review starts to execute (we can imagine a *review* action whose completion is tied to clicking a confirmation button in the UI). Some actions of the top-level plan can be further decomposed upon its start of execution into planning subproblems, where the execution model has been provided by a different domain. This hierarchical decomposition is constructed by attaching subproblem execution models with  $\overset{p}{\rightarrow}$ , conditioned by an infer node that checks the actions start through predicates  $b$  and  $e$ . Reactive and deliberative execution models are interchangeable with regard to execution, where reactive models tend to be faster for development but they do not provide a plan in advance, and the hierarchical decomposition during the execution can then be formed from any combination of reactive and deliberative decompositions. Building up a prediction of a fully decomposed plan is a topic for future work.

Execution of real-world deployments naturally manifests many failures, which we divide into categories *known*, *diagnosable*, and *unexpected*. Known failures are typically tied

to a variable in the state-space and there is an execution model to recover from them. Diagnosable failures are caught by the automated diagnosis (typically hundreds of atomic formulas) and they are addressed either from a transfer to known failures, or they are fixed in the responsible domains. Unexpected failures lead to a failure of the execution model node, potentially leading to a failure of the execution of the whole orchestrated execution tree. In practice, addressing failures of the system is an iterative process of adding new diagnostic formulas, and recovery models for known failures and eliminating the long tail of unexpected failures. While we have initially explicitly modeled individual failures in the top-level PDDL, the recovery modeling and modeling of combinations of failures impacted the maintainability of PDDL (additional thousands of lines in the PDDL domain), and we have shifted to model failures in a different execution model. For this purpose, composition  $\overset{1}{\rightarrow}$  is particularly useful, attaching reactive execution models to resolve failures, e.g. execution model starting with infer node conditioned with *motor-failure* can contain the necessary instructions for recovering from failed motor.

## Related Work

Deliberative and reactive planning has been interleaved in systems such as autonomous underwater vehicles T-Rex (McGann et al. 2008) and LAAS architecture for autonomous satellite and rover control (Ghallab et al. 2001). Task planning based on PDDL2.1 has also been integrated into a popular robotic framework as ROSPlan (Cashmore et al. 2015) - compared to which our architecture provides hierarchical reasoning, where different branches of the hierarchy can be implemented independently and composed together during orchestration.

Failure management systems are a common part of autonomous control systems and our current implementation is simpler by looking only into the current state and history of executed actions compared with failure management such as HyDe (Frank 2019), which diagnoses the whole evolution of state variables. Behavior trees are commonly used to capture reactive behaviors and they can also be constructed to capture deliberative behavior (Colledanchise and Ögren 2017) - The Plan node could be equivalently implemented by finding a plan and generating a behavior tree representing the plan.

Manually building execution models, be it reactive or deliberative modeling, has been a critical and demanding part of the automation development. While some approaches learn directly from observed action sequences (action traces) (Cresswell and Gregory 2011) or snapshots of the state of the world (state traces) (Bonet and Geffner 2019), learning on top of an imperfect model has been explored (Zhuo, Kambhampati, and Nguyen 2012) and recently deep reinforcement learning has shown capabilities to learn without a model (François-Lavet et al. 2018).

## Conclusions

This paper formalizes a hierarchical scalable autonomous control architecture for real-time problems with heteroge-

neously spread domain knowledge, practically applied in an increasing number of unique field deployments in the oil and gas industry. The architecture interleaves well-known reasoning techniques, discussing the intricacies of working within hard (undecidable) problem spaces, and provides support for independent parallel development of different domains (PDDL modeling, reactive planning, or other custom types of behaviors).

## References

- Bolton, W. 2015. Preface. In Bolton, W., ed., *Programmable Logic Controllers (Sixth Edition)*, ix – xii. Boston: Newnes, sixth edition edition. ISBN 978-0-12-802929-9. doi:<https://doi.org/10.1016/B978-0-12-802929-9.09990-8>. URL <http://www.sciencedirect.com/science/article/pii/B9780128029299099908>.
- Bonet, B.; and Geffner, H. 2019. Learning First-Order Symbolic Planning Representations from Plain Graphs. *CoRR* abs/1909.05546. URL <http://arxiv.org/abs/1909.05546>.
- Cashmore, M.; Fox, M.; Long, D.; Magazzeni, D.; Ridder, B.; Carrera, A.; Palomeras, N.; Hurtós, N.; and Carreras, M. 2015. Rosplan: Planning in the robot operating system. *Proceedings International Conference on Automated Planning and Scheduling, ICAPS 2015*: 333–341.
- Coles, A.; Coles, A.; Fox, M.; and Long, D. 2010. Forward-Chaining Partial-Order Planning. 42–49.
- Colledanchise, M.; and Ögren, P. 2017. Behavior Trees in Robotics and AI: An Introduction. *CoRR* abs/1709.00084. URL <http://arxiv.org/abs/1709.00084>.
- Cresswell, S.; and Gregory, P. 2011. Generalised Domain Model Acquisition from Action Traces.
- Fox, M.; and Long, D. 2003. PDDL2.1: An extension to PDDL for expressing temporal planning domains. *J. Artif. Intell. Res. (JAIR)* 20: 61–124. doi:10.1613/jair.1129.
- Frank, J. 2019. Artificial Intelligence: Powering Human Exploration of the Moon and Mars.
- François-Lavet, V.; Henderson, P.; Islam, R.; Bellemare, M. G.; and Pineau, J. 2018. An Introduction to Deep Reinforcement Learning. *Foundations and Trends® in Machine Learning* 11(3-4): 219–354. ISSN 1935-8245. doi:10.1561/22000000071. URL <http://dx.doi.org/10.1561/22000000071>.
- Ghallab, M.; Ingrand, F.; Solange, L.-C.; and Py, F. 2001. Architecture and Tools for Autonomy in Space.
- Ghallab, M.; Nau, D.; and Traverso, P. 2016. *Automated Planning and Acting*. USA: Cambridge University Press, 1st edition. ISBN 1107037271.
- Guarnieri, M. 2010. The Roots of Automation Before Mechatronics. *Industrial Electronics Magazine, IEEE* 4: 42 – 43. doi:10.1109/MIE.2010.936772.
- Helmert, M. 2002. Decidability and Undecidability Results for Planning with Numerical State Variables. In *Proceedings of the Sixth International Conference on Artificial Intelligence Planning Systems, AIPS’02*, 44–53. AAAI Press. ISBN 1577351428.
- Junker, U. 2004. QUICKXPLAIN: Preferred Explanations and Relaxations for over-Constrained Problems. In *Proceedings of the 19th National Conference on Artificial Intelligence, AAAI’04*, 167–172. AAAI Press. ISBN 0262511835.
- McGann, C.; Py, F.; Rajan, K.; Thomas, H.; Henthorn, R.; and McEwen, R. 2008. T-REX: A model-based architecture for AUV control .
- Nau, D.; Ghallab, M.; and Traverso, P. 2004. *Automated Planning: Theory & Practice*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc. ISBN 1558608567.
- Rintanen, J. 2007. Complexity of Concurrent Temporal Planning. 280–287.
- Warndorf, P. 2011. Resources. URL <https://www.mtconnect.org/resources>.
- Zhuo, H. H.; Kambhampati, S.; and Nguyen, T. 2012. Model-Lite Case-Based Planning.



# Multi-Objective Path-Based D\* Lite

Zhongqiang Ren<sup>1</sup>, Sivakumar Rathinam<sup>2</sup>, Howie Choset<sup>1</sup>

<sup>1</sup> Robotics Institute, Carnegie Mellon University, 5000 Forbes Ave., Pittsburgh, PA 15213, USA.

<sup>2</sup> Department of Mechanical Engineering, Texas A&M University, College Station, TX 77843-3123.  
zhongqir@andrew.cmu.edu, choset@andrew.cmu.edu, srathinam@tamu.edu

## Abstract

Incremental graph search algorithms, such as D\* Lite, reuse previous search efforts to speed up subsequent similar path planning tasks. These algorithms have demonstrated their efficiency in comparison with search from scratch, and have been leveraged in many applications such as navigation in unknown terrain. On the other hand, path planning typically involves optimizing multiple conflicting objectives simultaneously, such as travel risk, arrival time, etc. Multi-objective path planning is challenging as the number of “Pareto-optimal” solutions can grow exponentially with respect to the size of the graph, which makes it computationally burdensome to plan from scratch each time when similar planning tasks needs to be solved. This article presents a new multi-objective incremental search algorithm called Multi-Objective Path-Based D\* Lite (MOPBD\*) which reuses previous search efforts to speed up subsequent planning tasks while optimizing multiple objectives. Numerical results show that MOPBD\* is more efficient than search from scratch and runs an order of magnitude faster than existing incremental method for multi-objective path planning.

## Introduction

Given a graph with non-negative scalar edge costs, the shortest path problem (SPP) (Hart, Nilsson, and Raphael 1968) aims to compute a minimum-cost path connecting the given start and destination nodes in the graph. Incremental search algorithms, such as lifelong planning A\* (Koenig, Likhachev, and Furcy 2004), D\* Lite (Koenig and Likhachev 2002) etc., generalize these computations to a dynamic setting that allows for cost changes in the edges of the graph. When edge costs change, these algorithms reuse previous search efforts to speed up similar planning tasks which makes them very efficient in comparison to planning paths from scratch. Such efficiency makes incremental search a popular technique in many robotic applications such as navigation in unknown terrain (Koenig and Likhachev 2005; Urmson et al. 2008).

In many real-world applications, such as hazardous material transportation (Bronfman et al. 2015), UAV path planning for search and rescue (Hayat et al. 2017), robot routing in urban waterways (Shan et al. 2020), the path planning problem may involve optimizing multiple (conflicting) objectives such as minimizing travel risk, fuel usage, arrival time, to name a few. It may not be possible to convert

these objectives into a single, weighted objective because the choice of weights are difficult to obtain (Rojers et al. 2013). This leads us to addressing the multi-objective shortest path problem (MO-SPP) (Loui 1983). MO-SPP generalizes the conventional SPP by associating each edge with a cost vector (of constant length) where each component of the vector corresponds to an objective to be minimized.

In the presence of multiple conflicting objectives, there is no single solution path that optimizes all objectives in general. Therefore, the goal of MO-SPP is to find a Pareto-optimal set (of solution paths), whose cost vectors form the so-called Pareto-optimal front. A path is Pareto-optimal (or non-dominated) if no objective can be improved without deteriorating at least one of the other objectives. MO-SPP is NP-hard, even with two objectives (Hansen 1980; Ehr Gott 2005), as the size of the Pareto-optimal front can grow exponentially with respect to the number of nodes in the graph. To solve MO-SPP, there are several A\*-like multi-objective planners (Stewart and White 1991; Mandow and De La Cruz 2008; Ulloa et al. 2020; Goldin and Salzman 2021) which compute the exact or an approximated Pareto-optimal front.

In this work, we consider the dynamic version of the MO-SPP where the costs of the edges can change. After an event when some edge costs change, we aim to develop an incremental search algorithm that reuses previous search efforts to speed up similar planning tasks. Incremental search is important as a naive approach that computes Pareto-optimal front from scratch after every event can be computationally expensive for MO-SPP. To our knowledge, the only existing work that considers a similar problem is MOD\* (Oral and Polat 2015), which combines D\* Lite and MOA\* (Stewart and White 1991) to reuse previous search efforts. However, MOA\* has been shown (Mandow and De La Cruz 2008) to be inefficient due to its “node-based” expansion strategy during the search and is outperformed by NAMOA\* (Mandow and De La Cruz 2008) which employs a “path-based” expansion strategy.<sup>3</sup>

This work aims to leverage both D\* Lite and the path-

<sup>3</sup>In MO-SPP, there are multiple Pareto-optimal partial solution paths from the start to some node. The node-based expansion strategy selects nodes for expansion and extends (or re-extends) all partial solution paths at that node. The path-based expansion strategy selects a partial solution path for expansion and extends only the selected path. More detail can be found in the Preliminary section.

based expansion in NAMOA\* to create a novel incremental multi-objective algorithm to plan paths in a dynamic environment. Fusing D\* Lite and NAMOA\* is challenging. D\* Lite defines local consistency between adjacent nodes and keeps expanding nodes that are locally inconsistent until an optimal path is found. It's non-trivial to fuse local consistency between *nodes* with a *path*-based expansion strategy in a harmonic way.

To achieve this goal, we propose a new type of local consistency in multi-objective settings that is suitable for path-based methods. With that in hand, we develop a new algorithm named Multi-Objective Path-Based D\* Lite (MOPBD\*). We analyze and show that MOPBD\* is able to compute all Pareto-optimal solutions in a dynamic graph. Next, we verify the proposed algorithm with extensive numerical simulations in several dynamic graphs with two objectives, and demonstrate its efficiency by comparing with running NAMOA\* from scratch and the existing node-based incremental method (MOD\*). Our results show that MOPBD\* is more efficient than search from scratch (NAMOA\*) and runs an order of magnitude faster than MOD\*.

## Related Work

Applications such as navigation in unknown terrain requires the planner to solve a series of similar planning tasks efficiently. To achieve this goal, rather than naively searching from scratch each time, incremental search algorithms such as D\* (Stentz 1995), LPA\* (Koenig, Likhachev, and Furcy 2004), D\* Lite (Koenig and Likhachev 2002) reuse previous search efforts, namely a search tree that stores the identified partial solution paths, to speed up the current search without losing optimality guarantees. D\* Lite has been improved and extended in many ways (Likhachev et al. 2005; Aine and Likhachev 2013). However, all those algorithms optimize a single-objective: minimizing the sum of edge cost values along the planned path.

On the other hand, multi-objective shortest path problem (MO-SPP) extends the shortest path problem by associating a cost vector to each edge and aims to find all paths with Pareto-optimal cost vectors. Developing efficient algorithms for MO-SPP has a long history (Loui 1983) and remains an active research topic (Stewart and White 1991; Mandow and De La Cruz 2008; Ulloa et al. 2020; Goldin and Salzman 2021). To solve the problem, seminal works MOA\* (Stewart and White 1991) and NAMOA\* (Mandow and De La Cruz 2008) both extend A\* to handle multiple objectives but with different strategies. MOA\* uses a node-based selection and expansion strategy while NAMOA\* adopts a path-based one and outperforms MOA\* in general (Mandow and De La Cruz 2008).

This work focuses on multi-objective incremental search algorithms, which reuse previous search efforts to speed up the current multi-objective search. To our limited knowledge, the only known work that considers a similar problem is MOD\* (Oral and Polat 2015), which is an incremental node-based multi-objective search algorithm combining both MOA\* and D\* Lite. However, we learn from (Oral and Polat 2015) that NAMOA\* is not used as a baseline to verify

MOD\* and the property of MOD\* is not shown. Based on the existing analysis of MOA\* and NAMOA\* (Mandow and De La Cruz 2008), we take the view that an incremental algorithm with path-based expansion can possibly improve the computational efficiency over node-based approaches. Our contribution in this work is a novel algorithm named Multi-objective Path-based D\* Lite (MOPBD\*), which leverages both path-based expansion in NAMOA\* and incremental search in D\* Lite. We compared the proposed MOPBD\* with both MOD\*, a node-based incremental method (baseline 1), and running NAMOA\* to search from scratch each time (baseline 2). The numerical results show that MOPBD\* outperforms both baselines.

## Problem Description

Let  $\mathcal{G} = (\mathcal{U}, \mathcal{E})$  denote a directed graph representing the workspace of the robot, where the node set  $\mathcal{U}$  denotes the set of possible locations for the robot and the directed edge (arc) set  $\mathcal{E} = \mathcal{U} \times \mathcal{U}$  denotes the set of actions that move the robot between any two nodes in  $\mathcal{U}$ . In addition, let  $pred(u)$  and  $succ(u)$  denote the set of predecessors and successors of a node  $u \in \mathcal{U}$ . We use  $u_o, u_d \in \mathcal{U}$  to denote the initial and destination node of the robot respectively and let  $u_c$  represent the current node of the robot during the navigation. Note that before the navigation starts,  $u_c = u_o$ . An edge from  $w$  to  $u$ ,  $w, u \in \mathcal{U}$ , is denoted as  $(w, u) \in \mathcal{E}$  and the cost of an edge  $e \in \mathcal{E}$  is a non-negative cost vector  $\vec{c}(w, u) \in (\mathbb{R}^+)^M$  with  $M$  being a positive integer.

In this work, let  $\pi(u_1, u_\ell)$  represent a path connecting  $u_1, u_\ell \in \mathcal{U}$  via a sequence of nodes  $(u_1, u_2, \dots, u_\ell)$  in  $\mathcal{G}$ , where  $u_k$  and  $u_{k+1}$  are connected by an edge  $(u_k, u_{k+1}) \in \mathcal{E}$ , for  $k = 1, 2, \dots, \ell - 1$ . Let  $\vec{g}(\pi(u_1, u_\ell))$  denote the cost vector corresponding to the path, which is the sum of the cost vector of all edges present in the path, *i.e.*  $\vec{g}(\pi(u_1, u_\ell)) = \sum_{k=1,2,\dots,\ell-1} \vec{c}(u_k, u_{k+1})$ .

To compare any two paths, we compare the cost vector associated with them using the dominance relationship (Ehrgott 2005):

**Definition 1 (Dominance)** *Given two vectors  $a$  and  $b$  of length  $M$ ,  $a$  dominates  $b$  (referred as  $a \succeq b$ ) if and only if  $a(m) \leq b(m) \forall m \in \{1, 2, \dots, M\}$ , and there exists  $m \in \{1, 2, \dots, M\}$  such that  $a(m) < b(m)$ .*

If  $a$  does not dominate  $b$ , this non-dominance is denoted as  $a \not\succeq b$ . Any two paths  $\pi_1(u_c, u_d), \pi_2(u_c, u_d)$  are non-dominated (to each other) if the corresponding cost vectors do not dominate each other. The set of all the non-dominated paths between  $u_c$  and  $u_d$  is called the *Pareto-optimal* set. A maximal subset of the Pareto-optimal set, where any two paths in this subset do not have the same cost vector, is called a cost-unique Pareto-optimal set and is denoted as  $\mathcal{S}^*(u_c, u_d)$ . To simply notation, we denote  $\mathcal{S}^*(u_c, u_d)$  simply as  $\mathcal{S}^*$ .

In this work, we aim to compute an  $\mathcal{S}^*$  when robot moves in a dynamic environment, *i.e.* the cost vector of edges in  $\mathcal{G}$  can change. Note that  $\mathcal{S}^*$  is computed repetitively as (1)  $u_c$  changes as robot moves and (2) the cost vector of edges can change during the navigation.

## Preliminary

### D\* Lite

D\* Lite (Koenig and Likhachev 2002) is an incremental algorithm based on A\* (Hart, Nilsson, and Raphael 1968), and aims to reuse previous search efforts to speed up the subsequent search when edge costs (scalar values) change. D\* Lite searches backwards from the destination  $u_d$  to the current node  $u_c$  so that the constructed search tree, which stores partial solution paths, can be reused as  $u_c$  changes. To make the presentation consistent, for the rest of the work, we present all the method by searching backwards from  $u_d$  to  $u_c$ .

At any time of the search, D\* Lite maintains two types of cost-to-come at a node  $u \in \mathcal{U}$ : the  $g$ -value  $g(u)$  and  $v$ -value  $v(u)$ .<sup>4</sup> Value  $v(u)$  stores the cost of the best path found from  $u_d$  to  $u_c$  during its last expansion, while  $g(u)$  is computed from the  $v$ -values of  $\text{succ}(u)$  (note that we are searching backwards), and thus, is potentially better informed than  $v(u)$ . Formally,

$$g(u) = \begin{cases} 0 & \text{if } u = u_d \\ \min_{u' \in \text{succ}(u)} v(u') + c(u, u') & \text{otherwise} \end{cases} \quad (1)$$

Based on the  $g$ - and  $v$ -values, a node  $u$  is *consistent* if  $v(u) = g(u)$ , and *inconsistent* otherwise. An inconsistent node  $u$  is either under-consistent ( $v(u) > g(u)$ ) or over-consistent ( $v(u) < g(u)$ ). To initialize,  $g$ -values of all nodes but  $u_d$  are set to  $\infty$  and  $v$ -values of all nodes are set to  $\infty$ . Clearly,  $u_d$  is the only inconsistent node at initialization.

Let  $h(u)$  denote the cost-to-go, which underestimates the cost of paths from  $u$  to  $u_c$ , and define  $f(u) := g(u) + h(u)$ . Based on that, D\* Lite defines the key of nodes as  $\text{key}(u) = [k_1(u), k_2(u)]$  with

$$\begin{aligned} k_1(u) &= \min\{v(u), g(u)\} + h(u) \\ k_2(u) &= \min\{v(u), g(u)\} \end{aligned}$$

Let OPEN denote the priority queue containing all candidate nodes to be expanded, where the nodes are prioritized by comparing their keys in lexicographic order. In other word,  $\text{key}(u) < \text{key}(w)$ ,  $u, w \in \mathcal{U}$  if  $k_1(u) < k_1(w)$  or both  $k_1(u) = k_1(w)$  and  $k_2(u) < k_2(w)$ . The OPEN in D\* Lite always contains all inconsistent nodes, and in each search iteration, the node with the minimum key is selected for expansion. To expand an inconsistent node  $u$ ,  $v(u)$  is made equal to  $g(u)$ , which makes  $u$  consistent, and for every node  $w \in \text{pred}(u)$ ,  $g(w)$  is updated based on Equation (1). Additionally, a parent pointer  $\text{parent}(u)$  is maintained at node  $u$  when  $u$  is expanded so that a path from  $u_d$  to  $u$  can be easily reconstructed by iteratively following the parent pointers. D\* Lite terminates when no candidate node in OPEN has a smaller key than  $\text{key}(u_c)$ , which guarantees that  $v(u_c)$  has reaches the minimum and an optimal solution path from  $u_d$  to  $u_c$  can be reconstructed.

When computing the initial solution path  $\pi(u_d, u_o)$  (note that  $u_c = u_o$ ), D\* Lite is equivalent to (backwards) A\* search. After the generation of  $\pi(u_d, u_o)$ , if edge costs

<sup>4</sup>We follow the convention in (Aine and Likhachev 2013), where  $g$ - and  $v$ -values are introduced.

change, D\* Lite recomputes the  $g$ -values of nodes that are immediately affected by those edges. Among those nodes, inconsistent ones are inserted into OPEN with updated keys. Then, D\* Lite runs in the same manner by iteratively expanding inconsistent nodes until all remaining nodes in OPEN have keys no less than  $\text{key}(u_c)$ .

### MOA\*

The basic difference between MO-SPP and SPP is that there are multiple non-dominated partial solution paths between any pair of nodes in the graph in general. Consequently, different from A\* where  $g, h, f$ -values are computed for each node, MOA\* (Stewart and White 1991) introduces  $G, H, F$  sets. The  $G(u)$ ,  $\forall u \in \mathcal{U}$  is a set of non-dominated cost vectors, each of which represents a non-dominated path from  $u_d$  to  $u$ . Similarly,  $H(u)$  is a set of heuristic vectors, each of which underestimates the cost of a non-dominated path from  $u$  to  $u_c$ . The  $F$ -set is defined as  $F(u) := \text{ND}\{\vec{g} + \vec{h} \mid \vec{g} \in G(u), \vec{h} \in H(u)\}$ , where  $\text{ND}(\cdot)$  is an operator that takes a set of vectors (denoted as  $B$ ) as input and computes the non-dominated subset of it (denoted as  $\text{ND}(B)$ ), i.e. for any  $a, b \in \text{ND}(B)$ ,  $a$  and  $b$  are non-dominated. To simplify the presentation without losing generality, we consider the case where  $H(u)$  of a node  $u$  contains only a single heuristic vector  $\vec{h}(u)$  that (component-wise) under-estimates the cost vector of all paths from  $u$  to  $u_c$ .

As MOA\* aims to find all cost-unique non-dominated paths  $\mathcal{S}^*$ , let  $\mathcal{S}$  denote the set of non-dominated cost vectors of the solutions found during the search. In every search iteration, MOA\* selects a node  $u$  from OPEN so that there exists  $\vec{f} \in F(u)$  that is non-dominated by any vector  $\vec{f}' \in F(u')$  for any other node  $u' \in \text{OPEN}$ ,  $u' \neq u$ , and expand node  $u$  by extending *all* partial solutions represented by vectors in  $G(u)$ . For every  $w \in \text{pred}(u)$ , a set of new partial solution paths represented by cost vectors  $G_{u,w} = \{\vec{g}(u) + \vec{c}(w, u) \mid \vec{g}(u) \in G(u)\}$  is computed and  $G(w) \leftarrow \text{ND}(G(w) \cup G_{u,w})$  so that  $G(w)$  contains all non-dominated cost vectors at  $w$  after expanding  $u$ . In addition,  $F(w)$  is updated and node  $w$  is added to OPEN for future expansion if there exists  $\vec{f} \in F(w)$  that is non-dominated by cost vectors of paths in  $\mathcal{S}$ .

There are two important features of MOA\* (node-based) that distinguish it from NAMOA\* (path-based), which is presented in the next section:

- when a new non-dominated partial solution is found at node  $u$  during the search, node  $u$  is (re-) inserted into OPEN;
- when a node  $u$  is selected from OPEN for expansion, all non-dominated partial solutions at  $u$  are extended.

These two features show that MOA\* takes a *node-based* expansion strategy. As one can expect, MOA\* can lead to a lot of re-expansion of nodes as there are multiple non-dominated partial solutions at each node for a MO-SPP. In addition, node expansion can be computational demanding as all partial solutions at this node need to be extended.

## MOD\*

D\* Lite and MOA\* can be combined as MOD\* algorithm<sup>5</sup> by introducing  $V$ -set at each node, which resembles the  $v$ -value of a node in D\* Lite, and stores the set of non-dominated cost vectors during its last expansion. Formally, it has the following relationship with the  $G$ -set.

$$G(u) = \begin{cases} \vec{0} & \text{if } u = u_d \\ \text{ND}(\bigcup_{w \in \text{succ}(u)} V(w) + \vec{c}(u, w)) & \text{otherwise} \end{cases} \quad (2)$$

Correspondingly, a node  $u$  is *consistent* if  $G(u) = V(u)$  (two sets are exactly the same) and *inconsistent* otherwise. Similar to D\* Lite, the OPEN in MOD\* contains inconsistent nodes. MOD\* iteratively selects inconsistent node  $u$  from OPEN for expansion until all vectors in  $F(u)$  of any inconsistent nodes  $u$  in OPEN are dominated by the cost of some path in  $\mathcal{S}$ . This termination condition guarantees that all cost-unique Pareto-optimal paths from  $u_d$  to  $u_c$  are found.

When the cost vector of an edge changes, MOD\* first re-computes the  $G$ -set of each node  $u$  that are immediately affected and inserts  $u$  into OPEN if  $u$  is inconsistent. Then MOD\* searches in the same manner by keep expanding inconsistent nodes until all Pareto-optimal paths are found.

## NAMOA\*

Both MOA\* and MOD\* expand nodes. As aforementioned, this node-based expansion may incur a lot of re-expansions and the expansion of a node can be computationally expensive. In NAMOA\* (Madow and De La Cruz 2008), a *path-based* expansion is developed to mitigate the drawbacks of the node-based expansion.

Let  $s = (u, \vec{g})$  denote a *state*, a tuple of a node  $u$  and a cost vector  $\vec{g}$ , which represents a specific partial solution path from  $u_d$  to  $u$  with cost  $\vec{g}$ . Additionally,  $\vec{g}$  is said to be *at* node  $u$  and state  $s$  is said to *contain*  $\vec{g}$  and  $u$ . Let  $u(s)$  and  $\vec{g}(s)$  denote the node and cost vector contained in  $s$ . For each state  $s$ , the  $f$ -vector of  $s$  is defined as  $\vec{f}(s) := \vec{g} + \vec{h}(u(s))$ . Different from MOA\*, where nodes are stored as candidates in OPEN, NAMOA\* stores states as candidates in OPEN. In every search iteration, NAMOA\* expands a non-dominated state  $s$  in OPEN, i.e.  $\vec{f}(s)$  is non-dominated by the  $f$ -vector of any other states in OPEN. To expand  $s$ , the partial solution path represented by  $s$  is extended to each predecessor  $w \in \text{pred}(u(s))$ , where a new state  $s_w = (w, \vec{g}_w)$  is generated. Cost vector  $\vec{g}_w$  is then compared with both the cost vector of other partial solution paths at  $w$  and the cost vector of any paths in  $\mathcal{S}$ . If  $\vec{g}_w$  is non-dominated,  $s_w$  is added to OPEN for future expansion.

As every state represents a partial solution path, expanding a state is essentially expanding a path. This path-based strategy employed by NAMOA\* avoids the large number of re-expansion of nodes as in MOA\*. In addition, path expansion is computationally much cheaper than node expansion.

<sup>5</sup>The MOD\* algorithm presented in this section simplifies the method in (Oral and Polat 2015) to highlight the key idea. Readers can refer to (Oral and Polat 2015) for more details.

## MOPBD\*

### Algorithm Overview

MOPBD\* inherits the notions of  $G, H, F$ -sets at nodes from MOA\*, the concept of  $V$ -sets from MOD\*, and the definition of states from NAMOA\*. In addition, we introduce a new concept of *inconsistent state* as follows, which differs from the inconsistent node in MOD\*.

**Definition 2 (consistent state)** A state  $s$ , with  $\vec{g}(s) \in G(u(s))$ , is *consistent* if  $\vec{g} \in V(u(s))$ , and *inconsistent* if  $\vec{g} \notin V(u(s))$ .

As shown in Algorithm 1, MOPBD\* initializes (line 1-3) by inserting a zero vector into  $G(u_d)$  and creates an initial state  $s_d$ . Since  $V(u_d) = \emptyset$  at initialization, state  $s_d$  is an inconsistent state by Def. 2 and is inserted into OPEN for expansion. In MOPBD\*, OPEN is a list containing all inconsistent states at any time of the search (Invariant-1). Then MOPBD\* starts its first planning task to compute  $\mathcal{S}^*$  via ComputePath procedure, as shown in Algorithm 2. If the robot has not yet reached its destination, MOPBD\* keeps receiving updating information about the cost vector of edges, finds all inconsistent states caused by changes in edge costs (ProcessEdge procedure) and re-computes  $\mathcal{S}^*$ . If there is no change in edge costs, the robot navigates towards the destination along planned paths.

---

### Algorithm 1 MOPBD\*

---

```

1:  $G(u_d) = \{\vec{0}\}$ 
2:  $s_d \leftarrow (u_d, h(u_d))$ 
3: add  $s_d$  into OPEN
4:  $\mathcal{S} \leftarrow \text{ComputePath}()$ 
5: while  $u_c \neq u_d$  and  $\mathcal{S} \neq \emptyset$  do
6:    $\mathcal{E}' \leftarrow$  the set of edges with updated cost vectors.
7:   if  $\mathcal{E}' \neq \emptyset$  then
8:     for all  $(w, u) \in \mathcal{E}'$  do
9:       ProcessEdge( $w, u$ )
10:    Update  $\mathcal{S}$  based on  $V(u_c)$ 
11:     $\mathcal{S} \leftarrow \text{ComputePath}()$ 
12:   else
13:     FollowPath( $\mathcal{S}$ )

```

---

### Compute Pareto-optimal Paths

As shown in Algorithm 2, in each search iteration, an inconsistent state  $s$ , with a non-dominated  $\vec{f}(s)$  is popped.

- If  $u(s) = u_c$ , a new solution path with cost vector  $\vec{g}(s)$  is found from  $u_d$  to  $u_c$  and is added to  $\mathcal{S}$ . In addition,  $\vec{g}(s)$  is used to filter OPEN, which removes any candidate states  $s' \in \text{OPEN}$  with  $\vec{g}(s) \succeq \vec{f}(s')$  or  $\vec{g}(s) = \vec{f}(s')$  (note that  $\vec{g}(s) = \vec{f}(s)$  as  $\vec{h}(s) = 0$ ), since the partial solution path represented by  $s'$  can not lead to a cost-unique Pareto-optimal solution.
- If  $u(s) \neq u_c$ ,  $\vec{g}(s)$  is added to  $V(u(s))$ , which makes state  $s$  a consistent state by Def. 2. Then, state  $s$  is expanded as follows.

To expand a state  $s$  (line 7), for each  $w \in \text{pred}(u(s))$ , a partial solution that reaches  $w$  from  $u(s)$  is generated and represented by state  $s_w$  with  $u(s_w) = w$  and  $\vec{g}(s_w) = \vec{g}(s) + \vec{c}(w, u)$ . For the rest of the work, to make the presentation easier, we introduce an additional notation  $s(\vec{g})$  to denote the state that contains  $\vec{g}$  during the search. Note that, at each node  $u \in \mathcal{U}$ , for each cost vector  $\vec{g}$  in  $G(u)$  or  $V(u)$ , there is a unique state  $s(\vec{g})$  generated during the search that contains  $\vec{g}$ .

After a state  $s_w$  is generated,  $\vec{g}(s_w)$  is compared with each vector in  $G(w)$  (line 9). If no vector in  $G(w)$  dominates or is equal to  $\vec{g}(s_w)$ ,  $\vec{g}(s_w)$  is then added to  $G(w)$  and is used to filter  $G(w)$ : all vectors in  $G(w)$  that are dominated by or equal to  $\vec{g}(s_w)$  are deleted (line 10-12). Specifically, to delete a vector  $\vec{a}$  from  $G(w)$ , all descendant states of  $s(\vec{a})$  and  $s(\vec{a})$  itself are deleted via a recursive procedure, as shown in Algorithm 3. Here, the Delete procedure is invoked recursively for each *children* state of  $s(\vec{a})$  until all descendant states of  $s(\vec{a})$  are removed. Similar to the parent pointer  $\text{parent}(s)$ , let  $\text{children}(s)$  denote a set of children pointers, each of which represents a children state that is reached from  $s$ . Finally, if the cost vector being deleted is at  $u_c$ , it means a solution path from  $u_d$  to  $u_c$  is invalidated and thus this solution is also removed from  $\mathcal{S}$ .

---

**Algorithm 2** ComputePath

---

```

1: while not ShouldTerminate() do
2:    $s = (u, \vec{g})$  is popped from OPEN
3:   if  $u = u_c$  then
4:     add  $\vec{g}$  to  $\mathcal{S}$ 
5:     filter OPEN with  $\vec{g}$ 
6:   add  $\vec{g}$  to  $V(u)$ 
7:   for all  $w \in \text{pred}(u)$  do
8:      $\vec{g}_w \leftarrow \vec{g} + c(w, u)$ 
9:     if  $\vec{a} \not\geq \vec{g}_w$  and  $\vec{a} \neq \vec{g}_w \ \forall \vec{a} \in G(u)$  then
10:      for all  $\vec{a} \in G(u)$  do
11:        if  $\vec{g}_w \succeq \vec{a}$  then
12:          Delete( $w, \vec{a}$ )
13:      $s_w \leftarrow (w, \vec{g}_w)$ 
14:      $\text{parent}(s_w) \leftarrow s$ 
15:     add  $s_w$  to  $\text{children}(s)$ 
16:     add  $\vec{g}_w$  into  $G(w)$ 
17:     add  $s_w$  into OPEN

```

---

With the Delete procedure, it is guaranteed that, at any time of the search,  $V(u), \forall u \in \mathcal{U}$  contains all cost vectors that represent non-dominated partial solutions from  $u_d$  to  $u$  (Invariant-2). This invariant is important for the following reason. Intuitively, if  $V(u)$  at some node  $u$  contains invalid cost vectors (whose ancestors are deleted), then  $G(w), w \in \text{pred}(u)$  may also contain invalid cost vectors by Equation (2). As a result, dominance check can potentially use those invalid vectors in  $G(w)$  to prune other cost vectors generated at  $w$  and the algorithm may fail to compute an  $\mathcal{S}^*$  correctly.

The search process continues until the termination condition is met: either OPEN is empty or for each (inconsistent) state  $s \in \text{OPEN}$ ,  $\vec{g}(s)$  is dominated by (or equal to) the cost

of some path in  $\mathcal{S}$ . Clearly, when the termination condition is satisfied, all remaining states in OPEN, if any, cannot be part of a cost-unique Pareto-optimal solution and the computed  $\mathcal{S}$  is a cost-unique Pareto-optimal set  $\mathcal{S}^*$ .

---

**Algorithm 3** Delete( $u, \vec{a}$ )

---

```

 $A \leftarrow \emptyset$ 
for all  $\vec{b} \in \text{children}(\vec{a})$  do
   $v \leftarrow$  the node at which  $\vec{b}$  locates
   $A \leftarrow A \cup \text{Delete}(v, \vec{b})$ 
remove  $\vec{a}$  from  $R(u)$ .
remove  $\vec{a}$  from  $G(u)$  if  $G(u)$  contains  $\vec{a}$ .
remove  $\vec{a}$  from  $\text{children}(\text{parent}(\vec{a}))$ .
if  $u = u_c$  and  $\vec{a} \in \mathcal{S}$  then
  remove  $\vec{a}$  from  $\mathcal{S}$ 
Return  $A$ 

```

---

**Process Edge Change**

After the initial computation of  $\mathcal{S}^*$ , Algorithm 1 either follows the planned paths or finds changes in edge costs. When the cost vector of an edge changes, the algorithm needs to pre-process before ComputePath is invoked again.

As shown in Algorithm 4, when the cost vector of an edge  $(w, u)$  changes, for each  $\vec{g} \in G(w)$ , if the corresponding state  $s(\vec{g})$  represents a partial solution via  $u$ , then this partial solution is affected by the change of the cost vector (line 2-5). Cost vector  $\vec{g}$  and all descendant states of  $s(\vec{g})$  are thus deleted by invoking the Delete procedure (line 6). In the meanwhile, let set  $A$  record all nodes  $u$ , at which  $V(u)$  changes. As  $V(u_a), \forall u_a \in A$  changes, the  $G$ -sets at the predecessors of  $u_a$  need to be re-computed correspondingly by Def. 2 (line 7-8). During the recomputation of  $G$ -sets, if a new inconsistent state is generated, the state is added to OPEN (line 9-11).

---

**Algorithm 4** ProcessEdge( $w, u$ )

---

```

1:  $A \leftarrow \emptyset$  ▷ set of nodes affected
2: for all  $\vec{g} \in G(w)$  do
3:    $s_p = \text{parent}(s(\vec{g}))$ 
4:   if  $u(s_p) = u$  then
5:      $A \leftarrow A \cup \text{Delete}(w, \vec{g})$ 
6:   for all  $u_a \in A$  do
7:     for all  $u'_a \in \text{pred}(u_a)$  do
8:        $G' \leftarrow \text{ND}(\bigcup_{u_k \in \text{succ}(u'_a)} V(u_k) + c(u'_a, u_k))$ 
9:       for all  $\vec{a} \in G' \setminus G(u'_a)$  do
10:         $s'_a \leftarrow (u'_a, \vec{a})$ 
11:        Add  $s'_a$  to OPEN
12:        $G(u'_a) \leftarrow G'$ 

```

---

By invoking ProcessEdge for each edge whose cost vector changes, both Invariant-1 and Invariant-2 are maintained. After processing the changed cost vectors,  $\mathcal{S}$  is made an empty set at first and then updated by reconstructing the paths represented by the states  $\{s_c = (u_c, \vec{g}_c), \forall \vec{g}_c \in$

$V(u_c)\}$  (line 10 in Algorithm 1). By doing so,  $\mathcal{S}$  now represents the set of solutions that have been found in the previous search and are still valid after cost changes. Finally, Algorithm 1 invokes ComputePath procedure to continue the search until the aforementioned termination condition is met.

## Discussion

There is a big difference between (single-objective) D\* Lite and the proposed MOPBD\*. D\* Lite only modifies the nodes that are immediately affected by the edge cost changes and lets the search process to “propagate” the inconsistency across the graph. However, MOPBD\* recursively finds all states that are affected in ProcessEdge all at once. There are three reasons behind this approach: (1) MOPBD\* uses  $\mathcal{S}$  to filter candidates in OPEN and  $\mathcal{S}$  should always contain valid non-dominated solutions; (2) MOPBD\* uses  $G(u)$  at node  $u$  to prune any newly generated cost vectors at  $u$  and  $G(u)$  should not contain invalid cost vectors so that incorrect pruning is avoided; (3) In multi-objective settings, the  $g$ -vectors of two partial solutions can be in-comparable (i.e. non-dominated) to each other. This makes it hard to introduce a “key” of a state similar to the key used in D\* Lite, so that all inconsistent states are expanded in an order (based on the key of states), where incorrect pruning can be avoided.

## Analysis

We show that MOPBD\* is able to compute all cost-unique Pareto-optimal solutions by leveraging the aforementioned Invariant-1, Invariant-2 and Equation (2). For the first planning task ( $u_c = u_o$ ), MOPBD\* searches in the same manner as NAMOA\* does and the computed  $\mathcal{S}$  contains all cost-unique Pareto-optimal solutions. When edge costs change, by invoking ProcessEdge procedure, all invalid cost vectors at nodes are removed, which maintains Invariant-2. In addition,  $G$ -sets of affected nodes are re-computed by Equation (2), where inconsistent states are identified and inserted into OPEN, which maintains Invariant-1. With Invariant-1, all inconsistent states to be expanded are contained in OPEN at any time of the search. In each search iteration, MOPBD\* selects an inconsistent state from OPEN and makes it consistent. During the search, with Invariant-2, all non-dominated partial solutions at each node is stored in the  $V$ -set of that node. MOPBD\* terminates only when no candidate states in OPEN can lead to a unique-cost non-dominated solution. Therefore, when MOPBD\* terminates,  $\mathcal{S} = V(u_c)$  and contains all cost-unique Pareto-optimal solutions from  $u_d$  to  $u_c$ .

**Theorem 1** *MOPBD\* computes all cost-unique Pareto-optimal paths from  $u_d$  to  $u_c$ .*

## Numerical Results

### Simulation Settings

We selected maps (grids) of different categories from a on-line data set (Stern et al. 2019) and generated a graph  $G$  by making each grid four-connected bi-directionally. To assign cost vectors to edges in  $G$ , we assigned every edge in the graph a random integer vector of length  $M$  with components

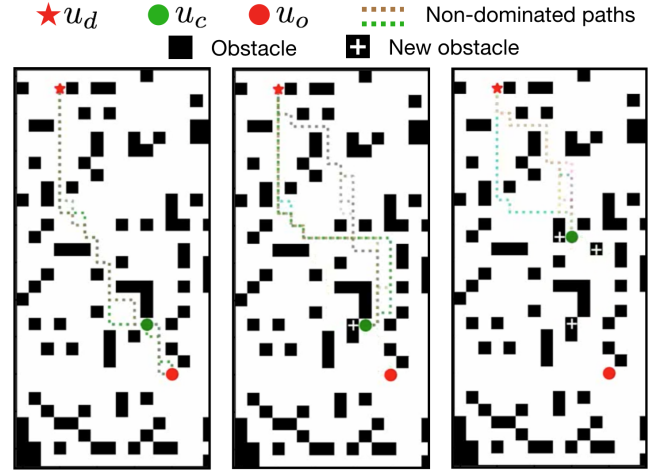


Figure 1: An illustration of the simulator. On the left, initial planning is finished and the robot is following a selected path (Step-1, Step-2). In the middle, a new obstacle is added in front of the robot and paths are replanned (Step-3, Step-4). On the right, the simulator keeps running after adding three obstacles.

randomly sampled from  $[1, 10]$ . To test a planning algorithm, which is called a “planner” hereafter, in a dynamic graph, we implemented the following simulator (Fig. 1). For each test instance, the simulator does the following steps in order.

- (Step-1) The planner computes the initial cost-unique Pareto-optimal set  $\mathcal{S}^*$  (note that  $u_c = u_o$ ).
- (Step-2) The simulator randomly selects a solution from  $\mathcal{S}^*$  for the robot to follow.
- (Step-3) After every  $k$  ( $k = 7$  in our tests) moves of the robot, the simulator adds an obstacle in front of the robot along the selected path.
- (Step-4) The simulator invokes the planner to re-compute  $\mathcal{S}^*$  (note that  $u_c \neq u_o$ ) and jumps back to Step-2.

The simulation terminates either when the robot arrives at  $u_d$  (i.e.  $u_c = u_d$ ), or when the computed  $\mathcal{S} = \emptyset$ , which means the added obstacle in Step-3 eliminates all feasible solutions.

In this work, we implemented MOD\*, NAMOA\* and MOPBD\* in Python. All algorithms use the same heuristic:  $\vec{h}(u), u \in \mathcal{U}$  is a unit vector scaled by the Manhattan distance between  $u$  and  $u_c$ . MOD\* is a node-based approach that searches by reusing previous efforts and NAMOA\* is a path-based approach that searches from scratch for each planning task. Both MOD\* and NAMOA\* serve as baselines to corroborate MOPBD\*.

### Node-based and Path-based

We summarized the test results in Table 1, where Exp. stands for the average number of expansions (either node expansion or path expansion, based on the algorithm), R.T. stands for the average run time and Sol. stands for the average number of solutions computed, and all averages are taken over



all subsequent planning tasks of all test instances. From Table 1, in terms of incremental search, MOPBD\* (path-based) runs faster than MOD\* (node-based) by roughly an order of magnitude. Note that, the number of expansions cannot be directly compared between MOPBD\* and MOD\* as they conduct node expansion and path expansion respectively. It is also worthwhile to note that for the last map (a game map of size  $65 \times 81$ ), the average number of solutions found by MOD\* is smaller than the other two algorithms. The reason is that MOD\* times out in some planning tasks.





Grids	Algorithm	Exp.	R.T.	Sol.
 (16x16)	NAMOA*	111.8	<b>0.03</b>	3.0
	MOD*	39.1	0.35	3.0
	MOPBD*	<b>3.9</b>	0.06	3.0
 (32x32)	NAMOA*	1556.6	0.55	10.5
	MOD*	92.1	3.15	10.5
	MOPBD*	<b>19.7</b>	<b>0.17</b>	10.5
 (32x32)	NAMOA*	829.5	0.22	4.9
	MOD*	311.0	3.51	4.9
	MOPBD*	<b>35.0</b>	<b>0.12</b>	4.9
 (65x81)	NAMOA*	5923.3	2.85	16.3
	MOD*	208.4	12.6	12.3
	MOPBD*	<b>28.0</b>	<b>2.43</b>	16.3

Table 1: Numerical results of the proposed MOPBD\* (path-based incremental search) compared with baselines: NAMOA\* (path-based from-scratch search) and MOD\* (node-based incremental search).

### Incremental Search and Search from Scratch

As shown in Table. 1, another comparison is between NAMOA\* and MOPBD\*, both of which conducts path-based expansion but NAMOA\* computes from scratch while MOPBD\* reuses previous search efforts. In terms of number of expansions, MOPBD\* outperforms NAMOA\* over all maps. In terms of run time, MOPBD\* outperforms NAMOA\* in general. However, as we observed in the  $16 \times 16$  empty map, MOPBD\* runs slower than NAMOA\* on average, which indicates that search from scratch (NAMOA\*) is even more efficient than reusing previous search efforts (MOPBD\*). The possible reason is that the ProcessEdge procedure in MOPBD\* is expensive when cost vectors change, as it requires all affected states to be deleted and requires recomputing the  $G$ -sets of all affected nodes over the entire graph.

### Detailed Run Time Comparison

To further investigate the performance of MOPBD\*, we visualized the run time of planning tasks for all instances in various maps as follows. Given a test instance, let  $l_k$  denote the average length of the Pareto-optimal paths computed in the  $k$ -th planning task in the instance. Note that when  $k = 0$ ,  $l_0$  denote the average length of the Pareto-optimal paths from the initial start to the destination. Additionally, the ratio  $l_k/l_0$  provides an estimate of the distance to the destination

for the  $k$ -th planning task. In Fig. 2, the horizontal axis represents the ratio  $l_k/l_0$  within range  $[0, 1]$  and the vertical axis represents the run time of planning tasks in all instances for a grid. Here, green dots correspond to running NAMOA\* from scratch for every planning task and stars correspond to MOPBD\*. Red stars mean that MOPBD\* is slower than NAMOA\*, while the blue stars mean MOPBD\* is faster. The digits on the upper left corner of each plot count the number of blue and red stars.

From Fig. 2, we can observe that, most of the red stars lie on the left half of the plot. This indicates that, when the robot is “close” to the destination, running NAMOA\* from scratch is more efficient than using MOPBD\*. The possible reason is that the ProcessEdge procedure in MOPBD\* is less efficient when the robot moves close to its destination: Modifying all affected paths and re-computing  $G$ -sets at nodes can be more computationally demanding than simply searching from scratch. It’s also worthwhile to mention that the result in Fig. 2 provides a practical guideline about using MOPBD\* in practice. One can compute  $l_k$  during the navigation process. When the robot is far away from the destination, (i.e. the estimated ratio  $l_k/l_0$  is larger than a pre-defined threshold), MOPBD\* is invoked to reuse previous search efforts. When the robot is close to the destination (i.e. the ratio is below the threshold), running NAMOA\* from scratch can potentially be more efficient.

## Conclusion and Future Work

A new incremental multi-objective path planning algorithm MOPBD\* is presented. We show that MOPBD\* is able to compute all cost-unique Pareto-optimal solutions. The numerical results demonstrate the efficiency of MOPBD\* with two objectives over two baseline approaches. For future work, one can consider incorporating other multi-objective or incremental search techniques into the algorithm to further improve the performance. One can also consider incremental approximation algorithms that can efficiently approximate the set of Pareto-optimal solutions.

## References

- Aine, S.; and Likhachev, M. 2013. Anytime truncated D\*: Anytime replanning with truncation. In *Sixth Annual Symposium on Combinatorial Search*.
- Bronfman, A.; Marianov, V.; Paredes-Belmar, G.; and L  er-Villagra, A. 2015. The maximin HAZMAT routing problem. *European Journal of Operational Research* 241(1): 15–27.
- Ehrgott, M. 2005. *Multicriteria optimization*, volume 491. Springer Science & Business Media.
- Goldin, B.; and Salzman, O. 2021. Approximate Bi-Criteria Search by Efficient Representation of Subsets of the Pareto-Optimal Frontier. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 31, 149–158.
- Hansen, P. 1980. Bicriterion path problems. In *Multiple criteria decision making theory and application*, 109–127. Springer.

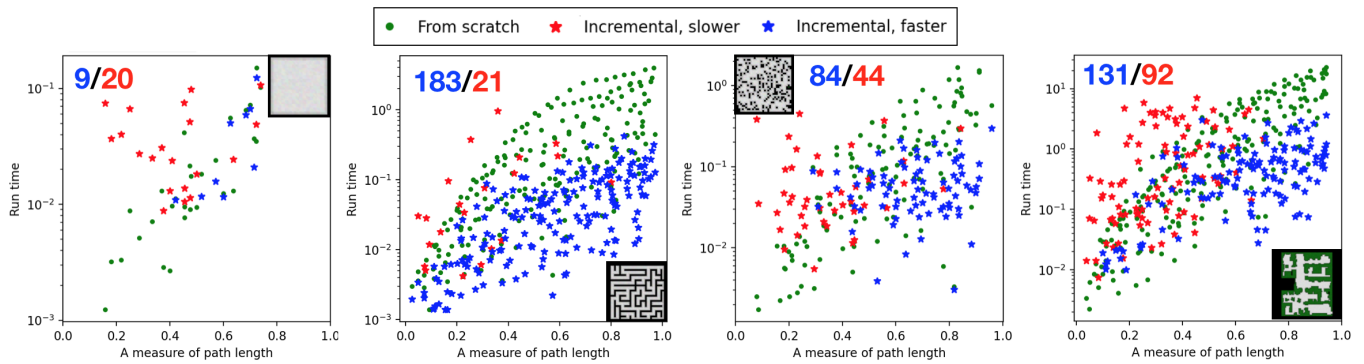


Figure 2: A detailed comparison between MOPBD\* and search from scratch using NAMOA\*. Green dots correspond to running NAMOA\* from scratch for every planning task and stars correspond to MOPBD\*. Red stars indicate that MOPBD\* runs slower than NAMOA\* while the blue stars mean MOPBD\* is faster. The digits on the upper left corner of each plot count the numbers of blue and red stars.

Hart, P. E.; Nilsson, N. J.; and Raphael, B. 1968. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics* 4(2): 100–107.

Hayat, S.; Yanmaz, E.; Brown, T. X.; and Bettstetter, C. 2017. Multi-objective UAV path planning for search and rescue. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, 5569–5574. IEEE.

Koenig, S.; and Likhachev, M. 2002. D\* lite. *AAAI/IAAI* 15.

Koenig, S.; and Likhachev, M. 2005. Fast replanning for navigation in unknown terrain. *IEEE Transactions on Robotics* 21(3): 354–363.

Koenig, S.; Likhachev, M.; and Furcy, D. 2004. Lifelong planning A\*. *Artificial Intelligence* 155(1-2): 93–146.

Likhachev, M.; Ferguson, D. I.; Gordon, G. J.; Stentz, A.; and Thrun, S. 2005. Anytime Dynamic A\*: An Anytime, Replanning Algorithm. In *Proceedings of the 15th International Conference on Automated Planning and Scheduling*, volume 5, 262–271.

Loui, R. P. 1983. Optimal paths in graphs with stochastic or multidimensional weights. *Communications of the ACM* 26(9): 670–676.

Madow, L.; and De La Cruz, J. L. P. 2008. Multiobjective A\* search with consistent heuristics. *Journal of the ACM (JACM)* 57(5): 1–25.

Oral, T.; and Polat, F. 2015. MOD\* Lite: an incremental path planning algorithm taking care of multiple objectives. *IEEE Transactions on Cybernetics* 46(1): 245–257.

Rojers, D. M.; Vamplew, P.; Whiteson, S.; and Dazeley, R. 2013. A survey of multi-objective sequential decision-making. *Journal of Artificial Intelligence Research* 48: 67–113.

Shan, T.; Wang, W.; Englot, B.; Ratti, C.; and Rus, D. 2020. A Receding Horizon Multi-Objective Planner for Autonomous Surface Vehicles in Urban Waterways. In *2020 59th IEEE Conference on Decision and Control (CDC)*, 4085–4092. IEEE.

Stentz, A. 1995. The focussed D\* algorithm for real-time replanning. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, volume 95, 1652–1659.

Stern, R.; Sturtevant, N.; Felner, A.; Koenig, S.; Ma, H.; Walker, T.; Li, J.; Atzmon, D.; Cohen, L.; Kumar, T.; et al. 2019. Multi-Agent Pathfinding: Definitions, Variants, and Benchmarks. In *Symposium on Combinatorial Search*, 151–158.

Stewart, B. S.; and White, C. C. 1991. Multiobjective A\*. *Journal of the ACM (JACM)* 38(4): 775–814.

Ulloa, C. H.; Yeoh, W.; Baier, J. A.; Zhang, H.; Suazo, L.; and Koenig, S. 2020. A Simple and Fast Bi-Objective Search Algorithm. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 30, 143–151.

Urmson, C.; Anhalt, J.; Bagnell, D.; Baker, C.; Bittner, R.; Clark, M.; Dolan, J.; Duggins, D.; Galatali, T.; Geyer, C.; et al. 2008. Autonomous driving in urban environments: Boss and the urban challenge. *Journal of Field Robotics* 25(8): 425–466.

# MarsExplorer: Exploration of Unknown Terrains via Deep Reinforcement Learning and Procedurally Generated Environments

Dimitrios I. Koutras<sup>1,2</sup>, Athanasios Ch. Kapoutsis<sup>2</sup>, Angelos A. Amanatiadis<sup>3</sup>, Elias B. Kosmatopoulos<sup>1,2</sup>

<sup>1</sup>Department of Electrical and Computer Engineering, Democritus University of Thrace, 671 00 Xanthi, Greece

<sup>2</sup>Information Technologies Institute, The Centre for Research & Technology, Hellas, 570 01 Thessaloniki, Greece

<sup>3</sup>Department of Production and Management Engineering, Democritus University of Thrace, 671 00 Xanthi, Greece  
{dkoutras, athakapo, kosmatop}@iti.gr, aamanat@pme.duth.gr

## Abstract

This paper is an initial endeavor to bridge the gap between powerful Deep Reinforcement Learning methodologies and the problem of exploration/coverage of unknown terrains. Within this scope, MarsExplorer, an openai-gym compatible environment tailored to exploration/coverage of unknown areas, is presented. MarsExplorer translates the original robotics problem into a Reinforcement Learning setup that various off-the-shelf algorithms can tackle. Any learned policy can be straightforwardly applied to a robotic platform without an elaborate simulation model of the robot's dynamics to apply a different learning/adaptation phase. One of its core features is the controllable multi-dimensional procedural generation of terrains, which is the key for producing policies with strong generalization capabilities. Four different state-of-the-art RL algorithms (A3C, PPO, Rainbow, and SAC) are trained on the MarsExplorer environment, and a proper evaluation of their results compared to the average human-level performance is reported. In the follow-up experimental analysis, the effect of the multi-dimensional difficulty setting on the learning capabilities of the best-performing algorithm (PPO) is analyzed. A milestone result is the generation of an exploration policy that follows the Hilbert curve without providing this information to the environment or rewarding directly or indirectly Hilbert-curve-like trajectories. The experimental analysis is concluded by comparing PPO learned policy results with frontier-based exploration context for extended terrain sizes. The source code can be found at: <https://github.com/dimikout3/GeneralExplorationPolicy>.

## Introduction

At this very moment, three different uncrewed spaceships, PERSEVERANCE (USA), HOPE (UAE), TIANWEN-1 (China), are heading towards Mars. Never before will such a diverse array of scientific gear have arrived at a foreign planet at the same time, and with such broad ambitions [Witze *et al.*2020]. On top of that, several lunar missions have been arranged for this year to enable extensive experimentation, investigation, and testing on an extraterrestrial body [Smith *et al.*2020]. In this exponentially growing field of extraterrestrial missions, a task of paramount importance

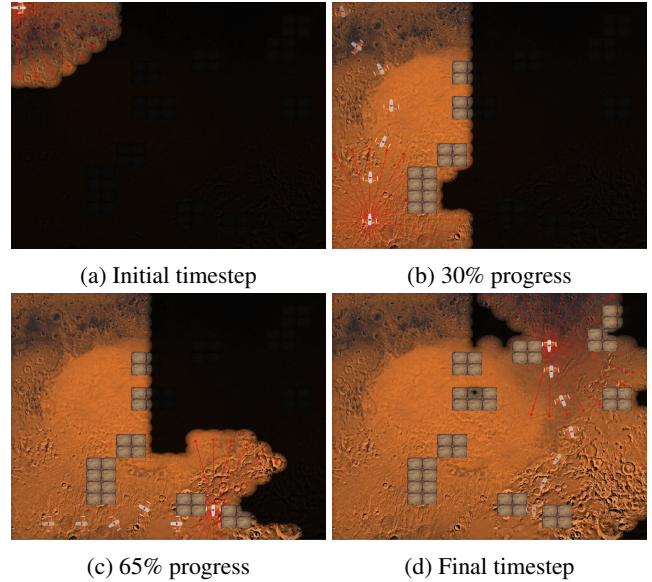


Figure 1: Indicative example: Trained RL agent executes exploration/coverage task in previously unknown and cluttered terrain utilizing MarsExplorer environment.

is the autonomous exploration/coverage of previously unknown areas. The effectiveness and efficiency of such autonomous explorers may significantly impact the timely accomplishment of crucial tasks (e.g., before the fuel depletion) and, ultimately, the success (or not) of the overall mission.

Exploration/coverage of unknown territories is translated into the online design of the path for the robot, taking as input the sensory information and having as objective to map the whole area in the minimum possible time [Shrestha *et al.*2019] [Kapoutsis *et al.*2016]. This setup shares the same properties and objectives with the well-known NP-complete setup of Traveling Salesman Problem (TSP), with the even more restrictive property that the area to be covered is discovered incrementally during the operation.

## Related Work

The well-established family of approaches incorporates the concept of *next best pose* process, i.e. a turn-based, greedy selection of the next best position (also known as *frontier-cell*) to acquire measurement, based on heuristic strategy (e.g., [Koutras *et al.*2020], [Renzaglia *et al.*2019], Simmons *et al.*2000], [Basilico and Amigoni2011]). Although this family of approaches has been extensively studied, some inherent drawbacks significantly constrain its broader applicability. For example, every deadlock that may arise during the previously described optimization scheme should have been predicted, and a corresponding mitigation plan should have been already in place [Palacios-Gasós *et al.*2016]; otherwise, the robot is going to be stuck in this locally optimal configuration. On top of that, to engineer a multi-term strategy that reflects the task at hand is not always trivial [Popov *et al.*2017].

The recent breakthroughs in Reinforcement Learning (RL), in terms of both algorithms and hardware acceleration, have spawned methodologies capable of achieving above human-level performance in high-dimensional, non-linear setups, such as the game of Go [Silver *et al.*2016], Atari games [Mnih *et al.*2015a], Multi-agent collaboration [Baker *et al.*2019]. A milestone in the RL community was the standardization of several key problems under a common framework, namely openai-gym [Brockman *et al.*2016]. Such release eased the evaluation among different methodologies and ultimately led to the generation of a whole new series of RL frameworks with standardized algorithms (e.g., [Dhariwal *et al.*2017], [Liang *et al.*2018]), all tuned to tackle openai-gym compatible setups.

These breakthroughs motivated the appliance of RL methodologies in the path-planning/exploration robotic tasks. Initially, the problem of navigating a single robot in previously unknown areas to reach a destination, while simultaneously avoiding catastrophic collisions, was tackled with RL methods [Lei *et al.*2018] [Wen *et al.*2020] [Zhang *et al.*2018]. The first RL methodology solely developed for exploration of unknown areas was developed in [Niroui *et al.*2019], and has successfully presented the potential benefits of RL. Recently, there have been proposed RL methodologies that seek to leverage the deployment of multi-robot systems to cover an operational area [Luis *et al.*2021].

However, [Luis *et al.*2021] assumes only a single geometry for the environment to be covered, and thus being prone to overfit, rather than being able to generalize in different environments. [Niroui *et al.*2019] mitigates this drawback by introducing a learning scheme with 30 different environments during the training phase. Although such a methodology can adequately tackle the generalization problem, the RL agent’s performance is still bounded to the diversity of the human-imported environments.

## Contributions

The main contribution of this work is to provide a framework for learning exploration/coverage policies that possess strong generalization abilities due to the procedurally gen-

erated terrain diversity. The intuition behind such an approach to exploration tasks is the fact that most areas exhibit some kind of structure in their terrain topology, e.g., city blocks, trees in a forest, containers in ports, office complexes. Thereby, by training multiple times in such correlated and procedurally generated environments, the robot will grasp/understand the underlining structure and leverage it to efficiently complete its goal, even in areas that it has never been exposed to.

Within this scope, a novel openai-gym compatible environment for exploration/coverage of unknown terrains has been developed and is presented. All the core elements that govern a real exploration/coverage setup have been included. MarsExplorer is one of the few RL environments where any learned policy can be transferred to real-world robotic platforms, providing that a proper translation between the proprioceptive/exteroceptive sensors’ readings and the generation of 2D perception (occupancy map), as depicted in figure 2, and also an integration with the existing robotic systems (e.g., PID low level control, safety mechanisms, etc.) are implemented.

Four state-of-the-art RL algorithms, namely A3C [Mnih *et al.*2016], PPO [Schulman *et al.*2017], Rainbow [Hessel *et al.*2018] and SAC [Haarnoja *et al.*2018], have been evaluated on MarsExplorer environment. To better comprehend these evaluation results, the average human-level performance in the MarsExplorer environment is also reported. A follow-up analysis utilizing the best-performing algorithm (PPO) is conducted with respect to the different levels of difficulty. The visualization of the produced trajectories revealed that the PPO algorithm had learned to apply the famous space-filling Hilbert curve, with the additional capability of avoiding on-the-fly obstacles that might appear on the terrain. The analysis is concluded with a scalability study and a comparison with non-learning methodologies.

It should be highlighted that the objective is not to provide another highly realistic simulator but a framework upon which RL methods (and also non-learning approaches) will be efficiently benchmarked in exploration/coverage tasks. Although there are available several wrappers for high-fidelity simulators (e.g. Gazebo [Zamora *et al.*2016], ROS [Lopez *et al.*2019]) that could be tuned to formulate an exploration coverage setup, in practice the required execution time for each episode severely limits the type of algorithms that can be used (for example PPO usually needs several millions of steps to environment interactions to converge). To the best of our knowledge, this is the first openai-gym compatible framework oriented for robotic exploration/coverage of unknown areas.

Figure 1 presents 4 sample snapshots that illustrate the performance of a trained RL robot inside the MarsExplorer environment. Figure 1a demonstrates the robot’s entry inside the unknown terrain, which is annotated with black color. Figure 1b illustrates all the so-far gained “knowledge”, which is either depicted with Martian soil or brown boxes to denote free space or obstructed positions, respectively. An attractive trait is depicted in figure 1c, where the robot chose to perform a dexterous maneuver between two obstacles to be as efficient as possible in terms of numbers



of timesteps for the coverage task. Note that any collision with an obstacle would have resulted in a termination of the episode and, as a result, an acquisition of an extreme negative reward. Figure 1d illustrates the robot's final position, along with all the gained information for the terrain (non-black region) during the episode.

## Environment

This section identifies the fundamental elements that govern the family of setups that fall into the coverage/exploration class and translates them to the openai gym framework [Brockman *et al.* 2016]. In principle, the objective of the robot is to cover an area of interest in the minimum possible time while avoiding any non-traversable objects, the position of which gets revealed only when the robot's position is in close proximity [Kapoutsis *et al.* 2019], [Burgard *et al.* 2000].

## Setup

Let us assume that area to be covered is constrained within a square, which has been discretized into  $n = rows \times cols$  identical grid cells:

$$\mathcal{G} = \{(x, y) : x \in [1, rows], y \in [1, cols]\} \quad (1)$$

The robot cannot move freely inside this grid, as some grid cells are occupied by non-traversable objects (obstacles). Therefore, the map of the terrain is defined as follows:

$$\mathcal{M}(q) = \begin{cases} 0.3 & \text{free space} \\ 1 & \text{obstacle} \end{cases} \quad q = (x, y) \in \mathcal{G} \quad (2)$$

The values of  $\mathcal{M}$  correspond to the morphology of the unknown terrain and are considered a priori unknown.

## Action space

Keeping in mind that the movement capabilities of the robot mainly impose the discretization of the area into grid cells, the action space is defined in the same grid context as well. The position of the robot is denoted by the corresponding  $x, y$  cell of the grid, i.e.  $p_a(t) = [x_a(t), y_a(t)]$ . Then, the possible next actions are simply given by the Von Neumann neighborhood [Gray *et al.* 2003], i.e.

$$\mathcal{A}_{p_a} = \{(x, y) : |x - x_a| + |y - y_a| \leq 1\} \quad (3)$$

In the openai-gym framework, the formulation above is realized by a discrete space of size 4 (North, East, South, West).

## State space

With each movement, the robot may acquire some information related to the formation of the environment that lies inside its sensing capabilities, according to the following lidar-like model:

$$y_q(t) = \begin{cases} 1 & \text{if } \|p_a(t) - q\| \leq d \text{ AND} \\ & \exists \text{ line-of-sight between} \\ & p_a(t) \text{ and } q \\ 0 & \text{otherwise} \end{cases} \quad \forall q \in \mathcal{G} \quad (4)$$

where  $d$  denotes the maximum scanning distance.

An auxiliary boolean matrix  $D(t)$  is introduced to denote all the cells that have been discovered from the beginning till  $t$  timestep.  $D(t)$  annotates with one all cells that have been sensed and with zero all the others. Starting from a zero matrix  $rows \times cols$ , its values are updated as follows:

$$D_q(t) = D_q(t-1) \vee y_q(t), \quad \forall q \in \mathcal{G} \quad (5)$$

where  $\vee$  denotes the logical OR operator. The state is simply an aggregation of the acquired information over all past measurements of the robot (4). Having updated (5), the state  $s(t_k)$  is a matrix of the same size as the grid to be explored (1), where its values are given by:

$$s_q(t) = \begin{cases} \mathcal{M}_q & \text{if } D_q(t) \\ 0 (= \text{undefined}) & \text{otherwise} \end{cases} \quad \forall q \in \mathcal{G} \quad (6)$$

Finally, the robot's position is declared by making the value of the corresponding cell equal to 0.6, i.e.  $s_{q=p_a(t)}(t) = 0.6$ . Overall, state  $s(t)$  is defined as a 2D matrix, that takes values from the following discrete set:  $\{0, 0.3, 0.6, 1\}$ . Figure 2 presents an illustrative example of a registration between the graphical environment (figure 2a) and the corresponding state representation (figure 2b).

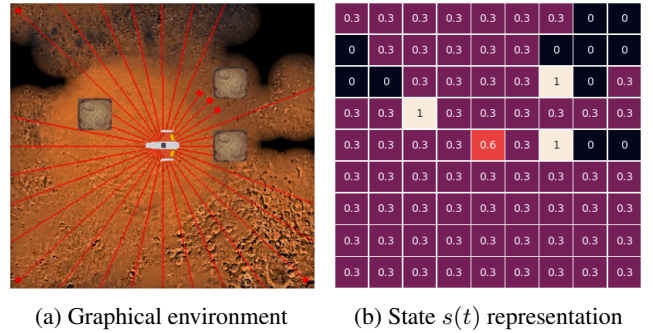


Figure 2: State encoding

## Reward function

Having in mind that the ultimate objective is to discover all grid cells, the instantaneous core reward, at each timestep  $t$ , is defined as the number of newly explored cells, i.e.

$$r_{explor}(t) = \sum_{q \in \mathcal{G}} D_q(t) - \sum_{q \in \mathcal{G}} D_q(t-1) \quad (7)$$

Intuitively, if  $\sum_{k=0}^T r_{explor}(k) \rightarrow n$ , then the robot has explored the whole grid (1) in  $T$  timesteps.

To force robot to explore the whole area (7), while avoiding unnecessary movements, an additional penalty  $r_{move} = 0.5$  per timestep is applied. In essence, this negative reward aims to distinguish among policies that lead to the same number of discovered cells but needed a different number of exploration steps. Please note that the value of  $r_{move}$  should be less than 1, to have less priority than the exploration of a single cell.

The action space, as defined previously, may include invalid next movements for the robot, i.e., out of the operational area (1) or crashing into a discovered obstacle.

Thus, apart from the problem at hand, the robot should be able to recognize these undesirable states and avoid them at all costs. Towards that direction, an additional penalty  $r_{invalid} = n$  is introduced for the cases where the next robot’s movement leads to an invalid state. Along with such a reward, the episode is marked as “done”, indicating that a new episode should be initiated.

At the other side of the spectrum, a completion bonus  $r_{bonus} = n$  is given to the robot when more than  $\beta\%$  (e.g., 95%) of the cells have been explored. Similar to the previous case, this is also considered a terminal state.

Putting everything together, the reward is defined as:

$$r(t) = \begin{cases} -r_{invalid} & \text{if next state is invalid} \\ \begin{cases} r_{explor}(t) - r_{move} + \\ r_{bonus} & \text{if } \frac{\sum_{q \in \mathcal{G}} D(t)}{n} \geq \beta \\ 0 & \text{otherwise} \end{cases} & \text{otherwise} \end{cases} \quad (8)$$

## Key RL Attributes

MarsExplorer was designed as an initial endeavor to bridge the gap between powerful existing RL algorithms and the problem of autonomous exploration/coverage of a previously unknown, cluttered terrain. This subsection presents the build-in key attributes of the designed framework.

**Straightforward applicability.** One of the fundamental attributes of MarsExplorer is that any learned policy can be straightforwardly applied to an appropriate robotic platform with little effort required. This can be achieved by the fact that the policy calculates a high-level exploration path based on the perception of the environment (6). Thus, assuming that a smooth integration with the sensor’s readings (for example, using a Kalman filter), can be used to represent the environment as in (6), no elaborate simulation model of the robot’s dynamics is required to adjust the RL algorithm into the specifics of the robotic platform.

**Terrain Diversity.** For each episode, the general dynamics are determined by a specific automated process that has different levels of variation. These levels correspond to the randomness in the number, size, and positioning of obstacles, the terrain scalability (size), the percentage of the terrain that the robot must explore to consider the problem solved, and the bonus reward it will receive in that case. This procedural generation [Cobbe *et al.*2020] of terrains allows training in multiple/diverse layouts, forcing, ultimately, the RL algorithm to enable generalization capabilities, which are of paramount importance in real-life applications where unforeseen cases may appear.

**Partial Observability.** Due to the nature of the exploration/coverage setup, at each timestep, the robot is only aware of the location of the obstacles that have been sensed from the beginning of the episode (5). Therefore, any long-term plan should be agile enough to be adjusted on the fly, based on future information about the unknown obstacles’ positions. Such a property renders the acquisition of a global exploration strategy quite tricky.

**Fast Evaluation.** Disregarding the environment from any irrelevant physics dynamics and focusing only on the explo-

ration/coverage aspect (1)-(8), MarsExplorer allows rapid execution of timesteps. This feature can be of paramount importance in the RL ecosystem, where the algorithms usually need millions of timesteps to converge, as it can enable fast experimental pipelines and prototyping.

## Performance Evaluation

This section presents an experimental evaluation of the MarsExplorer environment. The analysis begins with all the implementation details that are important for realizing the MarsExplorer experimental setup. For the first evaluation iteration, 4 state-of-art RL algorithms are applied and evaluated in a challenging version of MarsExplorer that requires the development of *strong generalization* capabilities in a highly randomized scenario, where the underlying structure is almost absent. Having identified the best performing algorithm, a follow-up analysis is performed with respect to the difficulty vector values. The learned patterns and exploration policies for different evaluation instances are further investigated and graphically presented. The analysis is concluded with a scale-up study in two larger terrains and a comparison between the trained robot and two well-established frontier-based approaches.

## Implementation details

Aside from the standardization as an openai-gym environment, MarsExplorer provides an API that allows manually controlled experiments, translating commands from keyboard arrows to next movements. Such a feature can assess human-level performance in the exploration/coverage problem and reveal important traits by comparing human and machine-made strategies.

Ray/RLlib framework [Liang *et al.*2017] was utilized to perform all the experiments. The fact that RLlib is a well-documented, highly-robust library also eases the build-on developments (e.g., apply a different RL pipeline), as it follows a common framework. Furthermore, such an experimental setup may also leverage the interoperability with other powerful frameworks from the Ray ecosystem, e.g., Ray/Tune for hyperparameters’ tuning.

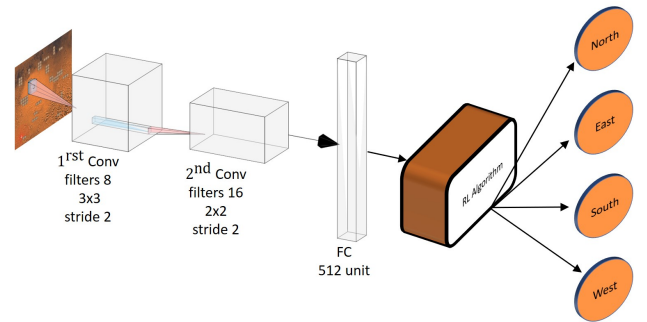


Figure 3: Overview of the experimental architecture

Table 1 summarizes all the fixed parameters used for all the performed experiments. MarsExplorer admits the distinguishing property of stochastically deploying the obsta-



Table 1: Implementation parameters

Parameter	Value	Equation
Grid size	$[21 \times 21]$	(1)
Sensor radius	$d = 6$ grid cells	(4)
Considered done	$\beta = 99\%$	(8)

cles at the beginning of each episode. This stochasticity can be controlled and ultimately determines the difficulty level of the MarsExplorer setup. The state-space of MarsExplorer has a strong resemblance to thoroughly studied 2D environments, e.g., ALE [Bellemare *et al.*2013], only with the key difference that the image is generated incrementally and based on the robot’s actions. Therefore, as it has been standardized from the DQN algorithm’s application domain [Mnih *et al.*2015b], a vision-inspired neural network architecture is incorporated as a first stage. Figure 3 illustrates the architecture of this *pre-processor*, which is comprised of 2 convolutional layers followed by a fully connected one. The vectorized output of the fully connected layer is forwarded to a “controller” architecture dependent on the RL algorithm enabled.

### State-of-the-art RL algorithms comparison

Apart from the details described in the previous subsection, for the comparison study, at the beginning of each episode, the formation (position and shape) of obstacles was set randomly. This choice was made to force RL algorithms to develop novel generalization strategies to tackle such a challenging setup. The list of studied RL algorithms is comprised by the following model-free approaches: PPO [Schulman *et al.*2017], DQN-Rainbow [Hessel *et al.*2018], A3C [Mnih *et al.*2016] and SAC [Haarnoja *et al.*2018]. All hyperparameters of these algorithms are reported in the Appendix.

Figure 4 presents a comparison study among the approaches mentioned above. For each RL agent, the thick colored lines stand for the episode’s total reward, while the transparent surfaces around them correspond to the standard deviation. Moreover, the episode’s reward (*score*) is normalized in such a way that 0 stands for an initial invalid action by the robot,  $r_{invalid}$  in (8), while 1 correspond to the theoretical maximum reward, which is the  $r_{bonus}$  in (8) plus the number of cells.

To increase the qualitative comprehension of the produced results, the average human-level performance is also introduced. To approximate this value, 10 players were drawn from the pool of *CERTH/ConvCAO* employees to participate in the evaluation process. Each player had an initial warm-up phase of 15 episodes (non-ranked), and after that, they were evaluated on 30 episodes. The average achieved score of the 300 human-controlled experiments is depicted with a green dashed line.

A clear-cut outcome is that the PPO algorithm achieves the highest average episodic reward, reaching an impressive 85.8% of the human-level performance. DQN-Rainbow achieves the second-best performance; however, the average is 50.04% and 42.73% of the PPO and human-level perfor-

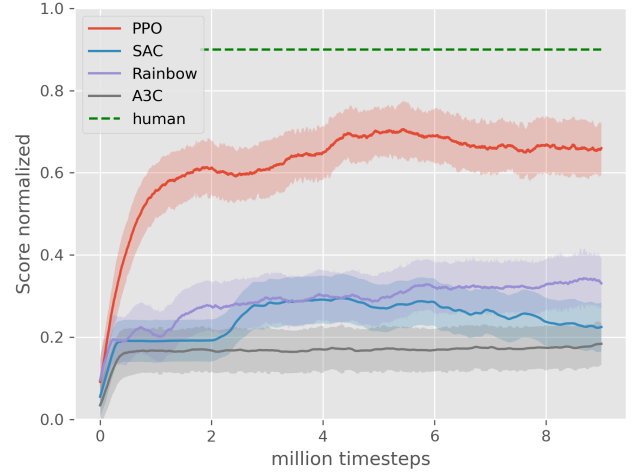


Figure 4: Learning curves for MarsExplorer with randomly chosen obstacles.

mance, respectively.

### Multi-dimensional difficulty

Having defined the best performing RL algorithm (PPO), now the focus is shifted on producing some preliminary results, related with the difficulty settings of MarsExplorer. As mentioned in the definition section, MarsExplorer allows for setting the elements of difficulty vector independently. More specifically, the difficulty vector comprised of 3 elements  $[d_t, d_m, d_b]$ , where:

- $d_t$  denotes the **topology stochasticity**, which defines the obstacles’ placement on the field. The *fundamental positions* of the obstacles are equally arranged in a 3 columns – 3 rows format.  $d_t$  controls the radius of deviation around these *fundamental positions*. As the value of  $d_t$  increases, the obstacles’ topology has more unstructured formation.  $d_t$  takes values from  $\{1, 2, 3\}$  discrete set.
- $d_m$  denotes the **morphology stochasticity**, which defines the obstacles’ shape on the field.  $d_m$  controls the area that might be occupied from each obstacle. The bigger the value of  $d_m$ , the larger the compound areas of obstacles that might appear on the MarsExplorer terrain.  $d_m$  takes values from  $\{1, 2\}$  discrete set.
- $d_b$  denotes the **bonus rewards**, that are assigned for the completion ( $r_{bonus}$ ) and failure ( $r_{invalid}$ ) of the mission (8). For this factor only two values are allowed  $\{1, 2\}$ , that correspond to cases of providing and not-providing the bonus rewards, respectively.

Higher values in the elements of the difficulty vector correspond to less structured behavior in the obstacles formation. Thus, a trained agent that has been successfully trained in greater difficulty setups may exhibit increased generalization abilities. Overall, the aggregation of the aforementioned elements’ domain generates 12 combinations of difficulty levels. Figure 5 shows the total average return of the evolution of the average episodic reward for each one of the 12 levels during the training of the PPO algorithm. To improve the readability of the graphs, the results are organized into 3

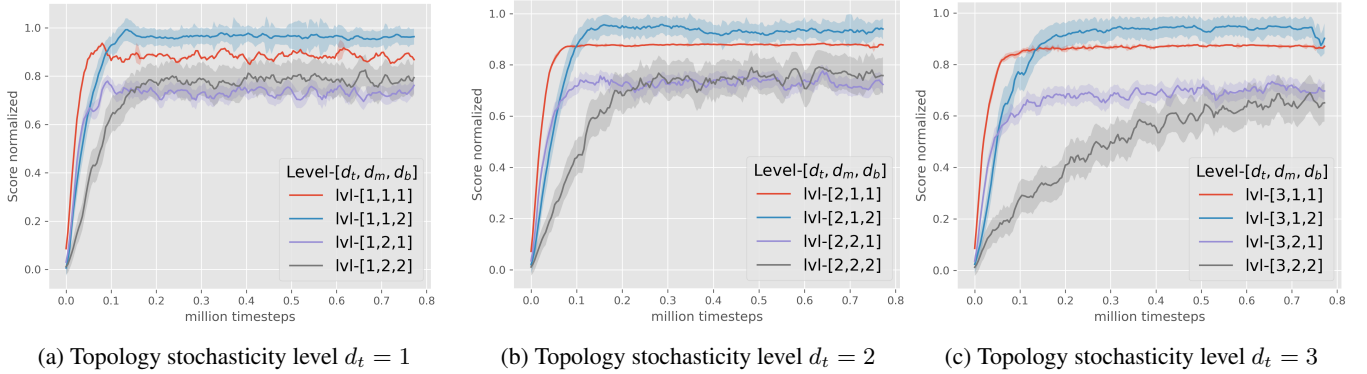


Figure 5: The sensitivity of PPO algorithm learning curves with respect to the different levels of multi-dimensional difficulty vector.

graphs, one for each level of  $d_t$ , with 4 plot lines each.

A study on the learning curves reveals that  $d_m$  has the largest effect on the learned policy. Blue and red lines (cases where  $d_m = 1$ ), in all three figures, demonstrate a similar convergence rate and also the highest-performance policies. However, a serious degradation in the results is observed in purple and gray lines ( $d_m = 2$ ). As it was expected, when  $d_m = 2$  and also  $d_t = 3$  (purple and gray lines in figure 5c) the final achieved performance reached only a little bit above 0.6 in the normalized scale.  $d_b$  seems that does not affect much the overall performance, at least until this vector of difficulty, apart from the convergence rate depicted in the gray line of figure 5c.

### Learned policy evaluation

This section is devoted to the characteristics of the learned policy from the PPO algorithm. For each of the 12 levels of difficulty defined in the previous section, the best PPO policy was extracted and evaluated in a series of 100 experiments with randomly (controlled by the difficulty setting) generated obstacles. Figure 6 presents one heat map for each difficulty level. Blue colormap corresponds to the frequency of the robot visiting a specific cell of the terrain. Green colormap corresponds to the number of detected obstacles in each position during the robot’s exploration.

A critical remark is that, for each scenario, the arrangement of discovered obstacles matches the drawn distribution as described in the previous subsection, implying that the learned policy does not have any “blind spots”.

Examining the heatmap of the trajectories in each scenario, it is crystal clear that the same family of trajectories has been generated in all cases and with great confidence. The important conclusion here is that this pattern is the first order of the Hilbert curve that has been utilized extensively in the space-filling domain (e.g., [Kapoutsis *et al.* 2017], [Sadat *et al.* 2015]). Please highlight that such a pattern has not been imported to the simulator or rewarded when achieved from the RL algorithm; however, the algorithm learned that this is the most effective strategy by interacting with the environment.

It would be an omission not to mention the learned policy’s ability to adapt to changes in the obstacles’ distribution

and, ultimately, find the most efficient obstacle-free route. This trait can be observed more clearly in subfigures 6k and 6l, where the policy needed to be extremely dexterous and delicate to avoid obstacles’ encounters.

### Comparison with frontier-based methodologies for varying terrain sizes

The analysis is concluded with a scalability study and comparison to non-learning methodologies. Two terrains with sizes  $[42 \times 42]$  and  $[84 \times 84]$  were used. The difficulty level was set to  $[d_t, d_m, d_b] = [2, 2, 1]$ , while 100 experiments were conducted for each scenario. Utility and cost-based frontier cell exploration methodologies [Basilico and Amigoni 2011] were enabled for positioning the achieved PPO policy in the context of non-learning approaches. In these frontier-based approaches, the exploration policy is divided into two categories based on the metric to be optimized:

- *Cost*: the next action is chosen based on the distance from the nearest frontier cell.
- *Utility*: the decision-making is governed by frequently updated information potential field.

Figure 7 summarizes the result of such evaluation study by presenting the average exploration time for each algorithm (PPO, cost frontier-based, utility frontier-based) over 100 procedurally generated runs. A direct outcome is that the learning-based approach requires the robot to travel less distance to explore the same percentage of terrain as the non-learning approaches. The final remark is devoted to the “knee” that can be observed in almost all the final stages of the non-learning approaches. Such behavior is attributed to having several distant sub-parts of the terrain unexplored, the exploration of which requires this extra effort. On the contrary, the learning-based approach (PPO) seems to handle this situation quite well, not leaving these expensive-to-revisit regions along its exploration path.

### Conclusions

A new openai-gym environment called MarsExplorer that bridges the gap between reinforcement learning and the real-life exploration/coverage in the robotics domain is presented. The environment transforms the well-known

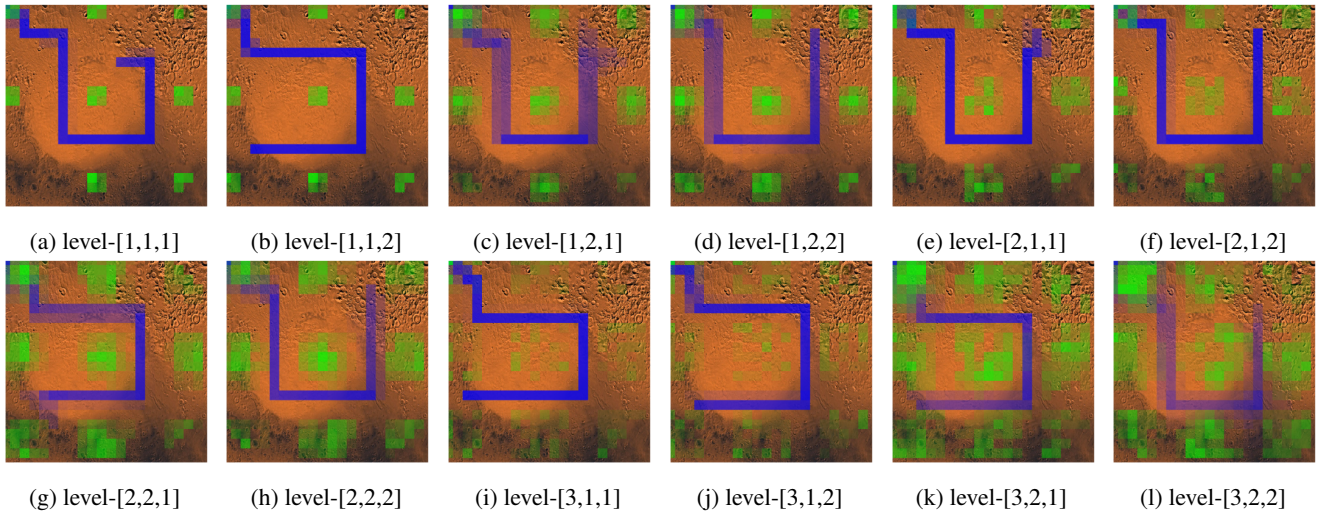


Figure 6: Heatmap of the evaluation results of the learned PPO policy. For each of the 12 difficulty levels, 100 experiments were performed, with the randomness in obstacles’ formation as imposed by the corresponding level. Blue colormap corresponds to the frequency of cell visitations by the RL agent, while green colormap corresponds to the location of the encountered obstacles for all the evaluations.

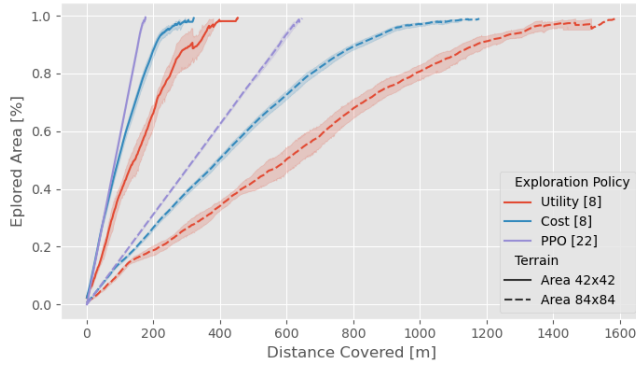


Figure 7: Comparison between 3 exploration methodologies, depicting the average and standard deviation over 100 procedurally generated environments. Red and blue colors correspond to the non-learning approaches, while purple color corresponds to the performance of the PPO trained policy. Line type (solid or dashed) denotes the terrain size ( $42^2$  or  $84^2$ ).

robotics problem of exploration/coverage of a completely unknown region into a reinforcement learning setup that can be tackled by a wide range of off-the-shelf, model-free RL algorithms. An essential feature of the whole solution is that trained policies can be straightforwardly applied to real-life robotic platforms without being trained/tuned to the robot’s dynamics. To achieve that, the same level of information abstraction between the robotic system and the MarsExplorer is required. A detailed experimental evaluation was also conducted and presented. 4 state-of-the-art RL algorithms, namely A3C, PPO, Rainbow, and SAC, were evaluated in a challenging version of MarsExplorer, and their training results were also compared with the human-level performance for the task at hand. PPO algorithm achieved the best score, which was also 85.8% of the human-level

performance. Then, the PPO algorithm was utilized to study the effect of the multi-dimensional difficulty vector changes in the overall performance. The visualization of the paths for all these difficulty levels revealed a quite important trait. The PPO learned policy has learned to perform a Hilbert curve with the extra ability to avoid any encountered obstacle. Lastly, a scalability study clearly indicates the ability of RL approaches to be extended in larger terrains, where the achieved performance is validated with non-learning, frontier-based explorations strategies.

Table 2: PPO Hyperparameters

Parameter	Value	Comments
$\gamma$	0.95	Discount factor of the MDP
$\lambda$	5e-5	Learning rate
Critic	True	Used a critic as a baseline
GAE $l$	0.95	GAE (lambda) parameter
KL coeff	0.2	Initial coefficient for KL divergence
Clip	0.3	PPO clip parameter

Table 3: DQN-Rainbow Hyperparameters

Parameter	Value	Comments
$\gamma$	0.95	Discount factor of the MDP
$\lambda$	5e-4	Learning rate
Noisy Net	True	Used a noisy network
Noisy $\sigma$	0.5	initial value of noisy nets
Dueling Net	True	Used dueling DQN
Double dueling	True	Used double DQN
$\epsilon$ -greedy	[1.0, 0.02]	Epsilon greedy for exploration.
Buffer size	50000	Size of the replay buffer
Prioritized Replay	True	Prioritized replay buffer used

Table 4: SAC Hyperparameters

Parameter	Value	Comments
$\gamma$	0.95	Discount factor of the MDP
$\lambda$	3e-4	Learning rate
Twin Q	True	Use two Q-networks
Q hidden	[256, 256]	Hidden layer activation
Policy hidden	[256, 256]	Hidden layer activation
Buffer size	1e6	Size of the replay buffer
Prioritized Replay	True	Prioritized replay buffer used

Table 5: A3C Hyperparameters

Parameter	Value	Comments
$\gamma$	0.95	Discount factor of the MDP
$\lambda$	1e-4	Learning rate
Critic	True	Used a critic as a baseline
GAE	True	General Advantage Estimation
GAE $l$	0.99	GAE(lambda) parameter
Value loss	0.5	Value Function Loss coefficient
Entropy coef	0.01	Entropy coefficient

## Acknowledgments

This project has received funding from the European Commission under the European Union’s Horizon 2020 research and innovation programme under grant agreement no 833464 (CREST). Also, we gratefully acknowledge the support of NVIDIA Corporation with the donation of GPUs used for this research.

## References

- [Baker *et al.* 2019] Bowen Baker, Ingmar Kanitscheider, Todor Markov, Yi Wu, Glenn Powell, Bob McGrew, and Igor Mordatch. Emergent tool use from multi-agent autototocurricula. *arXiv preprint arXiv:1909.07528*, 2019.
- [Basilico and Amigoni 2011] Nicola Basilico and Francesco Amigoni. Exploration strategies based on multi-criteria decision making for searching environments in rescue operations. *Autonomous Robots*, 31(4):401–417, 2011.
- [Bellemare *et al.* 2013] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, Jun 2013.
- [Brockman *et al.* 2016] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.
- [Burgard *et al.* 2000] Wolfram Burgard, Mark Moors, Dieter Fox, Reid Simmons, and Sebastian Thrun. Collaborative multi-robot exploration. In *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065)*, volume 1, pages 476–481. IEEE, 2000.
- [Cobbe *et al.* 2020] Karl Cobbe, Chris Hesse, Jacob Hilton, and John Schulman. Leveraging procedural generation to benchmark reinforcement learning. In *International conference on machine learning*, pages 2048–2056. PMLR, 2020.
- [Dhariwal *et al.* 2017] Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, Yuhuai Wu, and Peter Zhokhov. Openai baselines, 2017.
- [Gray *et al.* 2003] Lawrence Gray, A New, et al. A mathematician looks at wolfram’s new kind of science. In *Notices of the American Mathematical Society* 50 (2)(2003) 200–211. Citeseer, 2003.
- [Haarnoja *et al.* 2018] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv preprint arXiv:1801.01290*, 2018.
- [Hessel *et al.* 2018] Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.
- [Kapoutsis *et al.* 2016] Athanasios Ch Kapoutsis, Savvas A Chatzichristofis, Lefteris Doitsidis, Joao Borges de Sousa, Jose Pinto, Jose Braga, and Elias B Kosmatopoulos. Real-time adaptive multi-robot exploration with application to underwater map construction. *Autonomous robots*, 40(6):987–1015, 2016.
- [Kapoutsis *et al.* 2017] Athanasios Ch Kapoutsis, Savvas A Chatzichristofis, and Elias B Kosmatopoulos. Darp: divide areas algorithm for optimal multi-robot coverage path planning. *Journal of Intelligent & Robotic Systems*, 86(3-4):663–680, 2017.
- [Kapoutsis *et al.* 2019] Athanasios Ch Kapoutsis, Savvas A Chatzichristofis, and Elias B Kosmatopoulos. A distributed, plug-n-play algorithm for multi-robot applications with a priori non-computable objective functions. *The International Journal of Robotics Research*, 38(7):813–832, 2019.
- [Koutras *et al.* 2020] Dimitrios I Koutras, Athanasios Ch Kapoutsis, and Elias B Kosmatopoulos. Autonomous and cooperative design of the monitor positions for a team of uavs to maximize the quantity and quality of detected objects. *IEEE Robotics and Automation Letters*, 5(3):4986–4993, 2020.
- [Lei *et al.* 2018] Xiaoyun Lei, Zhian Zhang, and Peifang Dong. Dynamic path planning of unknown environment based on deep reinforcement learning. *Journal of Robotics*, 2018, 2018.
- [Liang *et al.* 2017] Eric Liang, Richard Liaw, Robert Nishihara, Philipp Moritz, Roy Fox, Joseph Gonzalez, Ken Goldberg, and Ion Stoica. Ray rllib: A composable and scalable reinforcement learning library. *arXiv preprint arXiv:1712.09381*, page 85, 2017.

- [Liang *et al.*2018] Eric Liang, Richard Liaw, Robert Nishihara, Philipp Moritz, Roy Fox, Ken Goldberg, Joseph Gonzalez, Michael Jordan, and Ion Stoica. Rllib: Abstractions for distributed reinforcement learning. In *International Conference on Machine Learning*, pages 3053–3062. PMLR, 2018.
- [Lopez *et al.*2019] Nestor Gonzalez Lopez, Yue Leire Erro Nuin, Elias Barba Moral, Lander Usategui San Juan, Alejandro Solano Rueda, Victor Mayoral Vilches, and Risto Kojcev. gym-gazebo2, a toolkit for reinforcement learning using ROS 2 and gazebo. *CoRR*, abs/1903.06278, 2019.
- [Luis *et al.*2021] Samuel Yanes Luis, Daniel Gutiérrez Reina, and Sergio L Toral Marín. A multiagent deep reinforcement learning approach for path planning in autonomous surface vehicles: The ypacarac-lake patrolling case. *IEEE Access*, 2021.
- [Mnih *et al.*2015a] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.
- [Mnih *et al.*2015b] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharmashan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, February 2015.
- [Mnih *et al.*2016] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937, 2016.
- [Niroui *et al.*2019] Farzad Niroui, Kaicheng Zhang, Zendai Kashino, and Goldie Nejat. Deep reinforcement learning robot for search and rescue applications: Exploration in unknown cluttered environments. *IEEE Robotics and Automation Letters*, 4(2):610–617, 2019.
- [Palacios-Gasós *et al.*2016] José Manuel Palacios-Gasós, Eduardo Montijano, Carlos Sagüés, and Sergio Llorente. Distributed coverage estimation and control for multi-robot persistent tasks. *IEEE transactions on Robotics*, 32(6):1444–1460, 2016.
- [Popov *et al.*2017] Iyavlo Popov, Nicolas Heess, Timothy Lillicrap, Roland Hafner, Gabriel Barth-Maron, Matej Vecerik, Thomas Lampe, Yuval Tassa, Tom Erez, and Martin Riedmiller. Data-efficient deep reinforcement learning for dexterous manipulation. *arXiv preprint arXiv:1704.03073*, 2017.
- [Renzaglia *et al.*2019] Alessandro Renzaglia, Jilles Diban-goye, Vincent Le Doze, and Olivier Simonin. Combining stochastic optimization and frontiers for aerial multi-robot exploration of 3d terrains. In *IROS*, pages 4121–4126, 2019.
- [Sadat *et al.*2015] Seyed Abbas Sadat, Jens Wawerla, and Richard Vaughan. Fractal trajectories for online non-uniform aerial coverage. In *2015 IEEE international conference on robotics and automation (ICRA)*, pages 2971–2976. IEEE, 2015.
- [Schulman *et al.*2017] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.
- [Shrestha *et al.*2019] Rakesh Shrestha, Fei-Peng Tian, Wei Feng, Ping Tan, and Richard Vaughan. Learned map prediction for enhanced mobile robot exploration. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 1197–1204. IEEE, 2019.
- [Silver *et al.*2016] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.
- [Simmons *et al.*2000] Reid Simmons, David Apfelbaum, Wolfram Burgard, Dieter Fox, Mark Moors, Sebastian Thrun, and Håkan Younes. Coordination for multi-robot exploration and mapping. In *Aaai/Iaai*, pages 852–858, 2000.
- [Smith *et al.*2020] Marshall Smith, Douglas Craig, Nicole Herrmann, Erin Mahoney, Jonathan Krezel, Nate McIntyre, and Kandyce Goodliff. The artemis program: An overview of nasa’s activities to return humans to the moon. In *2020 IEEE Aerospace Conference*, pages 1–10. IEEE, 2020.
- [Wen *et al.*2020] Shuhuan Wen, Yanfang Zhao, Xiao Yuan, Zongtao Wang, Dan Zhang, and Luigi Manfredi. Path planning for active slam based on deep reinforcement learning under unknown environments. *Intelligent Service Robotics*, pages 1–10, 2020.
- [Witze *et al.*2020] Alexandra Witze, Smriti Mallapaty, and Elizabeth Gibney. All aboard to mars, 2020.
- [Zamora *et al.*2016] Iker Zamora, Nestor Gonzalez Lopez, Victor Mayoral Vilches, and Alejandro Hernandez Cordero. Extending the openai gym for robotics: a toolkit for reinforcement learning using ros and gazebo. *arXiv preprint arXiv:1608.05742*, 2016.
- [Zhang *et al.*2018] Kaichena Zhang, Farzad Niroui, Maurizio Ficocelli, and Goldie Nejat. Robot navigation of environments with unknown rough terrain using deep reinforcement learning. In *2018 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, pages 1–7. IEEE, 2018.



# Extended Task and Motion Planning of Long-horizon Robot Manipulation

Tianyu Ren, Georgia Chalvatzaki, Jan Peters

Department of Computer Science, Technische Universität Darmstadt

Email: tianyu@robot-learning.de

**Abstract**—Task and Motion Planning (TAMP) requires the integration of symbolic reasoning with metric motion planning that accounts for the robot’s actions’ geometric feasibility. This hierarchical structure inevitably prevents the symbolic planners from accessing the environment’s low-level geometric description, vital to the problem’s solution. Most TAMP approaches fail to provide feasible solutions when there is missing knowledge about the environment at the symbolic level. The incapability of devising alternative high-level plans leads existing planners to a dead end. We propose a novel approach for decision-making on extended decision spaces over plan skeletons and action parameters. We integrate top-k planning for constructing an explicit skeleton space, where a skeleton planner generates a variety of candidate skeleton plans. Moreover, we effectively combine this skeleton space with the resultant motion parameter spaces into a single *extended* decision space. Accordingly, we use Monte-Carlo Tree Search (MCTS) to ensure an exploration-exploitation balance at each decision node and optimize globally to produce minimum-cost solutions. The proposed seamless combination of symbolic top-k planning with streams, with the proved optimality of MCTS, leads to a powerful planning algorithm that can handle the combinatorial complexity of long-horizon manipulation tasks. We empirically evaluate our proposed algorithm in challenging manipulation tasks with different domains that require multi-stage decisions and show how our method can overcome dead-ends through its effective alternate plans compared to its most competitive baseline method.

## I. INTRODUCTION

The intelligent robots of the future should perform multiple tasks in unstructured environments, like houses, hospitals, etc. Among the most critical features in robotics is the need for reasoning and acting for achieving multi-stage long-horizon tasks that require manipulation and mobility. Traditional approaches considered long-horizon manipulation as a strict decomposition of sub-tasks, either hard-coded or addressed as individual sub-problems to be solved independently. Inspired by advances in symbolic *Artificial Intelligence (AI) planning*, the previous strict hierarchical approach is now getting automated by a single solution for *Task and Motion Planning (TAMP)* [1, 2]. With a typical TAMP solver, in the high level, a task planner reasons over a sequence of symbolic actions for reaching a symbolic goal state; in the low level, a set of motion planners search for metric decisions to make each action geometrically feasible w.r.t. the environment.

TAMP hierarchical structure makes it suitable for solving long-horizon tasks; however, it brings some major issues in coordinating the two levels of symbolic reasoning and geometric planning. Since the task planner is mostly uninformed about

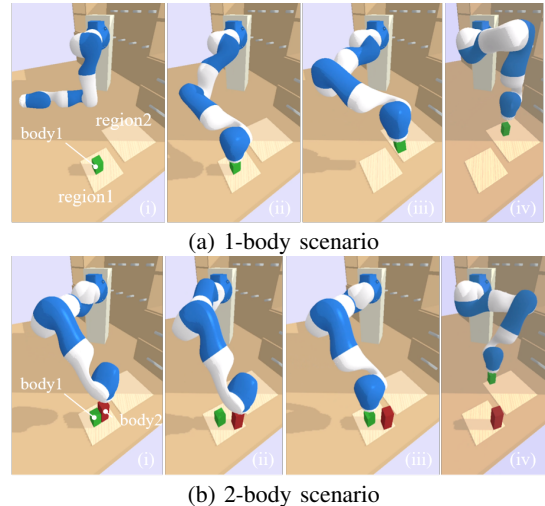


Fig. 1: Example task: transportation of body1 from region1 to region2. Bodies can only be grasped from the top. Though the 1-body scenario is trivial to most TAMP algorithms, the 2-body scenario is more challenging: the taller red body must be relocated before the green one can be reached without collisions.

detailed constraints of the environment, it tends to generate symbolic plans that are geometrically infeasible. Existing TAMP algorithms generally start with only one symbolic plan that is considered “optimal”. Sometimes such a plan may be enough (Fig. 1a). However, in more challenging scenarios, e.g., Fig. 1b, this single plan fails quickly. This issue is commonly described as an *incomplete domain description* [3, 4] in the sense that the task planner does not have enough domain knowledge to generate absolutely correct results.

Like AI planning, TAMP is an intrinsic offline process that relies heavily on the associated domain description. Research has delivered significant advancements, especially with the introduction of integrated TAMP methods [1, 2, 5, 6, 7], however particular domain specifications and assumptions hinder their application in real-world *general* settings.

This paper proposes a novel framework for general-purpose TAMP with extended decision spaces (eTAMP). Our proposed method generates diverse alternate symbolic plans (skeletons) for an extended decision space, optimizing both the skeleton selection and the actions’ parameters’ concrete bindings. We propose using a top-k skeleton planner to produce diverse skeletons, guaranteeing that no better solution exists under a current domain description [8]. Moreover, we augment the skeleton planner with streams [6, 9], a sampling procedure that



uses black-box conditional generators of action parameters in the task planning domain, as additional operators for symbolic AI planning. Streams address the hybrid discrete-continuous planning domains like those in PDDLStream [9, 6]. We integrate the decision over skeletons as an "additional stream" for sampling alternate skeletons in the TAMP's symbolic subspace. Notably, we propose using Monte-Carlo Tree Search (MCTS) to solve this stochastic decision-making problem over skeletons and concrete bindings of the action parameters. The main contribution of this work is twofold:

- we reformulate the TAMP framework by transforming the problem of skeleton planning with incomplete domain description into a series of generic planning problems that are tractable to any off-the-shelf top-k planner;
- we propose a tree-structured search algorithm for solving the resultant stochastic decision-making problem in the extended decision space.

We evaluate our proposed framework on challenging robotic tasks defined in various domains and long-horizon tasks with increasing difficulty, using both a 7-degrees-of-freedom (dof) manipulator and a 10-dof mobile manipulator robot. Our empirical results prove the ability of the proposed eTAMP method to find feasible plans in challenging manipulation scenarios by producing alternate skeletons and the decision of motion parameters of the actions compared to the state-of-the-art, thus taking one more step towards the creation of general-purpose robotic TAMPs.

## II. RELATED WORK

Incomplete domain specification is ubiquitous in TAMP applications. This incomplete domain fails to capture several environmental constraints crucial to the feasibility of the generated skeletons. A robust TAMP algorithm should search for alternate skeletons. Recent methods can discover new skeletons using replanning or reattempting mechanisms [9, 6, 10], and those are, in the best case, patched versions of the original infeasible plans. Therefore, most of the space of alternate skeletons will remain unexplored.

In particular, for robot TAMP, the developed algorithmic solutions vouched for performance instead of completeness and generality of the proposed planners [5]. A comprehensive review over integrated TAMP approaches can be found in [7]. Research for TAMP has been frequently approached as an optimization problem using logic geometric programming [11], or multi-modal motion planning with motion-modes switches for various tasks [12]. However, these methods are designed for particular manipulation problems and are not applicable across different domains.

We are concerned with TAMP problem formulations that use the Planning Domain Definition Language (PDDL) [13] for the task domain description. PDDL is used to symbolize the original planning domain to individual actions with pre-conditions and effects. Early works for TAMP used semantic attachments in PDDL for querying motion planners over the feasibility of task actions [14]. However, this method requires a pre-specification of sets of discrete parameters like object

TABLE I: The stream set  $Streams_0$  of the transportation problem.

Stream	Description
Sample-pose( $?body, ?region$ ) $\rightarrow ?pose$	Generate a collision-free $?pose$ for $?body$ on $?region$ .
Plan-motion ( $?body, ?pose\_from, ?pose\_to$ ) $\rightarrow ?traj$	Generate a collision-free $?traj$ that transports $?body$ from $?pose\_from$ to $?pose\_to$ .

poses, grasps, and robot configurations. Several methods consider finding constraint-satisfying states as in [5] to search for feasible task plans given some geometric constraints. [15] directly samples feasible regions for actions using conditional samplers.

The need for alternate skeletons remains an open research problem for generalization over different domains. Notably, in [10] the authors try to explicitly generate alternate skeletons through incremental solving. If the motion planner fails to refine a skeleton, they add additional feasibility constraints to the task planner. Similarly, in [2], the authors use an interface layer that generates logical facts that capture the failure of current skeletons and update the high-level state description. This heuristic treatment makes this decision process highly sub-optimal for new domains, ending up performing similarly to reattempting methods.

The robustness of TAMP and the quality of its solution will be improved by an extended optimal search in the skeleton space in addition to the search in the action parameter space. Top-k planning is one way of obtaining such a set, by finding a group of diverse solutions of size  $k$ . With a variety of candidate skeletons, it is possible to make a detour to avoid infeasible selections. Based on a Fast Downward [16, 17], [18] presents a complete top-k planner, the SYM-K, that scales efficiently to large sizes of  $k$ .

For searching for concrete bindings in a given decision space, MCTS methods (see App. ??), such as UCT [19, 20], provides a probabilistically optimal solution by performing a sample-based tree search. [21] uses Voronoi partitioning with MCTS for achieving higher efficiency in robot planning, however, it is restricted to deterministic planning problems and therefore are not applicable to TAMP problems with stochastic transition functions.

## III. PRELIMINARIES

*Robotic motion planning and AI planning.* Motion planning demands a continuous *state-space formulation*, and sample-based methods have satisfactorily solved it (e.g., the rapidly exploring dense tree family [22]). Within AI planning, *logic-based formulations* (e.g., STRIPS and PDDL) have been conveniently used to compactly represent enormous discrete state spaces and produce outputs that logically explain the steps involved in arriving at some goal state.

*PDDLStream.* A PDDL task is a tuple  $T = \langle Objects, S, G, Actions \rangle$ . It is about finding a sequence of actions from set *Actions* that when applied in succession will transform the world from the initial state  $S$  into one in which all literals of  $G$  are true. *Objects* is a finite set of

objects associated to the task. Let us consider the planning task of Fig. 1a. The initial state, goal state, and available actions for this scenario are

$$S_0 = \{ \text{On}(\text{body1}, \text{region1}) \} \quad (1)$$

$$G_0 = \{ \text{On}(\text{body1}, \text{region2}) \} \quad (2)$$

$$\text{Actions}_0 = \{ \text{Pick-place}(\text{?body}, \text{?from}, \text{?to}, \text{?traj}) \} \quad (3)$$

PDDLStream [6] attempts to increase the universality of PDDL by incorporating *streams* into PDDL operators in addition to actions. A PDDLStream task can be represented as a tuple  $T_{\text{stream}} = \langle \text{Objects}, S, G, \text{Actions}, \text{Streams} \rangle$ . Streams give out new objects during planning as black-box generators of any kind. This kind of operators is crucial to a *general-purpose TAMP system* for easily integrating state-of-the-art sub-task solvers. Let us assume the streams for the transportation problem, as shown in Table I. Then, the planner can start with a set of *undemanding* objects

$$\text{Objects}_0 = \{ \text{body1}, \text{pose11}, \text{region1}, \text{region2} \} \quad (4)$$

where *pose11* is the initial pose of *body1* in *region1*, and it is easily accessed. A skeleton plan  $\pi_s$  composed of both actions and streams can be found as

$$\begin{aligned} & \langle \text{Sample}(\text{body1}, \text{region2}) \rightarrow \# \text{pose12}, \\ & \text{Plan-motion}(\text{body1}, \text{pose11}, \# \text{pose12}) \rightarrow \# \text{traj112}, \\ & \text{Pick-place}(\text{body1}, \text{region1}, \text{region2}, \# \text{traj112}) \rangle. \end{aligned} \quad (5)$$

where the objects generated by streams are marked by  $\#$  and they demand further *bindings* to concrete values before the plan can be executed by the robot. These symbolized outputs are *optimistic* in the sense that they may not ever find their concrete counterparts under the search constraints. Thus, not all skeletons can be instantiated to concrete plans. The optimistic object introduced by PDDLStream is essential to the decomposition of the original manipulation planning problem into sub-problems of symbolic task planning and metric motion planning [23].

#### IV. TOP-K SKELETON PLANNING

Most TAMP systems satisfy their symbolic goals based on a single “best” skeleton. However, such a skeleton is usually vulnerable to the partially described environment during task planning. Let  $P_T$  be the set of all symbolic plans (possibly infinite) for a planning task  $T$ . The objective of top-k planning is to determine a set of  $k$  different plans  $\{\pi_1, \pi_2, \dots, \pi_k\} = P \subseteq P_T$  with the lowest costs for a given AI planning task [18]. We call a top-k algorithm *complete* iff it could find the set of all plans as required. A standard top-k planner can be described as  $P = \text{TOP-K}(\text{Objects}, S, G, \text{Actions}, k)$ .

Skeleton planning starts with (i) optimistic enrichment of the initial object set and the initial state set such that (ii) a variety of *action plans* can be found by a top-k symbolic planner. Streams are ignored during top-k planning since their presences are not accountable to the diversity of the final plans. (iii) Next, a skeleton is recovered from each action plan by retracing all supportive streams. And finally we have top-k skeletons produced.

#### A. Graph expansion for optimistic object generation

---

##### Algorithm 1: OPTMS-EXPD

---

**Input:** *Objects, S, Streams, level*  
 $\# \text{Objects} = \text{copy}(\text{Objects})$   
 $\# S = \text{copy}(S)$   
**for**  $i \in [1, 2, \dots, \text{level}]$  **do**  
     $\# \text{Objects}, \# S =$   
    NEXT-LAYER( $\# \text{Objects}, S, \text{Streams}$ )  
**return**  $\# \text{Objects}, \# S$

---

We use forward graph expansion with all streams, denoted  $S_0$ , as operators to enrich the initial object set while limiting the level of the planning graph. In the graph expansion, the first layer consists of the original objects (e.g., (4)), and the second layer is made-up of applicable streams whose preconditions are satisfied by the previous layer. Unique optimistic values are generated by the streams of the second layer, and are listed in the third layer. The expansion goes on until the maximum level is reached. This optimistic expansion is summarized in Alg. 1, where *Objects* and  $\# \text{Objects}$  denote the initial object set and the optimistically expanded set, respectively. *Streams* are the set of all available streams in the task, e.g., Table I. As new objects are generated, the original initial set  $S$  will be expanded to  $\# S$ . The operation NEXT-LAYER expands the graph by one level. For example, after applying Alg. 1 to  $\text{Objects}_0$  in (4) with  $\text{level} = 2$ , we get

$$\begin{aligned} \# \text{Objects}_0 = \{ & \text{body1}, \text{pose11}, \text{region1}, \text{region2}, \\ & \# \text{pose11a}, \# \text{pose12a}, \# \text{traj111a}, \\ & \# \text{traj112a}, \# \text{traj112b}, \# \text{pose11b}, \\ & \# \text{pose12b} \}. \end{aligned} \quad (6)$$

#### B. Searching top-k action plans

Here we are only interested in the diversity of the actions’ arrangement. Starting from the expanded sets,  $\# \text{Objects}$  and  $\# S$ , we formulate the top-k planning problem with only actions as PDDL operators

$$P_a = \text{TOP-K}(\# \text{Objects}, \# S, G, \text{Actions}, k) \quad (7)$$

where  $G$  and *Actions* are with the original problem description as in (2) and (3), and  $k = |P_a|$ . The planning subroutine TOP-K can be any complete and sound symbolic top-k planner. Here we choose the SYM-K algorithm [18].

#### C. Building top-k skeletons

Now we need to recovery the top-k skeletons  $P_s$  from the action plans  $P_a$ . A skeleton plan  $\pi_s$  must build on only undemanding PDDL conditions, e.g.,  $S_0$  from (1) and  $\text{Objects}_0$  from (4). For a action plan  $\pi_a$ , we retrace streams that are responsible for the optimistic objects  $\# \text{Objects}_0 \setminus \text{Objects}_0$  consumed by the actions of  $\pi_a$ , and include them into a

skeleton plan. We formalize this stream-retracing procedure as another symbolic planning problem and solve it as follows

$$\pi_s = \text{TOP-K}(\text{Objects}, S, G_s(\pi_a, G), \text{Actions} \cup \text{Streams}, 1) \quad (8)$$

$\text{Objects}$ ,  $S$ , and  $G$  are from the original task description.  $\text{Actions}$  and  $\text{Streams}$  constitute the operator set of this planning problem. Informally, by setting  $k = 1$ , the top-k algorithm reduces to a classical PDDL planner. To ensure that all member actions have the same presence in  $\pi_s$ , as in  $\pi_a$ , we must add extra constraints to the goal state. Here, we make the goal state a function of  $\pi_a$ . Assuming  $\pi_a = \langle a_1, a_2, \dots, a_n \rangle$ , we have

$$G_s(\pi_a, G) = \{(a_1 \prec a_2), \dots, (a_{n-1} \prec a_n)\} \cup G, \quad (9)$$

where  $(a_1 \prec a_2)$  denotes a literal asserting that  $a_1$  must be the predecessor of  $a_2$ .

#### D. The top-k skeleton planning algorithm

An overview of our proposed skeleton planning algorithm is shown in Alg. 2. The inputs of the algorithm include the original initial state  $S$ , goal state  $G$ , available  $\text{Streams}$  and  $\text{Actions}$ , and a desired number of skeletons  $k$ . The output  $P_s$  is the list of the top  $k$  skeletons, if it is found under a preset level limit  $\text{max-level}$  in the optimistic expansion of Alg. 1. The size of  $\# \text{Objects}$  ramps up quickly with increasing  $\text{level}$ . To address this, in Alg. 2 we regulate the number of optimistic objects by progressively increasing the  $\text{level}$  during the searching process.

*Property 1:* Probabilistic completeness in skeleton planning. Given a complete top-k planner, any potentially feasible skeleton will be contained by the result set of Alg. 2 as  $k$  goes to infinity.

We choose SYM-K [18] as our planning subroutine in this study. Due to its proved completeness and soundness, the property mentioned previously is well maintained. For the simple 2-body transportation task of Fig. 1b, the robot should relocate the taller red body, for accessing and relocating the green one. Applying our proposed Alg. 2 to the task  $T_{\text{stream}, 1\text{-body}} = \langle (4), (1), (2), (3), \text{Table I} \rangle$ , with (8) we get a feasible skeleton as:

$$\begin{aligned} &\langle \text{Sample-pose}(\text{body2}, \text{region1}) \rightarrow \# \text{pose21a}, \\ &\text{Plan-motion}(\text{body2}, \text{pose21}, \# \text{pose21a}) \rightarrow \# \text{traj211a}, \\ &\text{Pick-place}(\text{body2}, \text{region1}, \text{region1}, \# \text{traj211a}), \\ &\text{Sample-pose}(\text{body1}, \text{region2}) \rightarrow \# \text{pose12a}, \\ &\text{Plan-motion}(\text{body1}, \text{pose11}, \# \text{pose12a}) \rightarrow \# \text{traj112a}, \\ &\text{Pick-place}(\text{body1}, \text{region1}, \text{region2}, \# \text{traj112a}) \rangle. \end{aligned} \quad (10)$$

#### V. TREE SEARCH IN THE EXTENDED DECISION SPACE

A skeleton  $\pi_s$  describes a symbolically feasible path to the goal state. Its geometric feasibility must be evaluated along with the sequential bindings of its optimistic objects to concrete values. For example, there are 4 bindings to be

#### Algorithm 2: TOP-K-SKELETON

---

**Input:**  $\text{Objs}, S, G, \text{Actions}, \text{Streams}, k$   
**for**  $\text{level} \in [0, 1, \dots, \text{max-level}]$  **do**  
     $\# \text{Objs}, \# S =$   
    OPTMS-EXPD( $\text{Objs}, S, \text{Streams}, \text{level}$ ), see Alg. 1  
     $P_a = \text{TOP-K}(\# \text{Objs}, \# S, G, \text{Actions}, k)$ , see (7)  
    **if**  $|P_a| < k$  **then**  
        **continue for**  
     $P_s = \{\}$   
    **for**  $\pi_a \in P_a$  **do**  
         $\text{Operators} = \text{Actions} \cup \text{Streams}$   
         $\# G = G_s(\pi_a, G)$ , see (9)  
         $\pi_s = \text{TOP-K}(\text{Objs}, S, \# G, \text{Operators}, 1)$ , see (8)  
         $P_s \leftarrow \{\pi_s\} \cup P_s$   
**return**  $P_s$

---

TABLE II: Nodes of the binding tree model of the skeleton in (10)

Node	Member operators
decision1	Sample-pose(body2,region1) $\rightarrow$ #pose21a
transition1	Plan-motion(body2,pose21,#pose21a) $\rightarrow$ #traj211a Pick-place(body2,region1,region1,#traj211a)
decision2	Sample-pose(body1,region2) $\rightarrow$ #pose12a
transition2	Plan-motion(body1,pose11,#pose12a) $\rightarrow$ #traj112a Pick-place(body1,region1,region2,#traj112a)

decided in (10). As the decisions of the bindings are causally related to each other, and they extend to large sizes as the task horizon increases, we propose using an MCTS-like approach that can systematically explore this broad decision space.

#### A. The extended decision tree

We pose the decision over multiple skeletons as a multi-armed bandit problem, and it can be satisfactorily solved by UCB1 as shown in (??). In particular, we use UCT to make optimal sequential decisions for the binding list of each selected skeleton. In practice, not all optimistic objects of a skeleton necessarily belong to its binding list. Following [24], we can group the operators of (10) into two distinct and alternating types of nodes (see Table II and Fig. 2), the *transition* and the *decision* nodes. Operators like "Plan-motion" (with random outputs by RRT planners [22]) and "Pick-place" (with possibly positioning error—not implemented in this work) in (10) are modeled as stochastic transition systems. On the other hand, "Sample-pose" is completely depended on the task, and considers for example decision over different placement poses  $\text{pose21a}$  and  $\text{pose12a}$ , hence making it a *decision node*.

For simplicity, we pack successive transitional operators into a single transition node in Table II, e.g., "Plan-motion" and "Pick-place" belong to the same transition node. Finally, we can formulate the skeleton-selection and the binding search with a single model, based on the observation that skeleton-selection is just an extended decision node at the root of a decision tree. As a result, an extended decision tree can be built, as shown Fig. 2.

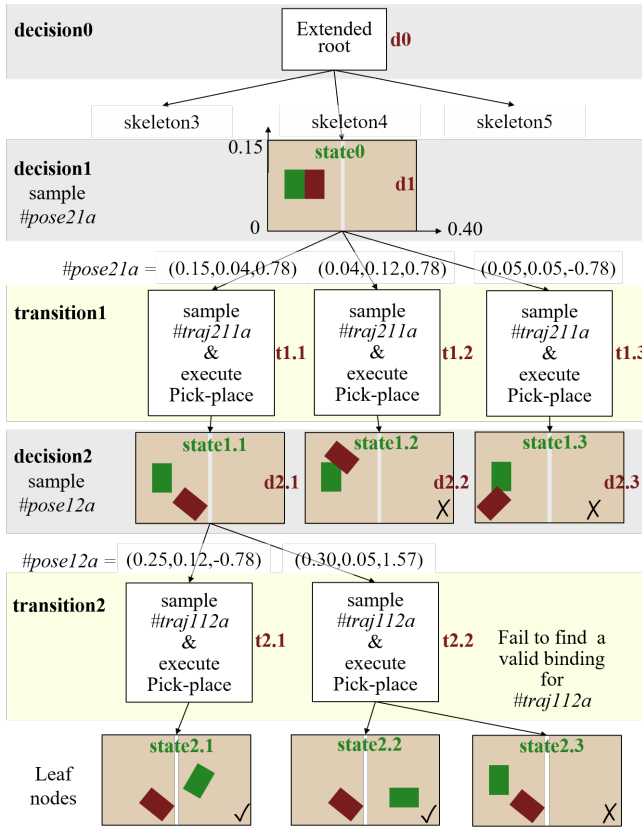


Fig. 2: An example of an extended decision tree for solving the 2-body transportation task of Fig. 1b

### B. Binding search in the extended decision tree

Starting from the extended root, a robot must make sequential decisions along the tree until a terminal state is encountered. A *terminal state* indicates that the state of the current node is infeasible to the task, and binding search must stop here (e.g., state1.2 and state1.3 in Fig. 2). Then it will receive a reward  $r \in \mathbb{R}$ . The robot's objective is to find a sequence of decisions with a planning horizon  $H_i$  that maximizes the final reward.  $H_i = 1 + \text{depth}_i$  where  $\text{depth}_i$  is the number of decision nodes inside the selected skeleton  $\pi_{s,i}$ . In our study, we use a reward function defined as

$$r = 0.1 \left( \frac{\text{depth}_{\text{end}}}{\text{depth}_i} + \frac{1}{\text{motionCost}_{\text{end}}} \right) + r_{\text{success}} \quad (11)$$

where  $\text{depth}_{\text{end}}$  is the depth in the decision tree in which the current node is terminated, and  $\text{motionCost}_{\text{end}}$  is proportional to the swept volume of the robot from its initial state to the terminal state. The first term in (11) is a normalized depth that encourages the robot to avoid branches where bindings have already failed earlier. The second term makes the robot prefer branches with motions that have less occupation in the workspace. For the third term,  $r_{\text{success}} = 1$  when all bindings are successfully found, otherwise  $r_{\text{success}} = 0$ . This reward design is effective for solving an array of robot manipulation problems, as we show in Sec. VI, and it can be specialized for better performance in specific tasks.

By now, we have a classical finite-horizon stochastic optimal search problem, and it can be described as a Markov Decision Process (MDP). With the UCT formulation, episodic MDP is repeated during the search loop, and information from the previous episodes is used for reaching optimal decisions in the subsequent episodes. The information, in this case, refers to the value  $V$  and the visit number  $M$  of a tree node. Since we have continuous bindings to decide on (e.g.,  $\# \text{pose12a}$  in Fig. 2), we employ a UCT variant with progressive widening (PW) techniques [25, 24, 26]. The basic idea is to limit the number of visits for existing nodes artificially. When the value of the existing nodes is estimated sufficiently well, new nodes will be created to explore the unreached regions of the decision spaces. The original PW criterion, however, assumes that the new decisions are always available, while it is not the case when decisions at a node are discrete and enumerable. To counteract this restriction, we define a new PW law with Alg. 3, that supports streams with both continuous and discrete outputs for versatile robotic applications. The argument

---

#### Algorithm 3: EXPAND-TEST

---

```

Input: node
if node is a decision node then
     $s = \text{streamFromNode}(\text{node})$ 
    if  $s.\text{output}$  is continuous then
        return PW-LAW(node)
    else
        if  $s.\text{outputSpace} \subseteq \{\text{node}.\text{decisionHistory}\}$  then
            return False
        else
            return True
else
    return PW-LAW(node)

```

---



---

#### Algorithm 4: CHILD-SELECT

---

```

Input: node
if node is a decision node then
     $\text{child\_node} = \text{UCB-SELECT}(\text{node})$ 
else
     $\text{child\_node} = \text{LEAST-SELECT}(\text{node})$ 
return child_node

```

---



---

#### Algorithm 5: NEW-CHILD

---

```

Input: node
if node is a decision node then
     $\text{new\_node} = \text{NEW-DECISION}(\text{node})$ 
else
     $\text{new\_node} = \text{NEW-TRANSITION}(\text{node})$ 
    node.addChild(new_node)
return new_node

```

---

False = EXPAND-TEST(node) means that new edges cannot

be created from *node* at the time. In this case, an existing edge connecting to an established child of *node* must be selected. For *decision nodes*, we use the UCB-SELECT(*node*) strategy to select the child of *node* with the maximum UCB score. For *transition nodes*, the strategy of LEAST-SELECT(*node*) is used to simply select the least-visited child of *node*. The overall child selection strategy is in Alg. 4.

If  $\text{True} = \text{EXPAND-TEST}(\text{node})$ , new edges leading from *node* to a new child node must be created. For *decision nodes*, we use the NEW-DECISION(*node*) strategy to sample a new decision and create the resultant child node. With this strategy, *node* with discrete stream outputs will randomly select an unvisited decision; *node* with continuous stream outputs will use Voronoi sampling to choose the next concrete binding for rapid exploration of the decision space. For the *transition node*, we define a subroutine called NEW-TRANSITION(*node*) that directs the environment to a new state on which a new child node is built. At last, we can summarize our extended tree search algorithm in Alg. 6. Its inputs include the top-k skeleton set  $P_s$  generated by Alg. 2 and a user-defined time budget  $t_{ts}$ . The output of Alg. 6 is a single concrete plan  $\pi_c$  that is optimized under given  $t_{ts}$ . On RECEIVE-VISIT(*node*), *node*

---

**Algorithm 6: EXTENDED-TREE-SEARCH**

---

**Input:**  $P_s, t_{ts}$   
 $\text{extended\_root} = \text{buildExtendedRoot}(P_s)$   
 $\text{node} \leftarrow \text{extended\_root}$   
**while**  $\text{timeCost}() < t_{ts}$  **do**  
  **while** *node* is not terminated **do**  
    RECEIVE-VISIT(*node*)  
    **if**  $\text{EXPAND-TEST}(\text{node})$  **then**  
       $\text{node} \leftarrow \text{NEW-CHILD}(\text{node})$ , see Alg. 5  
    **else**  
       $\text{node} \leftarrow \text{CHILD-SELECT}(\text{node})$ , see Alg. 4  
   $\text{backupRewardFrom}(\text{node})$   
   $\text{node} \leftarrow \text{extended\_root}$   
 $\pi_c = \text{highestValueBranchFrom}(\text{extended\_root})$   
**return**  $\pi_c$

---



---

**Algorithm 7: eTAMP**

---

**Input:**  $T_{\text{stream}}, k, t_b$   
 $P_s = \text{TOP-K-SKELETON}(T_{\text{stream}}, k)$ , see Alg. 2  
 $t_{ts} = t_b - \text{timeCost}()$   
 $\pi_c = \text{EXTENDED-TREE-SEARCH}(P_s, t_{ts})$ , see Alg. 6  
**return**  $\pi_c$

---

will increase its visit number by 1. Meanwhile, all actions ahead of *node* in the skeleton will be effectuated to update the environment to which state *node* is initially built on. By complying with Alg. 4 and Alg. 3, a global convergence of the output  $\pi_c$  to the optimal plan is guaranteed. We refer to [24] for the detailed proof.

*Property 2:* Probabilistic completeness in the extended tree search. Alg. 6 is structured in line with the PW-UCT framework, and it retains all the convergence property. The optimal concrete plan within the extended decision space (implied by  $P_s$ ) can be found after  $t_{ts}$  goes infinite.

*C. The eTAMP algorithm*

A serial combination of Alg. 2 and Alg. 6 gives out the overall eTAMP algorithm, as summarized in Alg. 7. It takes as input a standard PDDLStream task  $T_{\text{stream}} = \langle \text{Objects}, S, G, \text{Actions}, \text{Streams} \rangle$ , and two hyper-parameters:  $k$  and  $t_b$ , allocating computational resources to the planner. Its dependencies include a set of black-box generators for streams, and a robotic simulator.

## VI. EMPIRICAL EVALUATION

We evaluate the proposed eTAMP algorithm in three multi-stage robot manipulation tasks: transportation, cooking, and regrasping. None of them can be solved by a whole-piece motion planner, or a standalone AI planner. The Adaptive algorithm from [6] represents the best performance of existing PDDLStream methods, and it serves as a baseline in these evaluation tasks. The two general-purpose TAMP algorithms, adaptive PDDLStream, and eTAMP, share one internal simulator that is implemented with PyBullet [27] for hosting the environment state. Both planners are terminated after a 700-second timeout for each of the three tasks. For eTAMP we set  $k = 50$ , and it is shown enough for the considered tasks. The performance of each algorithm is evaluated over 100 random instances of each task. Whenever a feasible concrete plan is found, the algorithm is stopped and the computation time is calculated. It is noteworthy that both algorithms employ the same stream generators for decision nodes, which have zero built-in heuristics.

*A. The transportation task*

This task comes from the motivation example of Fig. 1. For the 1-body scenario, Adaptive PDDLStream and eTAMP can solve the problem respectively in **0.44s** and **3.94s** on average. The larger time cost of eTAMP is due to the extra effort of its top-k planner in finding all the  $k$  alternatives, which in this case are somewhat "unnecessary". When more than one body is involved, the problem becomes challenging since the symbolic planners at high hierarchies are unaware of the geometric constraints, as imposed by the taller red body in Fig. 1b. With the help of top-k backup plans, eTAMP solves the 2-body problem in **11.95s** on average. However, Adaptive PDDLStream is unable to find a feasible plan within the time budget of 700s. To further evaluate the algorithms, we propose a 3-body scenario, where two bodies must be relocated before the target one can be reached. eTAMP solves this harder problem in **157.70s** while Adaptive PDDLStream still fails to give a solution.

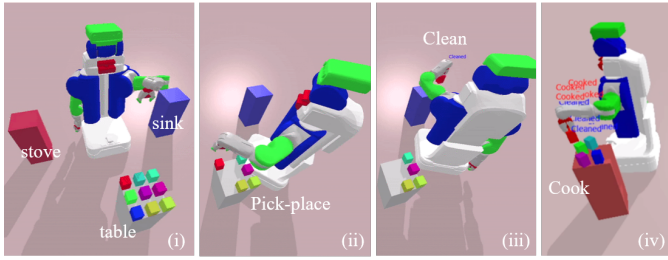


Fig. 3: The cooking task: Given an initial number of bodies to be cooked, the mobile manipulator should cook by first placing the body on the sink for cleaning, and then place it on the stove for washing.

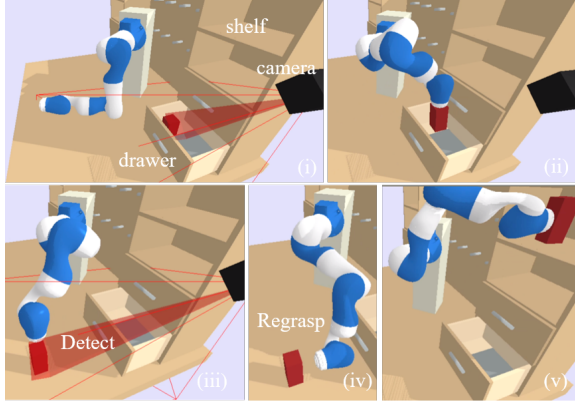


Fig. 4: The regrasping task: The only way the robot could move the cuboid body from the drawer to the shelf is by regrasping it with a different direction after taking it from the drawer.

### B. The cooking task

This task evaluates the TAMP algorithms’ scalability to mobile manipulation. As shown in Fig. 3, this task aims to transport a given number of bodies initially placed on the table to the stove for cooking. Before cooking, a body must be once placed on the sink for washing. The cooking task features long-horizon robot manipulation. For example, a task instance with 5 bodies involved requires at least 10 free-hand movements plus 10 hold-body movements. The main difficulty of this task lies in the tight target region, where several bodies need to be placed. As both Adaptive PDDLStream and eTAMP are uninformed about this potential congestion beforehand, they must identify this constraint through trial and error. During binding search, the consistent value estimation of each decision is critical for this delayed-reward case. Fig. 5 shows the time consumption of Adaptive PDDLStream and eTAMP in solving the cooking task. With problems with fewer bodies, eTAMP costs slightly more time than Adaptive. On larger numbers of bodies, eTAMP outperforms Adaptive PDDLStream by using optimal tree search, especially when the feasible space gets tighter (see Fig. 3 (iv)).

### C. The regrasping task

In this task, the robot should transport a cuboid body from the drawer to the shelf. To realize a real-world setting, we consider a fixed camera that has to be queried for object pose before it can be grasped. The robot can choose between 5 grasping directions to the body: along the normal vectors of the top surface and four side surfaces. The cuboid body at

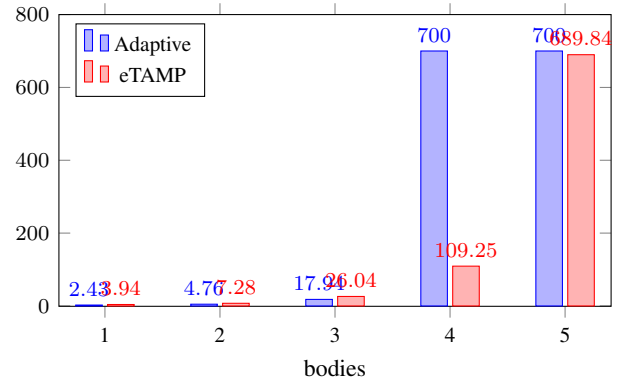


Fig. 5: Average planning cost (seconds) of Adaptive PDDLStream and eTAMP in the cooking task. A time cost of 700 means none solutions are found within the time budget.

its initial pose is limited to top grasping by its surroundings, while it must be grasped from the side while being placed on the shelf, as depicted in Fig. 4. This task is challenging for skeleton planning since a regrasping behavior must be composed by planners based on only elemental operators. Otherwise, the problem cannot be solved. Moreover, this task is geometrically difficult. Even if a correct skeleton is given, the planner must search the hybrid decision space (discrete grasping directions and continuous body poses) for feasible bindings. The average time cost of eTAMP for solving this task is **235.63 s**. In comparison, Adaptive PDDLStream is unable to find any solution within the time budget.

*Remarks:* Our empirical evaluation shows the effectiveness of the proposed eTAMP algorithm in solving challenging, multi-stage manipulation tasks. Whereas the most competitive baseline in the literature fails to scale to increasing numbers of decision steps, our algorithm trades off slightly higher planning time in simple scenarios for high effectiveness when multi-object and multiple actions are in place. The proposed seamless combination of symbolic top-k planning with streams for searching alternative skeleton plans, with the optimality of PW-UCT, leads to a powerful algorithm that can handle the combinatorial complexity of long-horizon manipulation tasks.

## VII. ACKNOWLEDGEMENT

G. Chalvatzaki’s work is funded by the EN DFG Programme iROSA (CH 2676/1-1).

## VIII. CONCLUSION

We propose eTAMP as a general-purpose planner of robot manipulation tasks with long horizons that demand symbolic sequencing of operators and binding search of motion parameters under geometric constraints. eTAMP addresses the difficulty of incomplete domain in task planning by using symbolic top-k planning for diverse backup skeletons. It integrates a uniform tree search method for the extended decision space based on UCB and PW that allow global convergence to optimal plans, in spite of the stochastic transition dynamics during planning. The empirical results reveal that our approach outperforms the existing state-of-the-art PDDLStream algorithm, especially in problems where alternative skeletons



are indispensable. We attribute the capability of eTAMP for solving long-horizon tasks to the proposed hierarchical planner structure over actions and skeletons, and the scalability of UCT to large decision sequences (empirically proved in [28]). In future work, we will explore a more informative sampling method for stream generators to improve binding efficiency.

#### REFERENCES

- [1] L. P. Kaelbling and T. Lozano-Pérez, “Hierarchical task and motion planning in the now,” in *2011 IEEE International Conference on Robotics and Automation*, pp. 1470–1477, IEEE, 2011.
- [2] S. Srivastava, E. Fang, L. Riano, R. Chitnis, S. Russell, and P. Abbeel, “Combined task and motion planning through an extensible planner-independent interface layer,” in *2014 IEEE international conference on robotics and automation (ICRA)*, pp. 639–646, IEEE, 2014.
- [3] C. Weber and D. Bryce, “Planning and acting in incomplete domains,” in *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 21, 2011.
- [4] H. H. Zhuo, T. A. Nguyen, and S. Kambhampati, “Refining Incomplete Planning Domain Models Through Plan Traces,” in *IJCAI*, pp. 2451–2458, 2013.
- [5] N. T. Dantam, Z. K. Kingston, S. Chaudhuri, and L. E. Kavraki, “Incremental Task and Motion Planning: A Constraint-Based Approach,” in *Robotics: Science and systems*, vol. 12, p. 00052, Ann Arbor, MI, USA, 2016.
- [6] C. R. Garrett, T. Lozano-Pérez, and L. P. Kaelbling, “PDDLStream: Integrating symbolic planners and black-box samplers via optimistic adaptive planning,” in *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 30, pp. 440–448, 2020.
- [7] C. R. Garrett, R. Chitnis, R. Holladay, B. Kim, T. Silver, L. P. Kaelbling, and T. Lozano-Pérez, “Integrated task and motion planning,” *arXiv preprint arXiv:2010.01083*, 2020.
- [8] M. Katz, S. Sohrabi, O. Udrea, and D. Winterer, “A novel iterative approach to top-k planning,” in *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 28, 2018.
- [9] C. R. Garrett, T. Lozano-Pérez, and L. P. Kaelbling, “Stripstream: Integrating symbolic planners and blackbox samplers,” *arXiv preprint arXiv:1802.08705*, 2018.
- [10] N. Shah, D. K. Vasudevan, K. Kumar, P. Kamojjhala, and S. Srivastava, “Anytime integrated task and motion policies for stochastic environments,” in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 9285–9291, IEEE, 2020.
- [11] M. Toussaint, “Logic-Geometric Programming: An Optimization-Based Approach to Combined Task and Motion Planning,” in *IJCAI*, pp. 1930–1936, 2015.
- [12] Z. Kingston, A. M. Wells, M. Moll, and L. E. Kavraki, “Informing multi-modal planning with synergistic discrete leads,” in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3199–3205, IEEE, 2020.
- [13] D. McDermott, M. Ghallab, A. Howe, C. Knoblock, A. Ram, M. Veloso, D. Weld, and D. Wilkins, “PDDL—the planning domain definition language,” 1998.
- [14] C. Dornhege, P. Eyerich, T. Keller, S. Trüg, M. Brenner, and B. Nebel, “Semantic attachments for domain-independent planning systems,” in *Towards service robots for everyday environments*, pp. 99–115, Springer, 2012.
- [15] C. R. Garrett, T. Lozano-Pérez, and L. P. Kaelbling, “Sampling-based methods for factored task and motion planning,” *The International Journal of Robotics Research*, vol. 37, no. 13-14, pp. 1796–1825, 2018.
- [16] M. Helmert, “The fast downward planning system,” *Journal of Artificial Intelligence Research*, vol. 26, pp. 191–246, 2006.
- [17] A. Torralba, V. Alcázar, D. Borrajo, P. Kissmann, and S. Edelkamp, “SymBA\*: A symbolic bidirectional A\* planner,” in *International Planning Competition*, pp. 105–108, 2014.
- [18] D. Speck, R. Mattmüller, and B. Nebel, “Symbolic top-k planning,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, pp. 9967–9974, 2020.
- [19] L. Kocsis and C. Szepesvári, “Bandit based monte-carlo planning,” in *European conference on machine learning*, pp. 282–293, Springer, 2006.
- [20] P.-A. Coquelin and R. Munos, “Bandit algorithms for tree search,” *arXiv preprint cs/0703062*, 2007.
- [21] B. Kim, K. Lee, S. Lim, L. Kaelbling, and T. Lozano-Pérez, “Monte Carlo tree search in continuous spaces using Voronoi optimistic optimization with regret bounds,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, pp. 9916–9924, 2020.
- [22] S. M. LaValle *et al.*, “Rapidly-exploring random trees: A new tool for path planning,” 1998.
- [23] C. Dellin and S. Srinivasa, “A unifying formalism for shortest path problems with expensive edge evaluations via lazy best-first search over paths with edge selectors,” in *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 26, 2016.
- [24] D. Auger, A. Couetoux, and O. Teytaud, “Continuous upper confidence trees with polynomial exploration-consistency,” in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pp. 194–209, Springer, 2013.
- [25] R. Coulom, ““elo ratings” of move patterns in the game of go,” *ICGA journal*, vol. 30, no. 4, pp. 198–208, 2007.
- [26] G. M. J. Chaslot, M. H. Winands, H. J. V. D. HERIK, J. W. Uiterwijk, and B. Bouzy, “Progressive strategies for Monte-Carlo tree search,” *New Mathematics and Natural Computation*, vol. 4, no. 03, pp. 343–357, 2008.
- [27] E. Coumans and Y. Bai, “Pybullet, a python module for physics simulation for games, robotics and machine learning,” *URL <http://pybullet.org>*.
- [28] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre,

G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, *et al.*, “Mastering the game of Go with deep neural networks and tree search,” *nature*, vol. 529, no. 7587, pp. 484–489, 2016.

# Limits and Possibilities of Multi-Goal Task-Motion Planning

Stefan Edelkamp, AI Center, CTU Prague

(joint work with Erion Plaku and Jan Faigl)

**Abstract.** Multi-Goal Motion Planning is a robotic TSP where it is necessary to satisfy motion constraints and physical limitations of real robotic systems, which range from data collection planning such as environment monitoring, human-robot collaborations, manipulation tasks, but also radiation therapy and robotic surgery.

We consider model-free approaches that apply to any vehicle model. In this work we look recent, current and blue sky ideas to advance multi goal task-motion planning. This way we study its limits and possibilities, working towards challenges and opportunities.

## Introduction

Sampling-based motion planning has seen tremendous improvements in terms of scalability and expressiveness. However, the integration of task and motion planning still poses several open challenges.

Consider for example the scenario, where a team of robots must coordinate their movements and their task allocations to complete a mission as soon as possible. Different tasks have different durations and resource consumption, and each task must be performed within one of the available time windows. Such a challenging scenario requires temporal reasoning in addition to finding the best path between task locations. We address these challenges by combining sampling-based motion planning with temporal task planning.

With the technology we have developed over the years we efficiently generate and execute plans for long-term tasks of several moving robots in rough 3D environments with

obstacles, fast enough even for real-time simulations (see Figure 1).

The robots, the environment model and the planning task specification can be adapted non-intrusively, essential for solution prototypes in many applications from surveillance with drones, via logistics applications to service robotics.

Multi-Goal Motion Planning (MGMP) is a robotic TSP where it is necessary to satisfy motion constraints and physical limitations of real robotic systems [1] that range from data collection planning such as environment monitoring [2], manipulation tasks [3], human-robot collaboration [4], but also radiation therapy [5] and robotic surgery [6]. We consider model-free approaches that work with any vehicle model.

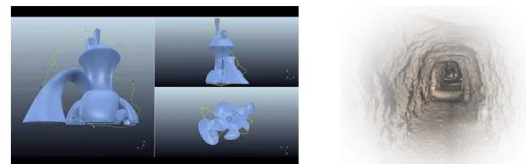


Figure 1. Involved 3D Environments for Multi-Goal Motion Planning.

Multi-Goal Task-Motion Planning combines task planning with multi-goal motion planning and touches many involved research questions, e.g., handling the dynamics of the robot, avoiding existing obstacles, visiting multiple waypoints, performing tasks, all this while dealing with ordering time windows, deadlines, and resource constraints (see Figure 3).

For a robot in a real-world environment there are frequent task requests like visiting goals (in the best possible order) to minimize travel time for inspecting an object, before considering another. These temporal requests are attached to the physical world and must be reflected in the motion planning of the robot. For complex

operations on a robot, the reachability of one goal via path planning is not enough, as many goals in the form of waypoints must be visited for completing a plan. Moreover, robots have tasks to perform at each waypoint, that take time and require resources.

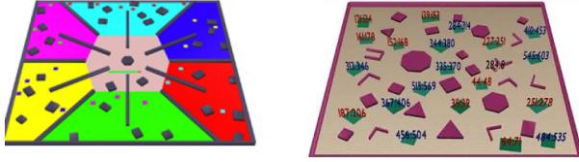


Figure 2. Inspection and Temporal Multi-Goal Motion Planning.

While there are automata-based solutions for finding plans for several goals, by the exponential growth in the automata representation, they do not scale well. Engineered solutions offer better results as the performance of our prototype system shows. Ever since there have been an attempt to solve the integrated task and motion planning problem for complex robots.

We join sampling-based motion planning with guidance, established by heuristics that are found through problem abstractions. The challenge is to combine the exploration in the continuous space of valid trajectories, imposed by the dynamics of the robot's nonlinear, non-holomorphic, and hybrid system specification, and the discrete space that is associated with the task planning. While there is an alternative to include motion planning modules in the preconditions of an action, our contribution is iterating a loop of solving the abstraction to guide the expansion of the sampling-based motion tree to generate valid trajectories.

## State-of-the-Art

Combinatorial optimization is an essential part of the robot motion planning problem where the TSP is the fundamental formulation to determine a cost-efficient plan for robotic systems. The TSP is well-studied problem with available optimal solvers such as [7] with efficient cutting-plane techniques, Dynamic Programming approaches [8, 9], efficient heuristic [10], soft-computing techniques such as genetic algorithms [11], ant colony optimization [12], neural networks for routing problems, [13], and also combinatorial

metaheuristics [14] such as simulated annealing [15], variable neighborhood search (VNS) [16] and greedy randomized adaptive search procedure (GRASP) [17] both with several variants, modifications, and improvements [18, 19].

Many generalizations of the TSP have been proposed motivated by applications fields such as logistics, where various formulations of the Vehicle Routing Problem (VRP) [20] play the essential role [21] but recently also motivated by the deployment of electric vehicles [22]. Although the proposed research might overlap with other routing problems, it is primarily motivated by data collection missions and robotic scenarios, where the fundamental challenge is arising from the need to reason about the travel cost to regions and not to single points of interest.

Thus, the TSPN and GTSP with Neighborhoods (GTSPN) [23] are of our particular interest as they combine routing with continuous optimization. Besides, depending on the application context, the Sequence Dependent TSP (SDTSP) [24] is also highly relevant, e.g., for drone delivery optimization with payload dependent battery consumption addressed by the Mixed-Integer Linear Programming (MILP) in [25].

The TSPN stands to find a cost-efficient plan on how to visit each of the given sensing regions that form the respective neighborhoods. TSPN is APX-hard [26], and approximation algorithms have been proposed for restricted variants of the neighborhoods such as disjoint unit disks that are arbitrarily connected [27], disjoint convex fat neighborhoods [28]. It has been recently been tackled by hyperplanes in the  $d$ -dimensional Euclidean space [29] and subproblem optimization [30]. The Mixed-Integer Non-Linear Programming (MINLP) is computationally intractable [31].

Therefore, heuristics are studied for the disk-shaped neighborhoods that attract the attention of the community as the Close Enough TSP (CETSP) [32] addressed by the concept of supernodes (region samples) [33]. Existing approaches to the CETSP are surveyed in [34], where the best results are reported for the GTSP-based solution with high computational requirements (in hours).

Contrary to that, high-quality solutions with low computational requirements are reported [35] for the Growing Self-Organizing Array (GSOA) [36] based on an online sampling of the regions during the unsupervised learning of the sequencing part of the problem. New best solutions of the existing benchmarks [34] are found in a fraction of second, which significantly outperforms GTSP-based approaches [36].

The GTSP is reported to be solved optimally using efficient branch-and-cut scheme up to 442 targets organized into cluster [37]. The existing approximation algorithms [38] are reported to guarantee low solution quality [39] and heuristic solutions have been proposed [40, 41] based on hybrid approach [42], memetic algorithm [39], and adaptive large neighborhood search [43].

A further generalization of the discrete GTSP to a continuous neighborhood called the GTSP with Neighborhoods (GTSPN) [23] has been proposed to overcome limitations of the continuous and discrete formulations of the TSPN [44]. Instances of the GTSPN are motivated by tasks of redundant robotic manipulators, inspection with multiple views, and surveillance missions have been addressed by the genetic algorithm [23] that has been outperformed by fast solvers based on the decoupled approach and GSOA-based construction heuristic [45].

The important generalization of the TSP-like formulations to address practical needs of robotic planning is the class of TSP with Profits [46], specifically Price-Collecting TSP (PCTSP) [47] and Selective TSP [48] also called the Orienteering Problem (OP) [49]. The PCTSP with Neighborhoods (PCTSPN) has been introduced in [50] as a suitable problem formulation for data collection planning for cases when a combination of sensor measurements can lead to information being subadditive or superadditive, and visiting the most profitable sensors can save travel cost by avoiding less profitable ones. The heuristic solution [50] has been surpassed by unsupervised learning [51] that has been further generalized to problems with spatially correlated sensors measurements [52, 53]. The OP formulation is suitable for scenarios with additive rewards and limited operational time [54] to determine a tour with the cost that does

not exceed the limited travel budget to visit the most rewarding locations. Existing combinatorial approaches [55, 56] for the regular OP have been generalized to address limitations arising from robotic systems such as fuel consumption [57] and non-holonomic constraints [58, 59]. The generalization allowing to exploit a non-zero sensing range has been introduced as the OP with Neighborhoods (OPN) [60] and addressed by unsupervised learning [61] later generalized for spatially correlated measurements [53] including multi-vehicle scenarios [62, 63].

The discrete variant of the OPN has been introduced as the Set OP (SOP) in [64] and addressed by the Mixed-Integer Programming (MIP) and a matheuristic solution algorithm. The MIP [64] is outperformed by the novel ILP formulation introduced in [65], but therein proposed less demanding VNS-based solver provides competitive solutions. The current advancements on multi-goal motion planning are limited to model-based planning with curvature-constrained path using the Dubins vehicle model or Bezier curves in an environment without obstacles [66]. Besides, the existing theoretical results are suitable only for scenarios without obstacles, and only for 2D cases, which has limited applicability.

## Objectives and Methodology

The general vision is to steer the robots of the 21st century. We generalize the results in multi-goal motion planning to more complex vehicle models, and to combine motion planning with task planning for longer-term autonomy. Specifically, we are targeting time-optimal planning as the necessary building blocks for multi-goal motion planning. Furthermore, we investigate the extension towards scenarios with obstacles, where we plan to leverage the current achievements with a discretized environment where search-based planning techniques with performance bounds can be utilized. Model-free frameworks and the integration of task and motion planning is of highest interest for the research communities in planning and robotics.

There is a rising number of contributions and workshops (e.g., at RSS, IROS ICRA, AAI, and IJCAI) dedicated to this topic, but the problems are widely considered being

unsolved. With our advances in model-free multi-goal motion planning with obstacles [67, 68, 69] we touched the tip of the iceberg, leading to wider range of robot controllers that is relevant also for reducing cost for complex robot missions in otherwise hardly accessible terrains.

Once the waypoints have been computed (itself a hard problem that we have approximated), the advance in multi-goal motion planning has immediate effects to the so-called inspection or surveillance problem, where objects must be examined by the robot for the presence of defects, such as oil leakage in underwater pipelines. We combine our scalable motion planning framework with fast discrete solvers and PDDL planners. The integrated approach of growing a motion tree of valid trajectories with planner guidance computed at its leaves is original and novel. By the higher efficiency of the robot's path planning, simulation and waypoint finding, task-motion planning problems are now ready to be solved.

We extend an existing framework for solving physical routing problems. Besides the standard setting of finding valid trajectories for a given robot model with nonlinear dynamics, in a mesh environment with obstacles it will support

- temporal task planning, with tasks to be performed only within certain time windows;
- multiple robots, computing motion trajectories that do not overlap;
- precedence constraints on waypoints extending routing problems to respect colors;
- resource reasoning such including capacity constraints for picking up larger or heavier objects;
- energy consumption reasoning such as multiple recharging and facility location;
- pickup and delivery constraints for moving objects between waypoints;
- inspection problem solving, generating a reduced set of waypoints;
- safe-emergency planning to have a backup plan if the overall plan fails.

The practical output is an integrated task and motion planning software module for robots in simulation and the real-world usable for a wide range of applications and robots.

## Limits and Possibilities

In the following we characterize possibilities and limits in multi-goal task-motion planning.

### Multi-Robot Multi-Goal Task-Motion Planning

One natural task is to extend our model-free approach [67] to several robots to visit multiple waypoints, and still be able to avoid unwanted behavior like deadlocks and collisions, i.e., solving complex multi-robot multi-goal scenarios. The solution is inspired and provides add-ons to multi-agent systems. We map the problem of every two robots having non-intersecting tours to approximate string matching, to be solved with dynamic programming.

Therefore, we look at multi-goal task-motion planning with multiple agents, as an extension MGTMP based on solving physical TSPs with one agent. There are early studies helping several robots in achieving their individual goals while minimizing both congestion and overall travel time [70].

First model-free solutions were given by [71] the discrete problem in the framework that has to be solved for each leaf node in the motion tree to guide its expansion is a multi-agent path-finding problem.

### Multi-Robot Multi-Goal Task-Motion Planning for the Simulation Framework

Based on the precursor work [70, 49] and initial studies on the physical VRP in simplified settings [72, 73], we have work on a flexible and efficient framework solution to solve the problem.

In a cluttered or narrow environment of the randomized roadmap robots have let others pass, which induces a lot of maneuvers and even without motions itself is an NP-hard problem (think of the sliding-tile puzzle, with the tiles being robots). The derived solutions including robot motions is implemented in the framework.



## Multi-Robot Multi-Goal Task-Motion PDDL Planning

Imposing the more general STRIPS formalism [74] with state-dependent action inter-dependencies can cover multi-robot scenario, at least for the abstract problem assigns cost 1 is assigned to each action. For faster prototypes and the integration of more complex tasks with motions, we push forward the use of the PDDL language to include domain-independent problem solvers in multi-robot multi-goal motion planning problems.

## Multi-Robot Multi-Goal Task-Motion Planning on (Simulated) Robots

The real challenge is to lift the results to a realistic robot scenario, with different kinds of robots solving together one problem, as, e.g., imposed in DARPA SubT or similar challenges serve as benchmarks for this task. As input comes in with point clouds, this leads to different kinds of maps.

The problem is set up and we provide an alternative prototypical implementation for solving the discrete multi-agent pathfinding problem that is based on dynamic programming, which optimally resolve conflicts for a deterministic or randomized selection of vehicle pairs. The robot motion planning simulation framework integrates individual solvers and (multiagent) PDDL planning. We expect the acquisition of PDDL models compatible with state-of-the-art AI planners, to assist with multiagent planning and software integration, and maps, e.g., generated from the point clouds of SubT.

## Temporal and Metric Multi-Goal Task-Motion Planning

Here we impose additional time windows constraints for reaching the goals, which is very important for real-world logistics e.g., for warehouses [75].

The temporal dimension of the task in form of scoring goals by the time point for achieving them is of high industrial impact, as projects with robots involved need to be scheduled and meet timing constraints. This has influenced the development of temporal planners. There are

solutions to the multi-robot path-finding problem, e.g., applied by Amazon Robotics (previously Kiva Systems) that operate on a grid of robot locations. By the lack of modelling dynamics, however, this limits the available speed of the vehicles substantially.

Thinking on logistics and resource handling this leads to more complex task-motion vehicle routing problems, that have been studied for decades in discrete form in OR [76], and if can be extended to motion simulations of robots in a virtual (or real) environments.

Besides multiple vehicles, time windows to be met, we have considered pickup- and delivery as well as capacity and priority constraints of homogeneous and heterogeneous vehicle fleets.

We envision to consider solving the multimodal so-called tourist travelers' problem or other types of the orienteering problem [49, 55, 56] for robots with motion dynamics. Here we have a tourist spending some days in the city. Starting at the hotel(s) he will use public and individual transport while increasing the number of sites that respect their opening hours.

## Traveling Salesman Problem with Time and Resources

Time is market; this means that visiting a waypoint in the plan is scored only in some time windows during the simulation, otherwise the payoff is reduced.

The temporal motion planning problem, however, is very challenging, as the complex dynamics of the robot may falsify suggested temporal plans during plan execution.

One of the discrete problems to solve is the TSP with time windows, which is even harder than the strongly NP-hard standard TSP problem [75] We have compared the working of a general temporal planner with specialized TSPTW solvers based on Branch-and-Bound and Monte-Carlo Tree Search. Similarly, refined resource handling and reduced energy consumption are two further objective in extending the motion planning approach that have to be addressed.

## Temporal and Metric PDDL Planner Integration

Fox & Long's PDDL2.1 [77] allows expressing numeric and temporal task planning and temporally extended goals. This enables resource handling as needed in logistic context of goods to be delivered.

With the help of planners capable of handling this larger level of expressiveness, we have derived some initial results shown that these more general solution methods are sufficiently competitive with the state-of-the-art in domain-dependent problem solving.

The problems are relevant for indoor logistics in warehouse and production systems of smart factories. For including timetables and time-dependent shortest path search in precursor work we have looked at monorail systems, cast as public transport for the robots.

### Application to Logistics

Besides the obvious application in warehouses, there is a large list of challenges in logistic domains, that request motion for completing a given task. The inclusion of time and resource handling in combination of moving agents is a game changer for which we have derived first solutions with pickup and deliveries using priced TSPs in the discrete abstraction [78].

There is a recent shift of attention from SubT to logistics that highlight the relevance. In this task, we connect our approaches to these application.

We contributed Capacitated TSP(TW/PD) solvers with APIs to the framework and metric-temporal PDDL-type planners, with an interface between the two to be used for communication. We work on optimal temporal solving and in multiagent robot solutions.

We provide logistic scenarios and implement the integration to the framework, setting up the scenarios and their evaluation in order to conduct a wider range of experiments and come up with vehicle routing problem solutions as obtained multiagent robot demonstrator.

## Inspection Problem Solving

Due to its industrial relevance, a holy grail in advanced robotics is solving the inspection problem [79], namely, to steer one (or a fleet of) autonomous vehicles (e.g., in an underwater mission) in order to inspect the surface of an object with specialized sensors. 3D trajectory planning requires advanced geometric reasoning (e.g. efficient collision detection, skeletonization algorithms).

For the simulation we used CGAL [80] and game engines [81]. While we have derived some initial results to the problem in 2D and 3D, inspecting objects from the in- and outside<sup>1</sup>, the problem induces many more facets than we have been considering so far.

### Refining 3D Solution on Meshes to Include Pyramid Visibility Constraints

For example, one would love to solve physical art gallery problems, namely computing waypoint locations for taking a larger sequence of 2D/3D in snapshots. Alternatively, followed by multi-goal motion planning, the task is to compute a low-cost tour to visit the waypoints, while limiting the range of the camera with pyramid visibility constraints and continuously taking pictures is even more ambitious.

### Solving Physical Watchman Routing Problems and PDDL Integration

We propose a robust and efficient inspection of the entire workspace in a watchman route based on automatically generated waypoints. The framework design includes several relevant technologies and refined algorithms such as medial axis transformation, shortest path approximation, and Monte-Carlo search for finding tours [81].

One simulation of the robot runs on Unity, while data processing executed in a Python server. The measured inspection coverage of the workspace on random terrains was at least 99.6%. We work on a framework and PDDL planner integration.

## Solving Multi-Robot Inspection Problems

The real quest, however, is to use not only one but several robots to join in and solve the inspection problem. There are challenges like task-sharing available in covering space in terms of increased visibility of the robots due to their limit sensor range.

As stated above, we have integrated the computational geometry algorithm library (CGAL) to the solver to support advanced mesh operations like Boolean intersections and skeletonizations. We look at scenarios to execute experiments in 2D and 3D in artificial maps and ones from the SubT DARPA challenge. The PhDs concentrate on APIs for discrete solvers for scalable motion planning and for hooks for integrating inspection with PDDL planning.

## Multi-goal Planning with Safe Emergency Trajectories

We investigate motion planning generalization, where additional mission constraints invoke a backup planning problem to satisfy the mission constraints. In particular, we address multi-goal planning with a safe emergency trajectory. The main challenge is to ensure at least one safe emergency plan for each point of the requested multi-goal trajectory to navigate to some designated location in a sudden state of emergency. A solution to such a mission constraint, one needs to solve a motion planning problem (finding the safe emergency trajectory) inside the multi-goal motion planning (finding multi-goal trajectory to perform the requested task) that further increasing the computational complexity of the MGTMP.

The addressed problem of safe task-motion planning calls for plans with the guaranteed trajectory to navigate to one of several designated spots at each point of the plan execution. Such a scenario can be reaching a landing an airplane, reaching a hospital, a battery location, or a populated area. Automated planning, however, often restricts to static environments, where only the acting agent can change its actions. Exogenous events may result in having to switch to emergency plans. Such safety constraints that require being able to

return to one of several designated secure areas influence plan quality so that the plan generation for one agent has to take into a plan in a limited set of trajectories.

We seek an approach for safe task-motion planning for a case of emergency. Planning in static environments accounts for generating optimized plans, for instance, for their length, makespan, or action cost. However, the environment is rarely static as exogenous events might occur without the consent of the agent. Therefore, safe task-motion planning needs to be “prepared” for sudden events requiring an emergency plan to be ready and available immediately when the events happen. We consider safe emergency multi-goal task-motion planning with resources in the presence of obstacles.

We aim at a full-fledged solution for high-dimensional robotic systems with nonlinear dynamics and non-holonomic constraints visit all goal regions fast in suitable cost-minimizing order in unstructured, complex environments and efficiently computes collision-free, dynamically-feasible, low-cost, energy-efficient, and safe trajectories that enable the robot to satisfy the task specification.

## Emergency-Aware Multi-goal Planning.

We leverage previous work [82] where multi-goal planning is employed in surveillance planning with safe emergency landing trajectories. The crucial property of safe landing is a minimum safe altitude of the vehicle that can be found by trajectory planning to nearby airports using sampling-based motion planning such as RRT\* [83]. A trajectory is considered safe if the vehicle is at least at the minimum safe altitude at any point of the trajectory.

Although [82] provides a solution of the multi-goal planning with a safe emergency landing guarantee, it is limited because it strongly depends on the precomputed map of the safe altitude for vehicle gliding in the total loss of thrust using the gliding model. Besides, it is assumed the vehicle moves only above the terrains and not, e.g., under a bridge, and it is limited to planning in 2D.

Therefore, we plan to generalize the safe landing trajectory to a safe emergency trajectory without assuming the motion only above the terrain (obstacles). We further propose to generalize emergency trajectories to multi-modal trajectories to address emergencies caused by faults with different impacts on the motion capabilities, e.g., partial loss of thrust.

## Risk-aware Multi-goal Planning

We generalize existing emergency aware multi-goal planning with a relaxed notion of the safe emergency trajectory guarantee. In [82], the target sites need to be at a safe altitude that might not always be possible. Hence, reaching such a location includes a possible risk depending on a failure type. The risk due to possible crashes can be based on the economic evaluation and casualties [84, 85]. A precise impact location cannot be predicted, and a stochastic model has to be used in the case of uncontrolled fall of the aircraft [86].

Possible risk to people, ground vehicles, and aircraft for small UAVs ground risk map is proposed in [84], and three possible events are discussed in [85], i.e., loss of control with an uncontrolled crash on the ground; impact with someone; and fatal injury to the hit person. We propose to address the risk-aware planning using the existing approach of risk map assessment [87] to find the trajectory with the minimal risk [88]. However, the existing approaches are limited to a fixed flight altitude. Therefore, we propose to extend the existing approaches and generalize them to various flight levels and speeds.

## Risk-aware Multi-goal Planning in Dynamic Environments

We plan to generalize the approaches being developed towards dynamic environments.

The proposed generalization to risk-aware multi-goal planning is ambitious, we plan ample time for it. We plan to address the dynamic environments by the concept of the life-long A\* [89] and RRTx [90] to achieve computationally feasible generalization towards dynamic environments. We maximize the effort to generalize the existing risk-aware and safe emergency guarantee towards different types of

failures for aerial vehicles and generalize the approaches to other types of vehicles and scenarios as those relevant for ground vehicles.

We plan to empirically evaluate our combined discrete search and sampling-based motion planning approach in a set of 3D scenarios. We define the setting for the problem and work on a solution in the framework. We work on the generalization of the existing approach [82] towards risk-aware planning in dynamic environments.

## Conclusion and Discussion

Integrated task and motion planning is essential, as robot motion planning alone does not serve long term goals, and task planning alone cannot deal with the intrinsic challenges of robot motion. The world is going to become more and more populated with robots, and our solution has a tremendous impact on their abilities to plan.

The scientific contribution of multi-goal motion planning aims to provide optimized solution of routing problems with continuous model-free robotic applications. The primary motivation is planning, where the current solutions rely on sampling continuous domains into a finite set of values being addressed as variants of the TSP.

Solving the inspection problem certainly is a holy grail and wrt applications like finding oil leakages in underwater pipes a million dollar business. The ultimate motivation, however, is to produce the software that steers the next generation of autonomous robots, a surplus towards long-term autonomy.

The optimal solution of the discretized problem does not guarantee an optimal solution to the original problem. Furthermore, heuristics usually provide better solutions, however, without any solution quality estimates. We aim to develop the fundamental blocks to assess the solution and provide quality guarantee. The model-free solvers are general enough to open a wide range of possible applications.

The established algorithmics foster further research in challenging optimization problems, and novel computational techniques improve scalability of nowadays and future algorithms to solve large instances.



The research fertilizes the deployments of robotic systems in various fields. Finally, stability analysis, identifying stability regions, and finding robust solutions under perturbations are important steps towards applications with dynamic and on-demand changes.



Figure 3. Established robotic systems for this research.

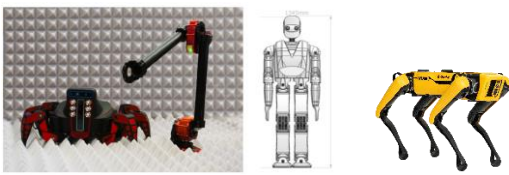


Figure 4. New robotic systems for this research.

CTU is experienced with complex robotic (see Figures 3 and 4) systems with the current highlight of the CTU-CRAS team in the DARPA SubT Challenge.

Using our motion planning framework, we perform research in simulation. On the other hand, experimenting with real robotic systems relies on hardware availability; however, the walking robots and aerial vehicles are part of the effort in the DARPA SubT Challenge.

Robotic arms are already available at CTU within the Center for Robotics and Autonomous Systems (CRAS). For execution plans on real robots, however, we mostly use the robot hardware infrastructure present of CTU. We also have a 3D print of the Nibro-OS2X, the humanoid robot from Univ. of Bonn, which will directly participate from the improved multi-goal task-motion motion planning.

## References

[1] S. Alartartsev, S. Stellmacher, and F. Ortmeier, "Robotic task sequencing problem: A survey," *Journal of Intelligent & Robotic Systems*, 80(2):279–298, 2015.

[2] M. Dunbabin and L. Marques, "Robots for environmental monitoring: Significant advancements and applications," *IEEE Robotics & Automation Magazine*, 19(1):24–39, 2012.

[3] M. Saha, T. Roughgarden, J.-C. Latombe, and G. Sánchez-Ante, "Planning tours of robotic arms among partitioned goals," *International Journal of Robotics Research*, 25(3):207–223, 2006.

[4] H. Cai and Y. Mostofi, "A human-robot collaborative traveling salesman problem: Robotic site inspection with human assistance," in *American Control Conference (ACC)*, 2016, pp. 6170–6176.

[5] B. Ye, Q. Tang, J. Yao, and W. Gao, "Collision-free path planning and delivery sequence optimization in noncoplanar radiation therapy," *IEEE Transactions on Cybernetics*, 49(1):42–55, 2019.

[6] E. Lobaton, J. Zhang, S. Patil, and R. Alterovitz, "Planning curvature-constrained paths to multiple goals using circle sampling," in *ICRA*, 2011, pp. 1463–1469.

[7] D. Applegate, R. Bixby, V. Chvátal, and W. Cook, "CONCORDE TSP Solver," 2003, URL <http://www.tsp.gatech.edu/concorde.html>, [cited 2020-03-20].

[8] M. Held and R. M. Karp, "A dynamic programming approach to sequencing problems," *Journal of the Society for Industrial and Applied Mathematics*, 10(1):196–210, 1962.

[9] C. Chauhan, R. Gupta, and K. Pathak, "Survey of methods of solving TSP along with its implementation using dynamic programming approach," *International Journal of Computer Applications*, 52(4):12–19, 2012.

[10] K. Helsgaun, "An effective implementation of the Lin-Kernighan traveling salesman heuristic," *European Journal of Operational Research*, 126(1):106–130, 2000.

[11] J.-Y. Potvin, "Genetic algorithms for the traveling salesman problem," *Annals of Operations Research*, 63(3):337–370, 1996.

[12] M. Dorigo, V. Maniezzo, and A. Colnori, "Ant system: optimization by a colony of cooperating agents," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 26(1):29–41, 1996.

[13] J. J. Hopfield and D. W. Tank, "“Neural” computation of decisions in optimization problems," *Biological Cybernetics*, 52(3):141–152, 1985.

[14] M. Gendreau and J.-Y. Potvin, "Metaheuristics in combinatorial optimization," *Annals of Operations Research*, 140(1):189–213, 2005.

[15] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, 220(4598):671–680, 1983.

[16] N. Mladenovic and P. Hansen, "Variable neighborhood search," *Computers & Operations Research*, 24(11):1097–1100, 1997.

[17] T. A. Feo and M. G. C. Resende, "Greedy randomized adaptive search procedures," *Journal of Global Optimization*, 6(2):109–133, 1995.

[18] P. Hansen, N. Mladenovic, and J. A. M. Perez, "Variable neighborhood search: Methods and applications," *Annals of Operations Research*, 175(1):367–407, 2010.

[19] M. G. C. Resende and C. C. Ribeiro, *Optimization by GRASP: Greedy randomized adaptive search procedures*, Springer-Verlag, 2016.

[20] R. Banos, J. Ortega, C. Gil, A. L. Márquez, and F. de Toro, "A hybrid meta-heuristic for multi-objective vehicle routing problems with time windows," *Computers & Industrial Engineering*, 65(2):286–296, 2013.

[21] L. Costa, C. Contardo, and G. Desaulniers, "Exact branch-price-and-cut algorithms for vehicle routing," *Transportation Science*, 53(4):946–985, 2019.

[22] T. Erdelic and T. Caric, "A survey on the electric vehicle routing problem: Variants and solution

- approaches,” *Journal of Advanced Transportation*, 2019, Art. ID 5075671.
- [23] I. Gentilini, Multi-goal path optimization for robotic systems with redundancy based on the traveling salesman problem with neighborhoods, Ph.D. thesis, Carnegie Mellon University, 2012.
- [24] A. Alkaya and E. Duman, “A new generalization of the traveling salesman problem,” *Applied and Computational Mathematics*, 9(2):162–175, 2010.
- [25] M. Torabbeigi, G. J. Lim, and S. J. Kim, “Drone delivery scheduling optimization considering payload-induced battery consumption rates,” *Journal of Intelligent & Robotic Systems*, 97(3):471–487, 2020.
- [26] S. Safra and O. Schwartz, “On the complexity of approximating TSP with neighborhoods and related problems,” *Computational Complexity*, 14(4):281–307, 2006.
- [27] A. Dumitrescu and J. Mitchell, “Approximation algorithms for TSP with neighborhoods in the plane,” *Journal of Algorithms*, 48(1):135–159, 2003.
- [28] M. de Berga, J. Gudmundsson, M. J. Katz, C. Levkopoulou, M. H. Overmars, and A. F. van der Stappen, “TSP with neighborhoods of varying size,” *Journal of Algorithms*, 57(1):22–36, 2005.
- [29] A. Antoniadis, K. Fleszar, R. Hoeksma, and K. Schewior, “A PTAS for euclidean TSP with hyperplane neighborhoods,” in *Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 1089–1105, Society for Industrial and Applied Mathematics, 2019.
- [30] A. Clark, “A submodular optimization approach to the metric traveling salesman problem with neighborhoods,” in *IEEE Conference on Decision and Control (CDC)*, 2019, pp. 3383–3390.
- [31] I. Gentilini, F. Margot, and K. Shimada, “The travelling salesman problem with neighbourhoods: MINLP solution,” *Optimization Methods and Software*, 28(2):364–378, 2013.
- [32] D. J. Gulczynski, J. W. Heath, and C. C. Price, “The close enough traveling salesman problem: A discussion of several heuristics,” in F. B. Alt, M. C. Fu, and B. L. Golden, eds., *Perspectives in Operations Research: Papers in Honor of Saul Gass’ 80th Birthday*, pp. 271–283, Springer US, 2006.
- [33] Bo Yuan, M. Orłowska, and S. Sadiq, “On the optimal robot routing problem in wireless sensor networks,” *IEEE Transactions on Knowledge and Data Engineering*, 19(9):1252–1261, 2007.
- [34] W. K. Mennell, Heuristics for solving three routing problems: Close-enough traveling salesman problem, close-enough vehicle routing problem, sequence-dependent team orienteering problem, Ph.D. thesis, University of Maryland, 2009.
- [35] H. Zheng, J. Guo, and C. Bai, “Self-organizing neural network based mission planning for space unmanned system,” in *2019 IEEE International Conference on Unmanned Systems (ICUS)*, 2019, pp. 114–118.
- [36] J. Faigl, “GSOA: growing self-organizing array - unsupervised learning for the close-enough traveling salesman problem and other routing problems,” *Neurocomputing*, 312:120–134, 2018.
- [37] M. Fischetti, J. J. Salazar Gonzalez, and P. Toth, “A branch-and-cut algorithm for the symmetric generalized traveling salesman problem,” *Operations Research*, 45(3):378–394, 1997.
- [38] N. Garg, G. Konjevod, and R. Ravi, “A polylogarithmic approximation algorithm for the group steiner tree problem,” *Journal of Algorithms*, 37(1):66–84, 2000.
- [39] B. Bontoux, C. Artigues, and D. Feillet, “A memetic algorithm with a large neighborhood crossover operator for the generalized traveling salesman problem,” *Computers & Operations Research*, 37(11):1844–1852, 2010.
- [40] D. Karapetyan and G. Gutin, “Lin–Kernighan heuristic adaptations for the generalized traveling salesman problem,” *European Journal of Operational Research*, 208(3):221–232, 2011.
- [41] K. Helsgaun, “Solving the equality generalized traveling salesman problem using the Lin–Kernighan–Helsgaun algorithm,” *Mathematical Programming Computation*, 7(3):269–287, 2015.
- [42] P. C. Pop, O. Matei, and C. Sabo, “A new approach for solving the generalized traveling salesman problem,” in M. J. Blesa, C. Blum, G. Raidl, A. Roli, and M. Sampels, eds., *Hybrid Metaheuristics, Lecture Notes in Computer Science*, 2010, pp. 62–72, Springer.
- [43] S. L. Smith and F. Imeson, “GLNS: An effective large neighborhood search heuristic for the generalized traveling salesman problem,” *Computers & Operations Research*, 87:1–19, 2017.
- [44] K. Elbassioni, A. V. Fishkin, and R. Sitters, “Approximation algorithms for the Euclidean traveling salesman problem with discrete and continuous neighborhoods,” *International Journal of Computational Geometry & Applications*, 19(02):173–193, 2009.
- [45] J. Faigl, J. Deckerova, and P. Vana, “Fast heuristics for the 3D multi-goal path planning based on the generalized traveling salesman problem with neighborhoods,” *IEEE Robotics and Automation Letters*, 4:2439–2446, 2019.
- [46] D. Feillet, P. Dejax, and M. Gendreau, “Traveling salesman problems with profits,” *Transportation Science*, 39(2):188–205, 2005.
- [47] E. Balas, “The prize collecting traveling salesman problems,” *Networks*, 19:621–636, 1989.
- [48] G. Laporte and S. Martello, “The selective travelling salesman problem,” *Discrete Applied Mathematics*, 26(2):193–207, 1990.
- [49] B. L. Golden, L. Levy, and R. Vohra, “The orienteering problem,” *Naval Research Logistics (NRL)*, 34(3):307–318, 1987.
- [50] G. Hollinger, U. Mitra, and G. Sukhatme, “Autonomous data collection from underwater sensor networks using acoustic communication,” in *IROS*, 2011, pp. 3564–3570.
- [51] J. Faigl and G. A. Hollinger, “Autonomous data collection using a self-organizing map,” *IEEE Transactions on Neural Networks and Learning Systems*, 29(5):1703–1715, 2018.
- [52] J. Faigl and P. Vana, “Self-organizing map for data collection planning in persistent monitoring with spatial correlations,” in *IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, 2016, pp. 3264–3269.
- [53] J. Faigl, “Data collection path planning with spatially correlated measurements using growing self-organizing array,” *Applied Soft Computing*, 75:130–147, 2019.
- [54] F. Mufalli, R. Batta, and R. Nagi, “Simultaneous sensor selection and routing of unmanned aerial vehicles for complex mission plans,” *Computers & Operations Research*, 39(11):2787–2799, 2012.
- [55] A. Gunawan, H. C. Lau, and P. Vansteenwegen, “Orienteering Problem: A survey of recent variants, solution approaches and applications,” *European Journal of Operational Research*, 255(2):315–332, 2016.
- [56] P. Vansteenwegen, W. Souffriau, and D. V. Oudheusden, “The orienteering problem: A survey,”



- European Journal of Operational Research, 209(1):1–10, 2011.
- [57] L. Evers, T. Dollevoet, A. I. Barros, and H. Monsuur, “Robust UAV mission planning,” *Annals of Operations Research*, 222(1):293–315, 2012.
- [58] R. Penicka, J. Faigl, P. Vana, and M. Saska, “Dubins orienteering problem,” *IEEE Robotics and Automation Letters*, 2(2):1210–1217, 2017.
- [59] J. Faigl and P. Vana, “Self-organizing map for the curvature-constrained traveling salesman problem,” in *International Conference on Artificial Neural Networks (ICANN)*, 2016, pp. 497–505.
- [60] G. Best, J. Faigl, and R. Fitch, “Multi-robot path planning for budgeted active perception with self-organising maps,” in *IROS*, 2016, pp. 3164–3171.
- [61] J. Faigl, R. Penicka, and G. Best, “Self-organizing map-based solution for the orienteering problem with neighborhoods,” in *IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, 2016, pp. 1315–1321.
- [62] J. Faigl, “On self-organizing maps for orienteering problems,” in *International Joint Conference on Neural Networks (IJCNN)*, 2017, pp. 2611–2620.
- [63] G. Best, J. Faigl, and R. Fitch, “Online planning for multi-robot active perception with self-organising maps,” *Autonomous Robots*, 42(4):715–738, 2018.
- [64] C. Archetti, F. Carrabs, and R. Cerulli, “The set orienteering problem,” *European Journal of Operational Research*, 267(1):264–272, 2018.
- [65] R. Penicka, J. Faigl, and M. Saska, “Variable neighborhood search for the set orienteering problem and its application to other orienteering problem variants,” *European Journal of Operational Research*, 276(3):816–825, 2019.
- [66] L. E. Dubins, “On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents,” *American Journal of mathematics*, pp. 497–516, 1957.
- [67] S. Edelkamp and E. Plaku, “Multi-goal motion planning with physics-based game engines,” in *Computational Intelligence and Games*, CIG, 2014, pp. 1–8.
- [68] S. Rashidian, E. Plaku, and S. Edelkamp, “Motion planning with rigid-body dynamics for generalized traveling salesman tours,” in *Motion in Games*, 2014, pp. 87–96.
- [69] E. Plaku, S. Rashidian, and S. Edelkamp, “Multi-group motion planning in virtual environments,” *Computer Animation and Virtual Worlds*, 29(6):e1688, 2018.
- [70] P. Vana, J. Faigl, and J. Slama, “Emergency landing aware surveillance planning for fixed-wing planes,” in *European Conference on Mobile Robots (ECMR)*, 2019, pp. 1–6.
- [71] D. Le and E. Plaku, “Multi-robot motion planning with dynamics via coordinated sampling-based expansion guided by multi-agent search,” *IEEE Robotics Autom. Lett.*, 4(2):1868–1875, 2019.
- [72] S. Edelkamp, D. Golubev, and C. Greulich, “Solving the physical vehicle routing problem for improved multi-robot freespace navigation,” in G. Friedrich, M. Helmert, and F. Wotawa, eds., *KI*, vol. 9904 of *Lecture Notes in Computer Science*, 2016, pp. 155–161, Springer.
- [73] S. Edelkamp and J. Lee, “Multi-robot multi-goal motion planning with time and resources,” in K. Althoefer, J. Konstantinova, and K. Zhang, eds., *TAROS*, vol. 11649 of *Lecture Notes in Computer Science*, 2019, pp. 288–299, Springer.
- [74] R. E. Fikes and N. J. Nilsson, “STRIPS: a new approach to the application of theorem proving to problem solving,” *Artificial Intelligence*, 2(3):189–208, 1972.
- [75] S. Edelkamp, M. Lahijanian, D. Magazzeni, and E. Plaku, “Integrating temporal reasoning and sampling-based motion planning for multigoal problems with dynamics and time windows,” *IEEE Robotics Autom. Lett.*, 3(4):3473–3480, 2018.
- [76] P. Toth and D. Vigo, eds., *The vehicle routing problem*, Society for Industrial and Applied Mathematics, USA, 2001.
- [77] M. Fox and D. Long, “PDDL2.1: an extension to PDDL for expressing temporal planning domains,” *J. Artif. Intell. Res.*, 20:61–124, 2003.
- [78] S. Edelkamp, E. Plaku, and Y. Warsame, “Monte-carlo search for prize-collecting robot motion planning with time windows, capacities, pickups, and deliveries,” 2019, pp. 154–167, Springer.
- [79] J. Faigl and L. Preucil, “Inspection planning in the polygonal domain by self-organizing map,” *Applied Soft Computing*, 11(8):5028–5041, 2011.
- [80] S. Edelkamp, M. Pomarlan, and E. Plaku, “Multiregion inspection by combining clustered traveling salesman tours with sampling-based motion planning,” *IEEE Robotics Autom. Lett.*, 2(2):428–435, 2017.
- [81] S. Edelkamp and Z. Yu, “Watchman routes for robot inspection,” in K. Althoefer, J. Konstantinova, and K. Zhang, eds., *TAROS*, vol. 11650 of *Lecture Notes in Computer Science*, 2019, pp. 179–190, Springer.
- [82] P. Vana, J. Slama, and J. Faigl, “Surveillance planning with safe emergency landing guarantee for fixed-wing aircraft,” *Robotics and Autonomous Systems*, 133:103644, 2020.
- [83] P. Vana, J. Slama, J. Faigl, and P. Pačes, “Any-time trajectory planning for safe emergency landing,” in *IROS*, 2018, pp. 5691–5696.
- [84] X. Hu, B. Pang, F. Dai, and K. H. Low, “Risk assessment model for uav cost-effective path planning in urban environments,” *IEEE Access*, 8:150162–150173, 2020.
- [85] K. Dalamagkidis, K. Valavanis, and L. Piegl, “On integrating unmanned aircraft systems into the national airspace system, international series on intelligent systems, control, and automation: Science and engineering,” *Science and Engineering*, 36, 2009.
- [86] A. la Cour-Harbo, “Ground impact probability distribution for small unmanned aircraft in ballistic descent,” in *International Conference on Unmanned Aircraft Systems (ICUAS)*, 2020, pp. 1442–1451.
- [87] S. Primatesta, G. Guglieri, and A. Rizzo, “A risk-aware path planning strategy for uavs in urban environments,” *Journal of Intelligent & Robotic Systems*, 95(2):629–643, 2019.
- [88] S. Primatesta, A. Rizzo, and A. la Cour-Harbo, “Ground risk map for unmanned aircraft in urban environments,” *Journal of Intelligent & Robotic Systems*, 97(3):489–509, 2020.
- [89] S. Koenig, M. Likhachev, and D. Furcy, “Lifelong planning A\*,” *Artificial Intelligence*, 155(1):93–146, 2004.
- [90] M. Otte and E. Frazzoli, “RRTx: Real-time motion planning/replanning for environments with unpredictable obstacles,” *International Journal of Robotics Research*, 35(7):797–822, 2016.

# SM2P: Towards a Robust Co-Pilot System for Helicopter EMS

Ian Mallett<sup>1</sup>, Marcus Hoerger<sup>1</sup>, Surabhi Gupta<sup>2</sup>, Nisal Jayalath<sup>2</sup>, Felipe Trevizan<sup>1</sup>, Andrew Hunt<sup>2</sup>,  
Hanna Kurniawati<sup>1</sup>, Christophe Guettier<sup>3</sup>

<sup>1</sup>School of Computing, Australian National University  
{hanna.kurniawati, felipe.trevizan}@anu.edu.au ; {ian.p.mallett, hoergems}@gmail.com

<sup>2</sup>Safran Electronics and Defense Australasia  
{surabhi.gupta, andrew.hunt}@safrangroup.com ; nisaljayalath@gmail.com

<sup>3</sup>Safran Electronics and Defense  
{christophe.guettier}@safrangroup.com

## Abstract

This paper presents our preliminary work towards developing robust decision-making components for an automated co-pilot system in Helicopter Emergency Medical Services (HEMS). Specifically, in this paper, we focus on the integrated mission-motion planning framework for such a mission, and propose Stochastic-based Mission-Motion Planner (SM2P). SM2P frames the mission planning as a Stochastic Shortest Path (SSP), and the trajectory planning as a Partially Observable Markov Decision Processes (POMDPs). Each planning problem is solved using state-of-the-art (approximate) solvers that can (re-)compute a good strategy that accounts for the various uncertainty plaguing an HEMS mission, on-line, within seconds for a typical HEMS mission. The use of SSP in mission planning allows SM2P to account for the non-deterministic effects of actions due to problem abstraction and limited condition in which the HEMS mission often takes place, while the use of POMDP allows SM2P to account for both the non-deterministic and partially observable nature of the operation as more information are perceived. Preliminary results on a simulation of three different HEMS scenarios in Corsica region indicates that SM2P reaches a success rate of over 95% in all scenarios.

## Introduction

Helicopter Emergency Medical Services (HEMS) is among the most challenging and dangerous air operations (Hart March 2017). Its accident rate is more than 28 times higher than that of commercial aircraft (Holland and Cooksley 2005). HEMS missions are time-critical and arrive with little to no forewarning, making extensive planning difficult. These conditions are known to increase the chance of catastrophic mistakes (nsa October 2013). Despite these difficulties, most HEMS are performed without a co-pilot, which means a single person—the pilot—is wholly responsible for: guidance and navigation, detecting hazards to avoid, evaluating the suitability of landing zones and the ability to takeoff after landing, maintaining situational and spatial awareness of the terrain and proximate obstacles during manoeuvres in ground proximity, planning and re-planning routes given weather or mission parameter changes, as well

as flying the helicopter. To help reduce the pilot's burden, in this paper, we present our preliminary work in developing a core planning component of an AI-based co-pilot for Helicopter Emergency Medical Services (HEMS).

Pilot Assistance Systems for helicopters, including auto-pilot, exist. For instance, DLR and ONERA, together with Eurocopter and Airbus, have developed multiple Pilot Assistance Systems (Lantzch et al. 2012; Le Blaye 2003; Lüken and Korn 2007). However, they are not suitable for HEMS, as they do not account for uncertainty and terrain characteristics, which are important if they were to provide manoeuvring strategies with ground proximity. Recently, Choudhury, et.al. have proposed a full-scale autonomous-flight system for HEMS missions (Choudhury et al. 2019). They identify uncertainty as a critical issue in HEMS mission, but to keep computation cost low, they address uncertainty by inflating risks: Deterministic planning is performed in a model of the world where risks have been inflated (e.g., obstacles have been enlarged) and assumes that no uncertainty remains. This approach works well when the feasible solution space is large. But, when the set of feasible solutions is small, inflating risks may remove all possible solutions, causing the mission to be deemed infeasible. Unfortunately, such scenarios are common in HEMS. For instance, when the patients to be picked up is in a bushfire area or mountainous areas—bushfires are frequent in many parts of the world, including in Australia, California, the Borneo Island in Indonesia, and South of France.

To alleviate the above difficulties, we propose to develop an AI-based co-pilot system that suggests good decisions within a limited computation time, while accounting for imperfect information about the mission and operating environment. Specifically, we propose an integrated mission and trajectory planner, where the mission planner is modelled as a Stochastic Shortest Path (SSP) and the trajectory planner is modelled as a Partially Observable Markov Decision Processes (POMDPs) problem. We call this approach Stochastic-based Mission-Motion Planner (SM2P).

The mission planner in SM2P abstracts the helicopter's operation area. It receives information from mission command and makes high-level strategy to accomplish the given mission. An example of this strategy is which victim to pick-

up first and the areas the helicopter needs to fly to in order to pick-up the victim. The trajectory planner receives a high-level guide from the mission planner and information about the environment from the helicopter's sensors and the pilot. It computes a motion strategy (e.g., collision-free trajectories, selection on landing area, etc.). This motion strategy is provided to the pilot. SM2P assumes the pilot will execute the provided strategy with some potential deviations from the suggested plan.

The mission and trajectory planners of SM2P communicate closely. When the mission planner receives an update on the mission, which require changes to its original mission plan, it will inform the trajectory planner, which will in turn update its motion strategy. Similarly, when the trajectory planner finds a certain guide is not feasible, it informs the mission planner, which in turn will find an alternative high-level strategy.

Preliminary results on three different HEMS scenarios in Corsica region near St Florent and Bastia indicate SM2P has a success rate of over 95% with 10 minutes offline computation per region, which can be computed once for HEMS missions in the region, 40 seconds offline computation per HEMS mission, and 5 seconds online computation.

## Background and Related Work

### Background on SSPs

A **Stochastic Shortest Path problem** (SSP) (Bertsekas and Tsitsiklis 1991) is a tuple  $\mathbb{S} = \langle S, s_0, G, A, P, C \rangle$  in which:  $S$  is the finite set of states;  $s_0 \in S$  is the initial state;  $G \subseteq S$  is the non-empty set of goal states;  $A$  is the finite set of actions;  $P(s'|s, a)$  is the probability of reaching  $s'$  after action  $a$  is applied in state  $s$ ; and  $C(s, a) \in \mathbb{R}_{>0}$  is the immediate cost of applying action  $a$  in state  $s$ . For simplicity, we assume  $s_0 \notin G$  and we represent by  $A(s)$  the actions applicable in state  $s$ .

A solution to an SSP is a *policy*  $\pi$ , i.e., a mapping from states to actions, and the optimal solution is any policy  $\pi^*$  that (i) reaches  $G$  from  $s_0$  with probability 1; and (ii) has minimum total expected cost of reaching the goal from  $s_0$ , where the expected cost of a policy satisfying (i) can be computed using the following set of equations:

$$V_\pi(s) = C(s, \pi(s)) + \sum_{s' \in S} P(s'|s, \pi(s))V_\pi(s')$$

for all  $s \in S \setminus G$  and  $V_\pi(s_g) = 0$  for all  $s_g \in G$ .

In this work, we consider SSPs with dead ends, that is, we do not assume that, for all  $s \in S$ , the probability for reaching  $G$  from  $s$  is 1. We use the fixed-cost approach for dead ends, i.e., if  $s$  is a dead end, then  $V_\pi(s)$  is defined as  $d$ , where  $d$  is a large positive penalty for not reaching the goal. This approach allows us to transform the original SSP  $\mathbb{S}$  into a new SSP  $\mathbb{S}'$  without dead ends in which there is an action that deterministically transitions from any state  $s$  to  $G$  with cost  $d$  (Mausam and Kolobov 2012). Notice that our approach can be trivially applied to different approaches to handling dead ends that optimizes different metrics relating cost and probability of reaching dead ends (e.g., see (Trevizan, Teichteil-Königsbuch, and Thiébaux 2017)).

### Background on POMDP

Formally a POMDP is a tuple  $\langle S, A, O, T, Z, R, \gamma \rangle$ , where  $S$ ,  $A$  and  $O$  are the state, action and observation spaces of the robot.  $T$  and  $Z$  model the uncertainty in the effect of taking actions and receiving observations as conditional probability functions  $T(s, a, s') = p(s'|s, a)$  and  $Z(s', a, o) = p(o|s', a)$ , where  $s, s' \in S$ ,  $a \in A$  and  $o \in O$ .  $R(s, a)$  models the reward the robot receives when performing action  $a$  from  $s$  and  $0 < \gamma < 1$  is a discount factor. Due to uncertainties in the effect of performing actions and receiving observations, the true state of the robot is only partially observable. Hence, instead of planning with respect to states, the robot plans with respect to probability distributions  $b \in \mathcal{B}$  over the state space, called beliefs, where  $\mathcal{B}$  is the set of all probability distributions over  $S$ . The solution of a POMDP is an optimal policy  $\pi^*$ , a mapping from beliefs to actions  $\pi^* : b \mapsto a$  such that the robot maximises the expected discounted future reward when following  $\pi^*$ . Once  $\pi^*$  has been computed, it can be used as a feedback-controller: Given the current belief  $b$ , the robot performs  $\pi^*(b)$ , receives an observation  $o \in O$  and updates its belief according to  $b' = \tau(b, a, o)$ , where  $\tau$  is the Bayesian belief update function. The value achieved by a policy  $\pi$  at a particular belief  $b$  can be expressed as

$$V_\pi(b) = R(b, \pi(b)) + \gamma \int_{o \in O} Z(b, \pi(b), o) V_\pi(\tau(b, \pi(b), o)) do \quad (1)$$

where  $R(b, a) = \int_{s \in S} R(s, a) b(s) ds$  and  $Z(b, a, o) = \int_{s' \in S} Z(s', a, o) \int_{s \in S} T(s, a, s') b(s) ds ds'$ . The optimal policy  $\pi^*$  is then the policy that satisfies  $\pi^*(b) = \operatorname{argmax}_\pi V_\pi(b)$ .

### Overall Framework

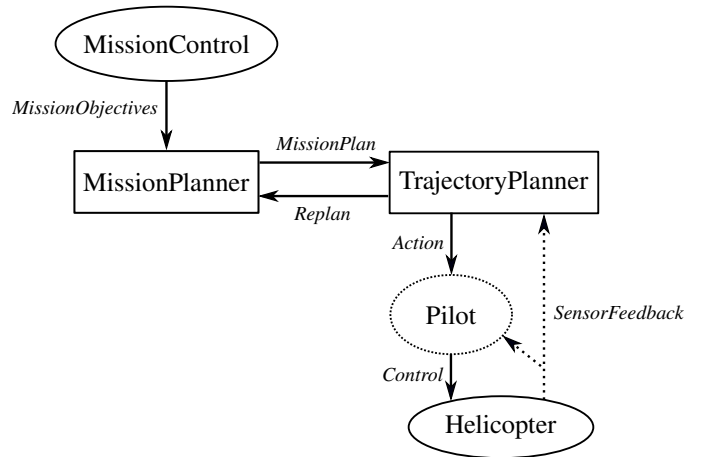


Figure 1: Overview of the planning architecture.

Most HEMS missions require the co-pilot system to solve a large and complex planning problem in three senses: Large hybrid state space, long planning horizon, and complex dynamics. The planner must operate on both continuous and

discrete variables, as it must identify which victims to pick-up and in which order, which medical facility to go to, so as to maximise the number of victims being saved, and navigation guidance in confined and imperfectly known areas that respect fuel and cloud ceiling requirements. The problem requires a long planning horizon. For instance, in the HEMS scenarios around Saint Florent and Bastia, for a helicopter to move from its starting position to the nearest hospital at average speed, it needs at least 60 planning steps. When the helicopter needs to pick-up a victim first, this planning step will substantially increase. Last but not least, helicopter dynamics is known to be complex. In this paper, we focus on the first two issues, and simplify the helicopter dynamics as a second-order discrete-time stochastic model according to:

$$f(\phi_t, \lambda_t, \theta_t, h_t, \nu_t) = \begin{bmatrix} \phi_t + \Delta \nu_t \cos \theta_t \\ \lambda_t + \Delta \nu_t \sin \theta_t \\ h_t + \Delta(\delta_h + e_h) \\ \theta_t + \Delta(\delta_\theta + e_\theta) \\ \nu_t + \Delta(\alpha + e_\nu) \end{bmatrix} \quad (2)$$

where  $\phi_t, \lambda_t, \theta_t, h_t, \nu_t \in \mathbb{R}$  are the latitude, longitude, yaw-orientation, elevation and forward velocity of the helicopter.  $\delta_h$  and  $\delta_\theta$  are fixed climb and turn rates, whereas  $\alpha$  is a fixed acceleration.  $\Delta$  is a fixed control duration and  $e_h, e_\theta, e_\nu$  represent random control errors that are drawn from zero-mean Gaussian distributions.

Figure 1 presents the overall framework of our integrated mission and trajectory planners. The mission planner computes the high level strategy on which victims to pick-up and which order should they be picked-up. These strategies are transformed into navigation guides that respect the estimated fuel usage and flying requirements under the estimated weather conditions. These strategies are computed, while respecting fuel and weather condition requirements.

To construct navigation guides, the mission planner constructs an abstraction of the operation area of HEMS. Suppose  $\mathcal{W} \subseteq \mathbb{R}^3$  is the bounded operational space of the HEMS mission. The space  $\mathcal{W}$  contains the terrain of the area. The mission planner uses the SPArse Roadmap Spanner (Dobson, Krontiris, and Bekris 2013), to construct a sparse graph  $\mathcal{G}(\mathcal{V}, \mathcal{E})$  that captures the connectivity of  $\mathcal{W}$  well. Each vertex in  $\mathcal{V}$  is associated with a position sampled from  $\mathcal{W}$ , while an edge  $\overline{vv'} \in \mathcal{E}$  means there is a collision-free straight line-path for the helicopter from the positions represented by  $v \in \mathcal{V}$  to  $v' \in \mathcal{V}$ . The environment  $\mathcal{W}$  is then decomposed into Voronoi regions, where the points are the positions associated with the vertices  $\mathcal{V}$ . The navigation guide provided by the mission planner to the trajectory planner is then a mapping from one Voronoi region to the neighbouring Voronoi region to visit, so as to achieve the mission objective.

The trajectory planner is an on-line POMDP planner and receives observations about the environment around its current location from the helicopter's sensors and from the pilot. Such observations will enable the trajectory planner to realise if moving into a certain Voronoi region has now become very difficult or even infeasible. For example, if a no-fly zone were to suddenly appear due to the spread of fire in bush-fire areas, the navigation guide from the mission

planner would be rendered invalid. When this situation happens, the trajectory planner notifies the mission planner. The mission planner will identify the affected components of  $\mathcal{G}$ , modify the graph, and re-plans with respect to the modified graph. The new navigation guide is then passed to the trajectory planner.

The subsequent sections provide details of the mission and trajectory planners.

## Mission Planner Details

The mission planner provides a high level policy to guide the trajectory planner to achieve its mission objectives, primarily guiding it to the additional victim to be rescued, and to the hospital. The policy is made with respect the helicopter's position, current fuel level, and the height of an overcast cloud ceiling. Under Visual Flight Rules conditions (VFR), the helicopter must not cross this ceiling unless absolutely necessary. In the experiment, we follow the VFR rules followed in France.

We assume that, in the case where there are one or more victims to pick up en route, that there is space for only one more victim on board, and that when arriving to pick up a victim, it may turn out that it is impossible to get the victim on board. Given this, the mission planner's goal is to either successfully pick up a victim, or to try to pick them all up.

We use the graph  $\mathcal{G}(\mathcal{V}, \mathcal{E})$  obtained using SPArse Roadmap Spanner (Dobson, Krontiris, and Bekris 2013) to define the SSP  $\langle S_{MS}, s_0, G, A_{MS}, P, C \rangle$  where:

1. The mission planner state space  $S_{MS}$  is the product of  $\mathcal{V}$  with a discretised set of fuel levels  $F$  and discretised cloud heights  $H_c$ . In the case where there are one or more victims to pick up en route (i.e.,  $|\Upsilon| > 1$ ), we include a variable `at-capacity`  $\in \{\top, \perp\}$ , and for each victim  $\nu \in \Upsilon$ , we include `attempted` $_\nu \in \{\top, \perp\}$ , representing whether the pilot has tried to pick up that victim.
2. The initial state  $s_0 \in S_{MS}$  is the initial position of the helicopter, fuel level equivalent to a full tank, and an initial cloud height.
3.  $G \subset S_{MS}$  is the set of states where the helicopter is at the hospital and, when there are extra victims in the scenario, either an extra victim is on board (`at-capacity` =  $\top$ ) or  $\forall \nu, \text{attempted}_\nu = \top$ .
4. The set of possible actions  $A_{MS}$  is  $\{\text{move}(v, v') \mid \overline{vv'} \in \mathcal{E}\} \cup \{\text{pick-up}_\nu \mid \nu \in \Upsilon\}$ . Let  $v_s \in \mathcal{V}$  be the helicopter's location in state  $s \in S_{MS}$ . If `at-capacity` equals  $\top$  in  $s$ , then the set of applicable actions  $A_{MS}(s)$  is  $\{\text{move}(v_s, v') \mid \overline{v_s v'} \in \mathcal{E}\}$ , otherwise  $A_{MS}(s)$  also includes  $\{\text{pick-up}_\nu \mid \nu \in \Upsilon, v_s \text{ is the closest vertex to } \nu\}$ .
5. The probability transition function  $P(s' \mid s, a)$  is such that each component of  $s$  is changed *independently* as follows:
  - **Location:** if  $a = \text{move}(v_s, v')$ , then the location at the new state  $s'$  is  $v'$  with probability 0.9 (i.e.,  $P(v_{s'} = v') = 0.9$ ) and, with probability 0.1, the location does not change (i.e.,  $P(v_{s'} = v_s) = 0.1$ ). The location remains the same for all non moving actions.

- **Fuel level:** for  $a = \text{move}(v_s, v')$ , the fuel level decreases deterministically, assuming the helicopter moves from  $v_s$  to  $v'$  at 150 km/h, using fuel at a steady rate of 4kg per minute. For  $\text{pick-up}_\nu$ , the fuel decreases by 20kg on a success, or 4kg on a failure (we assume the pickup takes 5 or 1 minutes).
  - **Cloud ceiling:** let  $c_s$  and  $c_{s'}$  be the cloud ceiling heights in states  $s$  and  $s'$ .  $P(c_{s'} = c_s) = 0.98$ ,  $P(c_{s'} = c_s + 100m) = 0.01$  and  $P(c_{s'} = c_s - 100m) = 0.01$ . If  $c_{s'}$  would be outside  $[600m, 900m]$ , it is set to the nearest bound.
  - **Victims:** for  $a = \text{move}(v_s, v')$ ,  $\text{at-capacity}$  and  $\text{attempted}_\nu$  are not changed. If  $a = \text{pick-up}_\nu$ ,  $P(\text{attempted}_\nu = \top) = 1$ ,  $P(\text{at-capacity} = \top) = 0.95$ ,  $P(\text{at-capacity} = \perp) = 0.05$ .
6. The cost function represents the expected time it takes to complete an action, with a 15 minute penalty for entering the cloud layer. Moreover,  $C(s, \text{pick-up}_\nu) = 288$ ,  $C(s, \text{move}(v_s, v'))$  is the time in seconds to travel from  $v_s$  to  $v'$  at 150 km/h, plus 900 if  $v_s$  is above the current cloud height.

Our probability transition function  $P(s'|s, a)$  defined above implies that moving from a location  $v$  to  $v'$  follows a geometric distribution with  $p = 0.9$  since the action  $\text{move}$  succeeds with probability 0.9 and fails by staying at the same location  $v$  with probability 0.1. While successive executions of the same  $\text{move}$  action might not change the helicopter's location, it changes the current state  $s \in S_{\text{MS}}$  of the system due to the deterministic fuel consumption (the fuel component of the state monotonically decreases) and potential change in the cloud ceiling.

## Trajectory Planner Details

The trajectory planner's purpose is to compute a low-level motion strategy for the pilot to successfully complete the mission on-line, using the high-level mission-planner strategy as a navigation guide. During run-time the trajectory planner uses sensor information from the helicopter to construct and maintain a local map of the environment that includes the terrain, as well as obstacles (such as trees, power lines or buildings), cloud-ceiling and possible no-fly zones within a bounded region  $\mathcal{W}_{\text{local}} \subset \mathcal{W}$  around the helicopter. This local map is updated after every step (in this paper we assume that the geometries and locations of the obstacles within  $\mathcal{W}_{\text{local}}$  are perfectly known to the trajectory planner). The low-level motion strategy is then computed with respect to the current local map and the stochastic dynamics of the helicopter.

Due to imperfect controls and sensor information the true state of the helicopter is only partially observed. Thus, we maintain a belief  $b_t \in \mathcal{B}$  over the current state of the helicopter and formulate the problem of computing a low-level motion strategy for the pilot as a POMDP. To increase efficiency, we further decompose the problem into two sub-problems, both formulated as separate POMDPs: Navigating and landing/take-off. Details on the POMDP formulation of both sub-problems are provided in Sub-Sections **POMDP-**

**Formulation of the Navigation Problem** and **POMDP-Formulation of the Landing Problem** respectively. For the first sub-problem, the trajectory planner computes a policy to navigate to the Voronoi regions in the environment as specified by the mission planner policy. For the second sub-problem, we first construct a set of possible landing zones across the environment off-line (details on how these landing zones are constructed are provided in Sub-Section **Constructing the Landing Zones**). At run-time, the trajectory planner then determines the closest landing zone to the victim and computes a motion strategy to safely touch-down at the landing zone in order to pick-up the victim. During run-time the trajectory planner switches between both POMDP problems depending on the current mission planner objective.

To compute a policy from the current belief  $b_t \in \mathcal{B}$ , we use ABT (Kurniawati and Yadav 2013), one of the fastest Monte-Carlo-Tree-Search based on-line solvers. A summary of this method is presented here for completeness. Starting from  $b_t$  (we represent beliefs as sets of particles) ABT approximates the optimal policy by constructing and evaluating a belief-tree, whose nodes represent beliefs and edges represent pairs of actions and observations. To construct the belief-tree, ABT samples *episodes*, that is, sequences of state-action-observation-reward quadruples, starting from the current belief and associates the states of the episode with nodes in the belief-tree. To select actions during the episode sampling process, ABT uses Upper Confidence Bounds1 (UCB1) (Auer, Cesa-Bianchi, and Fischer 2002) which ensures asymptotic convergence towards the optimal policy. The states, observations and reward of an episode are sampled from a generative model that encodes the transition, observation and reward functions. An episode is expanded until either a terminal state is reached, or the episode reaches a belief-node for which there are actions that haven't been visited before. If the first terminating condition occurs, ABT backpropagates the sampled reward-trajectory to update the estimates of  $\hat{Q}(b, a)$  for each belief  $b \in \mathcal{B}$  associated to a state in the episode, where  $Q(b, a)$  is the value of executing action  $a \in A$  from  $b$  and continuing optimally afterwards. Otherwise, ABT first estimates the value of the last belief node by simulating a rollout strategy from the last state of the episode and then continues backpropagating the sampled reward-trajectory as described above.

Once the planning time for the current step is over, ABT selects an action according to  $\pi(b_t) = \arg\max_{a \in A} \hat{Q}(b_t, a)$ . After executing the action and receiving an observation, the current belief is updated (we use a Sequential-Importance-Resampling (SIR) particle filter (Arulampalam et al. 2002) to update the belief) and planning continues from the updated belief.

To embed the navigation guides from the mission planner into the POMDP policy search, we construct a rollout strategy that encodes the mission planner strategy. This is done as follows: Suppose  $b_t$  is the current belief and  $\bar{s} \in S$  is the mean state of  $b_t$ . Using  $\bar{s}$ , we estimate the Voronoi region – with associated vertex  $v_1 \in \mathcal{V}$  – the helicopter is currently located in and query the mission planner policy.

This provides us with a high-level action that can either be `move`( $v_1, v_2$ ), i.e. to navigate from the Voronoi region associated with  $v_1$  to the Voronoi region associated with  $v_2 \in \mathcal{V}$ , or `pick-up` $_{v_1}$ , i.e. to pick up the victim located in the current Voronoi region. In the first case, we compute a motion strategy to reach the Voronoi region associated to  $v_2$ , assuming deterministic effects of actions. In the second case we assume deterministic dynamics too, but compute a motion strategy to reach a landing zone near the victim.

This allows us to guide the search towards achieving the high-level strategy computed by the mission planner, while simultaneously planning with respect to the stochastic helicopter dynamics and the local environment around the helicopter. Note that we only query the mission planner after the current belief  $b_t$  has been updated. Within a planning step (that is, during the policy computation from  $b_t$ ), the high-level action remains constant.

### POMDP-Formulation of the Navigation Problem

**State, Action and Observation Spaces** The state space of the helicopter is defined as the cross-product of four components  $S = \mathbb{R}^3 \times \Pi \times [0, \nu_{max}] \times \mathbb{R}^+ \times H_c$ , where the first component is the 3D real-vector space consisting of the latitude, longitude and elevation of the helicopter above median sea level.  $\Pi = [-180.0, 180.0]$  is the set of yaw-orientations (in degrees) of the helicopter, whereas  $[0, \nu_{max}]$  are the minimum and maximum forward-velocities (in  $m/s$ ) of the helicopter. The component  $\mathbb{R}^+$  is the set of all fuel loads of the helicopter, whereas  $H_c$  is the discrete set of cloud-ceiling heights.

The action space of the helicopter is defined as  $A = \{\text{accelerate}, \text{decelerate}, \text{climb}, \text{descend}, \text{turnLeft}, \text{turnRight}\}$ . The `accelerate` and `decelerate` actions set the acceleration  $\alpha$  in eq.(2) to a fixed positive/negative value. Similarly, the `climb`/`descend` and `turnLeft`/`turnRight` actions set the climb rate  $\delta_h$  and turn rate  $\delta_\theta$  in eq.(2) to fixed positive/negative values respectively.

We assume that the helicopter is equipped with two types of sensors: A localization sensor which provides information regarding the current latitude, longitude and elevation of the helicopter and a gyroscope which provides information regarding the helicopter's yaw-orientation. More formally, the observation space is defined as  $O = \mathbb{R}^3 \times \Pi$ , where the first component describes the latitude, longitude and elevation of the helicopter  $\Pi$  is defined as above.

**Transition Function** To model the transition dynamics of the helicopter, we use the second-order stochastic dynamic system defined in eq.(2) given the latitude  $\phi$ , longitude  $\lambda$ , elevation  $h$  and yaw-orientation  $\theta$  associated to the current state of the helicopter. Additionally, we assume that the fuel load of the helicopter decreases deterministically by a constant rate (in our experiments, we assume a fuel consumption of  $\frac{1}{3}kg$  per time step).

**Observation Function** The observation model of the helicopter is defined as

$$o_t = [\phi_t; \lambda_t; h_t; \theta_t]^T + e_o \quad (3)$$

where  $\phi_t, \lambda_t, h_t, \theta_t$  are the latitude, longitude, elevation and yaw angle components of the state.  $e_o \in \mathbb{R}^4$  is a random vector drawn from a zero-mean multivariate Gaussian distribution representing sensor noise.

**Reward function** To encode the objective of reaching the hospital, the helicopter receives a reward of 10,000 when entering a goal area around the hospital (modelled as a sphere with radius 1,000m around the location of the hospital). Once the helicopter enters this goal area, the mission is considered successful. If the helicopter collides with the terrain or an obstacle, it receives a penalty of -50,000 and the mission is considered as unsuccessful. To encourage the helicopter to reach the goal as quickly as possible, it receives a penalty of -10 at every step. Since crossing the cloud ceiling is considered dangerous due to low visibility, the helicopter receives a penalty of -10 for every state where it is located above the cloud ceiling, discouraging the helicopter from crossing the cloud layer unless necessary (for instance to avoid an obstacle).

### POMDP-Formulation of the Landing Problem

Once the trajectory planner receives a `pick-up` $_v$  action from the mission planner, it switches to the landing problem. For this problem the task is to safely navigate to a landing zone close to the injured person in the environment and perform a landing maneuver.

The POMDP formulation of this problem is similar to the one for the navigation problem described in the previous section, with some notable differences: We extend the action space of the helicopter with an additional `land` action whose purpose is to perform a vertical touch-down at the current location of the helicopter to pick-up the victim. For this action, the helicopter enters a terminal state and receives a reward of 10000 if the following conditions are met: a.) The helicopter is located above a landing zone area (the method to define and construct a landing zone is described in the next subsection), b.) The vertical distance between the helicopter and the terrain is within 75m and c.) The forward velocity of the helicopter is smaller than 10m/s. If at least one of these conditions is not satisfied, the helicopter enters a terminal state too, but receives a penalty of -50000 and the mission is considered unsuccessful. Note that we assume that in case the `land` action is successful, the injured person is automatically picked up and the trajectory planner switches back to the navigation problem.

**Constructing the Landing zones** In order for the helicopter to successfully pick up an injured person, it has to determine areas in the environment that are suitable for landing, that is, areas that are sufficiently large and flat. To construct such areas, we use a simple geometric approach: Suppose the terrain is represented by a triangular mesh, i.e. a set  $U$  of triangles. For each triangle  $u \in U$ , we compute its slope  $s_u$  via  $s_u = \arccos(n_u \cdot z / (\|n_u\| \|z\|))$ , where  $n_u$  is the normal vector of triangle  $u$ ,  $z = (0, 0, 1)^T$  and " $\cdot$ " denotes the dot product. If  $s_u$  yields a value larger than a given threshold  $s_{max}$  (in our experiments we use  $s_{max} = 9deg$ ), we remove the triangle  $u$  from  $U$ . In other words, we remove triangles from  $U$  that are too "steep" for the helicopter



to land on. We then incrementally merge the remaining triangles in  $U$  into subsets of triangles  $\mathbb{U} = \{U_1, \dots, U_k\}$ , with  $U_i \subset U$  where for each triangle  $u_1 \in U_i$ , there is at least one  $u_2 \in U_i$ ,  $u_1 \neq u_2$  such that  $u_1$  and  $u_2$  share an edge in the original terrain mesh. Each  $U_i \in \mathbb{U}$  is then a possible landing zone. However, some triangle sets in  $\mathbb{U}$  might have an area (defined as the sum of the area of all triangles in the triangle set) that is too small for the helicopter to land on. To determine whether the triangles in a triangle set  $U_i$  provide a sufficient area for landing, we project the triangles in  $U_i$  onto the  $xy$ -plane and compute the area of the largest inscribed circle within the (possibly non-convex) boundary polygon of the projected triangles. If this area is smaller than a given threshold (in our experiments we use  $75\pi m^2$ ), we remove  $U_i$  from  $\mathbb{U}$ . The remaining triangle sets in  $\mathbb{U}$  are then the resulting landing zones.

Note that this process of constructing the landing zones is done off-line. During run-time, once the trajectory planner switches to the landing problem, we select the closest landing zone in  $\mathbb{U}$  to the injured person (in terms of the distance of the location of the injured person to the geometric centers of the landing zones) which then becomes the target landing zone for the pilot to land on.

## Parallelization of Belief Update and Policy Computation

Most on-line POMDP solvers (including ABT) typically follow a strictly sequential order of execution, that is, policy computation – policy execution – belief update. In practice, such an implementation would incur significant delays between the execution of two actions, due to the time required to update the current belief and compute a policy for the updated belief. To reduce these delays to a minimum, we parallelize policy computation, policy execution and parts of the belief update, similar to the method proposed in (Hoerger et al. 2019): While the helicopter executes an action  $a \in \mathcal{A}$ , we run two processes in parallel.

The first process is the belief-update process which draws samples from a proposal distribution, (in our case  $T(s, a, s')$ ) using state samples drawn from the current belief and the currently executed action. Once the helicopter receives an observation, all that remains for the belief update is to update the importance weights, up to a normalization constant, based on the perceived observation, which can be done fast.

The second process is the policy-update process. Once the helicopter starts executing  $a$  from the current belief  $b$ , our implementation of ABT plans for the next step by sampling additional episodes starting from the current belief, using the currently executed action as the first action of the sampled episodes, thereby improving the policy within the entire descendant of  $b$  via  $a$  in the belief tree. This strategy increases the chances that after the helicopter has executed  $a$  and the belief is updated based on the observation perceived, a good policy for the next belief is readily available.

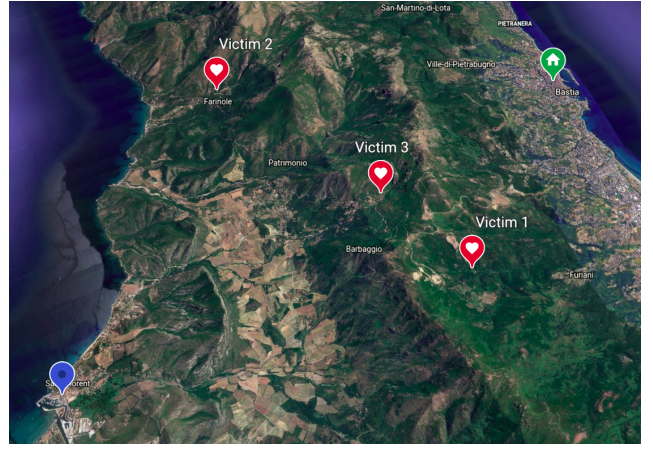


Figure 2: The mission area used throughout the experiments. Initially the helicopter starts at Saint-Florent (shown as a blue marker) with a victim on board and the task is to navigate to the hospital in Bastia (shown as a green marker). The red markers indicate the locations of the additional victims the helicopter has to pick-up before continuing its mission to Bastia. Image: Google Earth, [earth.google.com/web/](http://earth.google.com/web/)

## Experiments and Results

### Problem Scenarios

To evaluate our system, we tested it on three problem scenarios in which a pilot operates on a map of size  $(28.1 \times 26.0)$ km in the Corsica region in France near Saint-Florent and Bastia, shown in Figure 2. In all three scenarios the pilot starts near Saint-Florent at location  $(42.68^\circ N, 9.302^\circ E)$  (shown as a blue marker in Figure 2) with an injured person on board of the helicopter and its task is to safely navigate to a hospital in Bastia at location  $(42.740105^\circ N, 9.457107^\circ E)$  (shown as a green marker in Figure 2) to deliver the injured person. Furthermore, at the beginning of the mission, the helicopter is instructed to pick up an additional injured person in the environment before continuing its flight to Bastia. For the dynamic model of the helicopter defined in eq.(2) we assume that the constant acceleration  $\alpha$ , climb rate  $\delta_h$  and turn rate  $\delta_\theta$  is  $\alpha = \pm 4m/s^2$ ,  $\delta_h = \pm 6m/s$  and  $\delta_\theta = \pm 4deg/s$  respectively. We further assume that the pilot applies each action for a control duration of  $\Delta = 5s$ .

For **Scenario 1**, the additional victim is located at  $(42.667617^\circ N, 9.392734^\circ E)$  (shown as Victim #1 in Figure 2) and the pilot has to find a motion strategy to navigate to Victim #1, safely land near the additional victim to board it, and finally safely deliver both victims to the hospital in Bastia.

In **Scenario 2**, the pilot must pick-up Victim #1, too. However, at time  $t = 15$ , the pilot is informed about a no-fly zone near the current location of the helicopter. This no-fly zone causes the initial mission plan to become infeasible. Subsequently, the pilot must find an alternative strategy to reach Victim #1 before delivering both victims to the hospital.

The no-fly zone is simulated by modifying the mission

planner graph. Specifically, by removing the edge between vertices  $v_1 \in \mathcal{V}$  and  $v_2 \in \mathcal{V}$ , where  $v_1$  is the vertex associated to the Voronoi region the helicopter is located in at time  $t = 15$  and  $v_2$  is the vertex associated to the target Voronoi region of the current mission planner action  $\text{move}(v_1, v_2)$ .

For **Scenario 3** the pilot is initially informed that there's a victim at location  $(42.732068^\circ N, 9.363066^\circ E)$  (shown as Victim #2 in Figure 2). At time  $t = 20$  the mission planner receives an emergency call from Mission Control, informing it about a second victim at location  $(42.688724^\circ N, 9.384275^\circ E)$  (shown as Victim #3 in Figure 2). However, since there's already a victim on board at the start of the mission, the helicopter has space for only one additional victim on board due to space and weight limits. Subsequently, the pilot must decide which of the two victims in the environment to pick up before continuing the mission to the hospital in Bastia.

## Experimental Setup

For all three problem scenarios we first generated 10 graphs for the mission planner using the SPARSe implementation provided by OMPL. Each graph was constructed by running SPARSe for 10 minutes. We then compute, for each scenario and each graph a mission planner policy that serves as the initial mission plan for the trajectory planner.

For the mission planner we use Labelled SSiPP (Trevizan and Veloso 2012) with the Regrouped Operator Counting heuristic (Trevizan, Thiébaux, and Haslum 2017). The initial mission plan is computed off-line using one thread on an Intel Xeon Silver 4110 CPU with 2.1Ghz and 128GB of memory, and took approximately 13.6s (10s to construct the graph and 3.6s to compute the mission planner policy) on average per scenario per graph.

For the trajectory planner, all POMDP models as well as the parallelized version of ABT (as discussed in Sub-Section **Parallelization of Belief Update and Policy Computation**) were implemented in C++ within the OPPT-framework (Hoerger, Kurniawati, and Elfes 2018). For the POMDP models we used a discount factor of  $\gamma = 0.98$ . The size of the local map the trajectory planner maintained during run-time was set to be  $4,000 \times 4,000\text{m}$ . All simulations were run using 3 threads on an Intel Xeon Silver 4110 CPU with 2.1Ghz and 128GB of memory. We assume that each action the pilot executes takes 5s to complete, and provides a 5s planning time per step to the trajectory planner.

## Results

For each problem scenario and each initial mission planner policy, we tested our system using 100 simulation runs. Table 1 shows the success rate, average number of steps and average total discounted reward achieved by SM2P in all three problem scenarios. In all three scenarios, SM2P achieved a success rate of at least 96% where the pilot successfully picked up a victim in the environment and reached the hospital, demonstrating the robustness of SM2P in challenging HEMS missions.

Table 2-5 provides detailed results for all Scenarios and mission planner graphs. These results indicate the robustness of SM2P. Recall that since the mission planner graphs

are computed using a sampling-based method, these graphs are different between one run and another. Despite such differences, SM2P consistently achieve a success rate of over 94%.

Looking at the average number of steps for Scenario 1 and 2 in Table 1, we can see that it typically takes slightly longer (around 20 steps longer) to complete the mission in Scenario 2 compared to Scenario 1. This is not surprising. Due to the no-fly zone introduced at  $t = 15$  in Scenario 2, the helicopter must take a slight detour to reach the victim. Furthermore, since the no-fly zone causes the initial mission plan to become invalid, it is paramount that the mission planner is able to quickly update its policy. On average, SM2P can update the mission plan during run-time in 5 seconds. This is roughly the same time it takes for the pilot to execute an action, which indicates that updating the mission planner policy is efficient enough for an on-line planning setting, even if there are structural changes to the mission planner graph (such as an edge being deleted).

Table 5 shows the number of times (in percent) per graph, SM2P decided to pick-up Victim #3 in Scenario 3. The decision of which victim to pick-up is affected by the Voronoi region the helicopter is estimated to be located in at time  $t = 20$ , when the mission planner is informed regarding the location of the additional Victim #3, causing the mission plan to be updated. For the majority of the runs and mission planner graphs, SM2P decided to pick up Victim #3, since the location of Victim #3 is closer to the hospital. A notable exception is graph 9 for which SM2P decided to pick up Victim #2 in 30% of the runs. The reason is that for graph 9, the helicopter often operates near the border of two neighbour Voronoi regions at time  $t = 20$ . Depending on which Voronoi region the helicopter is estimated to be located in, the mission planner policy suggests to navigate either to Victim #2 or Victim #3. Recall from Sub-Section **Trajectory Planner Details** that the current Voronoi region is estimated from the mean state of the current belief. The consistency of selecting which victim to pick up can be improved, for instance, by estimating the Voronoi region that contains the largest amount of probability mass of the current belief.

	Success Rate	Avg. num steps	Avg. total discounted reward
Scenario 1	96.8%	$197.4 \pm 2.1$	$-385.8 \pm 56.2$
Scenario 2	96.9%	$217.1 \pm 3.7$	$-468.2 \pm 52.3$
Scenario 3	97.6%	$206.8 \pm 2.3$	$-416.6 \pm 55.6$

Table 1: The success rate (in percent), Average number of steps and Average total discounted reward achieved by SM2P in all three problem scenarios. The average is taken over all 10 mission planner graphs using 100 simulation runs per scenario and graph.  $\pm$  indicates the 95% confidence intervals.

## Summary

This paper presents our preliminary work in developing a robust decision-making component to help a pilot performing

Graph	Success Rate	Avg. num steps	Avg. total discounted reward
1	94%	191.02 $\pm$ 8.8	-618.67 $\pm$ 183.54
2	95%	221.81 $\pm$ 6.3	-583.94 $\pm$ 230.53
3	100%	218.5 $\pm$ 3.8	-146.58 $\pm$ 109.75
4	97%	194.09 $\pm$ 4.4	-361.88 $\pm$ 312.16
5	99%	177.76 $\pm$ 4.4	-128.89 $\pm$ 190.0
6	98%	192.19 $\pm$ 9.8	-225.26 $\pm$ 377.9
7	94%	181.60 $\pm$ 6.0	-589.94 $\pm$ 228.06
8	95%	224.0 $\pm$ 5.5	-591.90 $\pm$ 155.67
9	100%	176.02 $\pm$ 6.3	-93.37 $\pm$ 90.39
10	96%	201.32 $\pm$ 6.9	-308.08 $\pm$ 196.99

Table 2: Results for **Scenario 1**. The success rate, average number of steps and average total discounted rewards are computed over 100 simulation runs per mission planner graph.  $\pm$  indicates the 95% confidence intervals.

Graph	Success Rate	Avg. num steps	Avg. total discounted reward
1	100%	207.83 $\pm$ 6.8	-46.49 $\pm$ 93.99
2	94%	225.72 $\pm$ 8.47	-618.64 $\pm$ 748.15
3	95%	220.52 $\pm$ 10.5	-525.93 $\pm$ 125.34
4	97%	218.61 $\pm$ 9.5	-638.31 $\pm$ 102.78
5	98%	206.82 $\pm$ 4.5	-167.16 $\pm$ 108.46
6	95%	192.55 $\pm$ 9.7	-618.4 $\pm$ 265.65
7	98%	195.9 $\pm$ 6.8	-452.21 $\pm$ 197.56
8	95%	218.36 $\pm$ 6.61	-663.73 $\pm$ 303.60
9	98%	211.09 $\pm$ 8.22	-530.31 $\pm$ 683.77
10	99%	205.77 $\pm$ 4.6	-167.32 $\pm$ 107.25

Table 3: Results for **Scenario 2**. The success rate, average number of steps and average total discounted rewards are computed over 100 simulation runs per mission planner graph.  $\pm$  indicates the 95% confidence intervals.

a HEMS mission —time-critical missions that are typically plagued by various types of uncertainty. In this work, we propose Stochastic-based Mission-Motion Planner (SM2P) as a framework that enable us to quantify uncertainty and account such a quantification in its decision-making. SM2P uses a combination of existing SSP and POMDP solvers that are tightly coupled by their ability to revise and recompute plans on-line within a few seconds. Simulation results indicate that SM2P is sufficiently efficient to compute good strategies for a co-pilot system in HEMS missions.

Future work abounds. For instance, in this work, we use a simplified helicopter model. Using high fidelity helicopter model will slow down the POMDP solver. Therefore, incorporating solvers that can perform well for problems with complex dynamics, such as (Hoerger et al. 2019) would be useful. Furthermore, uncertainty due to pilot stress level and experience have not been taken into account in this work. However, a reliable co-pilot system would account for such factors. Another dimension is handling potential degradation of the pilot’s visibility. Although sensors that can “see

Graph	Success Rate	Avg. num steps	Avg. total discounted reward
1	100%	210.9 $\pm$ 7.4	-118.32 $\pm$ 116.30
2	97%	204.83 $\pm$ 6.1	-928.58 $\pm$ 458.79
3	96%	213.56 $\pm$ 8.2	-397.28 $\pm$ 396.25
4	100%	216.28 $\pm$ 7.4	-189.51 $\pm$ 173.52
5	98%	196.11 $\pm$ 7.4	-253.78 $\pm$ 386.41
6	96%	191.79 $\pm$ 8.2	-587.23 $\pm$ 195.28
7	98%	213.56 $\pm$ 8.9	-321.42 $\pm$ 164.33
8	96%	227.56 $\pm$ 6.8	-634.76 $\pm$ 587.32
9	100%	215.81 $\pm$ 5.4	-104.11 $\pm$ 176.50
10	95%	201.73 $\pm$ 5.4	-781.69 $\pm$ 426.47

Table 4: Results for **Scenario 3**. The success rate, average number of steps and average total discounted rewards are computed over 100 simulation runs per mission planner graph.  $\pm$  indicates the 95% confidence intervals.

Graph	Victim #3 pickup (in %)
1	93%
2	90%
3	100%
4	100%
5	87%
6	100%
7	100%
8	94%
9	70%
10	100%

Table 5: The number of times (in percent) SM2P decided to pick-up Victim #3 in **Scenario 3** for each mission planner graph over all simulation runs.

through” fogs have been developed, uncertainty due to degraded visibility conditions when operating in certain conditions (e.g., bushfires) and its effects to the pilot’s capability will need to be considered in the recommendations provided by a co-pilot system. Last but not least, experiments on high fidelity simulator or physical system is needed for better validation.

## Acknowledgments

We thank Colonel Andjy Zouag for the many discussions in developing realistic EMS scenarios used in the experiments for this paper.

This project is sponsored by Safran Electronics and Defense and Safran Electronics and Defense Australasia.

## References

- October 2013. National Search Rescue (NSARA). Helicopter Rescue Techniques. [https://www.dco.uscg.mil/Portals/9/CG-5R/nsarc/Helicopter\\_Rescue\\_Techniques\\_NSARA\\_Manual\\_10-23-2013.pdf](https://www.dco.uscg.mil/Portals/9/CG-5R/nsarc/Helicopter_Rescue_Techniques_NSARA_Manual_10-23-2013.pdf).
- Arulampalam, M. S.; Maskell, S.; Gordon, N.; and Clapp, T. 2002. A tutorial on particle filters for online nonlinear/non-

Gaussian Bayesian tracking. *IEEE Transactions on signal processing* 50(2): 174–188.

Auer, P.; Cesa-Bianchi, N.; and Fischer, P. 2002. Finite-time Analysis of the Multiarmed Bandit Problem. *Machine Learning* 47(2-3): 235–256. ISSN 0885-6125.

Bertsekas, D.; and Tsitsiklis, J. 1991. An Analysis of Stochastic Shortest Path Problems. *Mathematics of Operations Research* 16(3): 580–595. ISSN 0364-765X.

Choudhury, S.; Dugar, V.; Maeta, S.; MacAllister, B.; Arora, S.; Althoff, D.; and Scherer, S. 2019. High performance and safe flight of full-scale helicopters from takeoff to landing with an ensemble of planners. *Journal of Field Robotics* .

Dobson, A.; Krontiris, A.; and Bekris, K. E. 2013. Sparse roadmap spanners. In *Algorithmic Foundations of Robotics X*, 279–296. Springer.

Hart, C. A. March 2017. National Transportation Safety Board (NTSB). [https://www.nts.gov/news/speeches/CHart/Documents/hart\\_20170302.pdf](https://www.nts.gov/news/speeches/CHart/Documents/hart_20170302.pdf).

Hoerger, M.; Kurniawati, H.; and Elfes, A. 2018. A Software Framework for Planning Under Partial Observability. In *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 1–9. IEEE.

Hoerger, M.; Song, J.; Kurniawati, H.; and Elfes, A. 2019. POMDP-based Candy Server: Lessons Learned from a Seven Day Demo. In *Proc. Int. Conference on Automated Planning and Scheduling (ICAPS)*.

Holland, J.; and Cooksley, D. G. 2005. Safety of helicopter aeromedical transport in Australia: a retrospective study. *Medical journal of Australia* 182(1): 17–19.

Kurniawati, H.; and Yadav, V. 2013. An Online POMDP Solver for Uncertainty Planning in Dynamic Environment. In *Proc. Int. Symp. on Robotics Research*.

Lantzech, R.; Greiser, S.; Wolfram, J.; Wartmann, J.; Müllhäuser, M.; Lüken, T.; Döhler, H.-U.; and Peinecke, N. 2012. ALLFLIGHT: Helicopter pilot assistance in all phases of flight. .

Le Blaye, P. 2003. A concept of Flight Execution Monitor (FEM) for helicopter pilot assistance. Technical report, OFFICE NATIONAL D’ETUDES ET DE RECHERCHES AERONOSPATIALES CEDEX FRANCE.

Lüken, T.; and Korn, B. 2007. PAVE: A prototype of a helicopter pilot assistant system. Technical report.

Mausam; and Kolobov, A. 2012. *Planning with Markov Decision Processes*. Morgan&Claypool.

Trevizan, F.; Teichteil-Königsbuch, F.; and Thiébaux, S. 2017. Efficient Solutions for Stochastic Shortest Path Problems with Dead Ends. In *Proc. of 33rd Int. Conf. on Uncertainty in Artificial Intelligence (UAI)*.

Trevizan, F.; Thiébaux, S.; and Haslum, P. 2017. Occupation Measure Heuristics for Probabilistic Planning. In *Proc. of 27th Int. Conf. on Automated Planning and Scheduling (ICAPS)*.

Trevizan, F.; and Veloso, M. 2012. Trajectory-Based Short-Sighted Probabilistic Planning. In *Advances in Neural Information Processing Systems (NIPS)*.

# State-Temporal Decoupling of Multi-Agent Plans under Limited Communication

Yuening Zhang, Jingkai Chen, Eric Timmons, Marlyse Reeves, Brian Williams

Computer Science and Artificial Intelligence Laboratory  
Massachusetts Institute of Technology

zhangyn@mit.edu, jkchen@csail.mit.edu, etimmons@mit.edu, mreeves@mit.edu, williams@mit.edu

## Abstract

When a team of agents execute a mission in a distributed fashion, they often communicate with each other to synchronize their progress. However, in situations where communication may be delayed, unavailable or costly, such as when a suite of underwater vehicles is scouting an underwater area in the ocean, pre-coordination is needed beforehand to compensate for limited communication. Previous work proposed decoupling algorithms for Multi-Agent Simple Temporal Network with Uncertainty (MaSTNU) in order to find decoupled execution strategies for the agents, including communication strategy, that satisfy all the inter-agent temporal constraints. However, there is often the coupling between temporal and state constraints, such as the constraint that the vehicles may only communicate with each other when they are within a certain distance. In this paper, we propose using Multi-Agent Qualitative State Plan (MaQSP) that extends MaSTNU to including continuous state constraints in order to model multi-agent plans with coupled state and temporal constraints. We describe a decoupling algorithm for MaQSP using a mixed-integer linear programming (MILP) encoding, which includes a novel path planning algorithm under temporal uncertainty.

## Introduction

When multiple agents execute a shared task, the agents often depend on each other, resulting in inter-agent precedence or synchronization constraints. To satisfy those constraints, the agents communicate with each other to synchronize their tasks and update their progress. However, in many cases, the team may operate under limited communication, where communication is not always available and may be delayed or costly. For example, when deploying of a fleet of autonomous underwater vehicles (AUVs) to scout an underwater area in the ocean, communication is mostly unavailable as the AUVs are operating underwater, and has to be planned ahead if communication is required.

Previous work has proposed modeling the multi-agent execution problem as a Multi-Agent Simple Temporal Network (MaSTN) (Hunsberger 2002; Boerkoel Jr and Durfee 2013) or a Multi-Agent Simple Temporal Network with Uncertainty (MaSTNU) (Casanova et al. 2016). They handle limited communication between the team by finding decoupled execution strategies for the agents in the form of

a set of local executable networks that depend only on observable information, which is called a temporal decoupling. More specifically, the decoupling for a MaSTN completely removes the need for communication, whereas MaSTNU explicitly models the available communication and its decoupling involves planning for how communication is used to support the mission, resulting in a more flexible coordination strategy. Additionally, MaSTNU also allows the modeling of uncertain durations of activities. While MaSTNU addresses the important problem of temporal coordination under limited communication, it fails to handle the case when there is coupling between temporal constraints and state constraints. In real-life deployments, the AUVs can only communicate when they surface. While AUVs can communicate with each other within a short distance, the ship often has a larger communication range, and can communicate with both of them over a much larger distance. Additionally, there may also exist inter-agent state constraints such as when an AUV finishes its scouting mission, another AUV may want to pick up the scouting mission from where the first AUV left off. Previous work that considers multi-agent coordination with both temporal constraints and continuous state constraints only considers the problem under full observability, modeled as a Qualitative State Plan (QSP) (Léauté and Williams 2005), and its solution is an execution strategy that assumes the existence of a centralized authority that controls all the agents (Fernández-González, Williams, and Karpas 2018; Reeves, Fernández-González, and Williams 2019).

In this paper, we draw insights from the above work, and propose using Multi-Agent Qualitative State Plan (MaQSP) to model both the state and temporal constraints for multi-agent plans under limited communication. With the addition of continuous state constraints, MaQSP allows us to represent the common problem of combined temporal coordination and task planning. We describe a decoupling algorithm for MaQSP by encoding the problem into a mixed-integer linear program (MILP), which depends on a novel path planning algorithm for QSPs under temporal uncertainty assuming first-order dynamics of the agents. Finally, we provide preliminary experiment results on the algorithm.

## Motivating Example

Consider a pedagogical example where two AUVs are deployed from the ship on a scouting mission. The vehicle's

position is a vector  $\langle x, y, d \rangle$ , where  $d$  is the depth and  $d \geq 0$ . The initial position is  $\langle 0, 0, 5 \rangle$  for both AUVs, and  $\langle 0, 0, 0 \rangle$  for the ship. The ship is always above the surface of the water, i.e.  $d = 0$  throughout the mission. AUV1 and AUV2 need to take a sample at science site  $L1$  and  $L2$ , respectively.  $L1$  region is a rectangular cuboid enclosed by its two corners  $[\langle 65, 65, 6 \rangle, \langle 70, 70, 20 \rangle]$  and  $L2$  by  $[\langle -45, 35, 6 \rangle, \langle -40, 40, 20 \rangle]$ . Each of these sampling missions may take any time between 10 to 30 minutes. It is also required that AUV2 must start its sampling mission within 15 minutes after AUV1 has finished its mission.

Notice that since the AUVs are operating underwater and relatively distant from each other, they cannot observe each other's progress. As a result, AUV2 cannot observe when AUV1 has finished its mission, which makes it difficult to satisfy the inter-agent temporal constraint. However, it is possible for the vehicles to communicate their progress by notifying each other upon the occurrence of certain events, though communication is costly and subject to certain constraints. The AUVs can communicate only when they surface. The AUVs can communicate with each other if they are within 30 meters of each other. The AUV and the ship can communicate if they are within 100 meters of each other.

Our problem is to find a coordination strategy so that the AUVs can successfully execute the mission. In this case, because the sampling mission may take any time between 10 to 30 minutes, which cannot be determined beforehand, it becomes necessary for AUV1 to notify AUV2 upon finishing its mission so that AUV2 can react in time to start its mission within 15 minutes. Because the AUVs will be far away from each other in their respective sampling missions, the only possible way to communicate would be to relay the communication from the ship. Additionally, since the AUVs can only communicate when they surface, they must plan for enough time to surface before communication.

## Problem Definition

### Multi-Agent Qualitative State Plan

We represent the above multi-agent plan by a Multi-Agent Qualitative State Plan (MaQSP), which is an extension of MaSTNU (Casanova et al. 2016) to capture both temporal and state constraints. MaQSP introduces episodes in place of the original temporal constraints, which is a concept from Qualitative State Plan (QSP) (Léauté and Williams 2005). We may also consider MaQSP as an extension of QSP to the multi-agent context. Compared to the typical QSPs, our definition allows the modeling of temporal uncertainty.

**Definition 1 (QSP).** A *Qualitative State Plan* (QSP) is a tuple  $\langle V, X, EP \rangle$ , where

- $V$  is a set of events representing designated time points.
- $X$  is a set of continuous state variables.
- $EP$  is a set of *episodes*, where each episode  $ep \in EP$  is a tuple  $\langle s, t, e, SC, DC \rangle$ , in which
  - $s, t \in V$  is the start and end event of the episode.
  - $e$  is a temporal constraint  $\langle s, t, lb, ub, ctg \rangle$ , where  $lb \in \mathbb{R} \cup \{-\infty\}$ ,  $ub \in \mathbb{R} \cup \{+\infty\}$  is the lower bound and upper bound from  $s$  to  $t$ , i.e.  $lb \leq t - s \leq ub$ , and  $ctg$

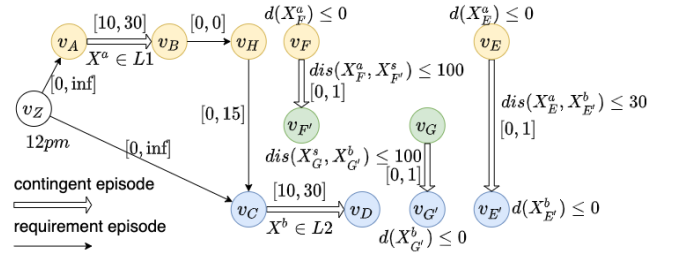


Figure 1: Example MaQSP. We omit the  $[0, \infty]$  requirement constraint from  $v_Z$  to every other event to avoid cluttering.

is a Boolean indicating if  $e$  is a contingent constraint, in which case,  $0 \leq lb < ub < \infty$ .

- $SC$  is a set of state constraints scoped on  $X$ .
- $DC$  is a set of delta constraints, where  $dc(X_s, X_t) \in DC$  specifies a constraint between the state variables evaluated at the start event and at the end event.

When the  $ctg$  flag is set to false, the temporal constraint is a *simple temporal constraint* (Dechter, Meiri, and Pearl 1991), also referred to as a *requirement constraint*, that requires the scheduling of two events to be within certain lower bound and upper bound. When set to true, it is a *simple temporal contingent constraint* (Vidal 1999), or a *contingent constraint* for short, whose end event is an *uncontrollable event* that cannot be directly controlled by the agent, but can be observed when it occurs. A contingent constraint specifies the bound in which the uncontrollable event may occur, and there is a unique contingent constraint for each uncontrollable event. In our example, we can use contingent constraints to express that fact that it may take anytime between 10 to 30 minutes to take a sample, but the duration cannot be determined beforehand. When an episode does not include any state constraints, it is effectively a temporal constraint.

Under the multi-agent context, MaQSP can be considered as a partition of QSP into a set of agents, resulting in a set of agents' local QSPs and additional inter-agent episodes.

**Definition 2 (MaQSP).** A *Multi-Agent Qualitative State Plan* (MaQSP) is a tuple  $\langle N^A, E_X, C_X, v_Z \rangle$ , where

- Each  $N^a \in N^A$  is the *local state plan* for agent  $a \in \mathcal{A}$ , which is a QSP  $\langle V^a, X^a, EP^a \rangle$ , where  $V^a$ ,  $X^a$ , and  $EP^a$  are the local events, local state variables, and local episodes for agent  $a$ , respectively.
- $E_X \cup C_X$  is a set of external episodes, whose temporal constraint connects the local events of two different agents. *External requirement episodes*  $E_X$  and *external contingent episodes*  $C_X$  include a requirement and contingent temporal constraint, respectively.
- $v_Z$  is a reference event, an absolute time point proceeding all other events and shared by all agents, such as 12 pm.

Our motivating example can be formulated as a MaQSP shown in Figure 1, where each event is represented by a circle and an episode is represented by an arrow pointing from the start event to the end event, with contingent episodes represented by double arrows. The reference event  $v_Z$  is the initial time point 12 pm. Each vehicle's local state plan is high-



lighted in color, with AUV1 in yellow, AUV2 in blue and the ship in green. For example, AUV1's local state plan consists of local events  $\{v_A, v_B, v_E, v_F, v_H\}$  and episode  $ep_{AB}$  represents its sampling mission at science site  $L1$ . The external requirement episode  $ep_{HC}$  represents the inter-agent temporal constraint that AUV2 has to start its sampling mission within 15 minutes after AUV1 finishes its mission. The external contingent episodes  $C_X = \{ep_{EE'}, ep_{FF'}, ep_{GG'}\}$  are also referred to as the *communication links* that represent available communication between the agents. In this case, we allow AUV1 to communicate to AUV2 once with  $ep_{EE'}$ , and similarly from AUV1 to the ship  $ep_{FF'}$ , and the ship to AUV2  $ep_{GG'}$ . The timing of communication is unconstrained. For example,  $ep_{EE'}$  means that AUV2 can observe the occurrence of event  $v_{E'}$  within a delay of 1 min after event  $v_E$  is scheduled by AUV1, if they are within 30 meters of each other, AUV1's depth is 0 at event  $v_E$ , and AUV2's depth is 0 at event  $v_{E'}$ .

We characterize three types of state constraints allowed in an episode  $ep = \langle v_i, v_j, e, SC, DC \rangle$ . Notation-wise, we use  $X_i$  to denote the state variables  $X$  evaluated at event  $v_i$ , and  $a(v_i)$  denotes the agent that event  $v_i$  belongs to.

- For  $sc \in SC_{start}$ ,  $sc$  is satisfied at the start event  $v_i$  of the episode. That is,  $sc(X_i)$  holds, where  $X_i \subseteq X^{a(v_i)}$ .
- For  $sc \in SC_{end}$ ,  $sc$  is satisfied at the end event  $v_j$  of the episode. That is,  $sc(X_j)$  holds, where  $X_j \subseteq X^{a(v_j)}$ .
- For  $sc \in SC_{overall}$ ,  $sc$  is satisfied throughout the episode. That is,  $\forall T$  s.t.  $v_i \leq T \leq v_j$ ,  $sc(X_T)$  holds, where  $X_T \subseteq X^{a(v_i)} \cup X^{a(v_j)}$ .

In this paper, we consider the following forms of continuous state constraints, where  $A$  is a constant matrix,  $B$  is a constant vector, and  $c$  is a constant:

- $AX \in L$ , where  $L$  is a convex region approximated by a set of linear inequalities.
- $AX \leq B$ , which is a set of linear inequalities.
- $dis(A_1X, A_2X) \leq c$ , where  $A_1$  and  $A_2$  has the same size, and  $A_1X, A_2X$  usually corresponds to the state variables belonging to different agents.

While the state constraints may take many forms, the above are among the ones typically encountered that also guarantees convexity of our problem. Additionally, we will use  $dis(A_1X_s, A_2X_t) \leq c$  to denote distance constraint when it is a delta constraint to differentiate it from an overall state constraint. Note that for distance constraints, since we use a MILP encoding in this paper, we can use L1 distance instead of L2 distance, but we can easily extend it to L2 distance by using MIQCP.

For example, in Figure 1,  $X^a \in L1$  for episode  $ep_{AB}$  is an overall state constraint that requires AUV1 to stay within science site  $L1$  throughout the episode.  $d(X_E^a) \leq 0$  for communication link  $ep_{EE'}$  is a state constraint to be satisfied at the start event. In our example,  $dis(X_E^a, X_{E'}^b) \leq 30$  for  $ep_{EE'}$  is a delta constraint that requires the location where AUV1 initiates the communication and the location where AUV2 receives the communication need to be within 30 meters from each other. In this case, we assume the delay

is caused by the transmission over media, but the initiation and reception of message is instantaneous. In other cases, it may be reasonable to model a communication link with an overall state constraint  $dis(X^a, X^b) \leq 30$  that requires the two vehicles to be within 30 meters of each other throughout the entire communication process, for example, to transmit data. State constraints and delta constraints apply to external requirement episodes too. For example, we may require as a delta constraint that when AUV1 finishes its scouting mission, AUV2 should continue scouting from where AUV1 left off to maintain the consistency of data collected. An example of  $SC_{overall}$  may be a tethering constraint, such as when a remotely operated vehicle (ROV) is deployed underwater but is tethered to the ship, it has to stay within a certain distance to the ship throughout the entire mission.

### State Temporal Decoupling Problem

Our state temporal decoupling problem for MaQSP is a natural extension of the temporal decoupling problem for MaSTNU (Zhang and Williams 2021).

**Definition 3** (State Temporal Decoupling). Given a MaQSP, the set of agents' local state plans  $N^A$  forms a *state temporal decoupling* of the MaQSP if:

- (*feasibility*) All local state plans  $N^A = \{N^{a_1}, N^{a_2}, \dots, N^{a_n}\}$  are feasible. That is, there exists a dynamic and valid execution strategy for each local state plan.
- (*validity*) Merging *any* combination of execution strategies for the local state plans  $N^A$  yields a solution to the MaQSP, that is, given that the external contingent episodes  $C_X$  are satisfied, all the external requirement episodes  $E_X$  are also satisfied.

The execution strategy for a local state plan is dynamic as it may need to react on the fly to real-time observations of when the uncontrollable events occur. The execution strategy is valid if the resulting execution satisfies all the temporal and state constraints in the state plan. In this paper, we assume that the evolution of each continuous state variable follows a first-order dynamical model  $\dot{x} = v$ , where  $v \leq v_{max}$  with  $v_{max}$  being a fixed maximum change rate.

**Definition 4** (Decoupling Problem). The state temporal decoupling problem for MaQSP is a tuple  $\langle M, X_0 \rangle$ , where  $M$  is a MaQSP, and  $X_0 = \cup_{a \in A} X_0^a$  specifies the initial state of the agents at the reference event  $v_Z$ . The goal is to find a set of *decoupling episodes* for each agent  $EP_d^a$ , such that the set of augmented local state plans  $N_{+\Delta}^a = \langle V^a, X^a, EP^a \cup EP_d^a \rangle$  for each agent  $a$  forms a state temporal decoupling of the MaQSP.

The feasibility condition in Definition 3 requires that the addition of decoupling episodes does not over-constrain any local state plan and makes it infeasible. The validity condition requires that if the local execution strategies satisfy the decoupling episodes, then the external requirement episodes must also be satisfied.

Figure 2 shows an example decoupling for our motivating example, where communication from AUV1 to AUV2 is relayed through the ship and used to support the satisfaction of  $ep_{HC}$ . The highlighted red arrows represent the decoupling episodes. For example,  $ep_{ZF}$  requires that AUV1

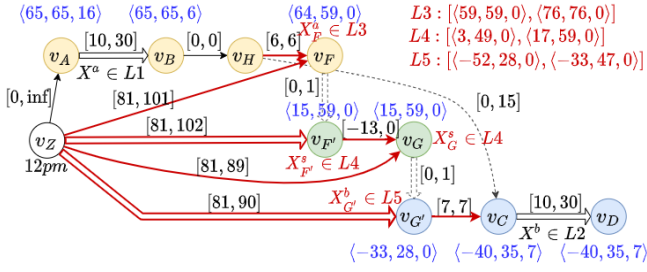


Figure 2: Decoupling solution for motivating example

must execute event  $v_F$  between 81 to 101 minutes after  $v_Z$ , and it must be in region  $L3$  at  $v_F$ , where  $L3$  is a rectangular cuboid enclosed by its two corners  $\langle 59, 59, 0 \rangle, \langle 76, 76, 0 \rangle$ . The execution strategy for the ship is that it should schedule  $v_G$  as soon as it receives  $v_{F'}$  until 89 minutes after  $v_Z$ , at which point even if  $v_{F'}$  is not received, it should schedule  $v_G$ . A feasible trajectory for each vehicle is shown by its  $\langle x, y, d \rangle$  position at each event highlighted in blue. Notice that a decoupling solution retains the flexibility for each agent to execute its own state plan, and it only has to enforce the necessary constraints to ensure validity.

### Decoupling Algorithm

Our state temporal decoupling algorithm for MaQSP builds on top of the temporal decoupling algorithm for MaSTNU (Casanova et al. 2016) to handle additional continuous state constraints, and follows their use of mixed-integer linear programming (MILP) in order to solve the problem. More specifically, such an encoding involves encoding both the validity condition and the feasibility condition in a single MILP problem, and solving it using off-the-shelf optimization solvers. Intuitively, the validity condition specifies a set of MILP constraints that ensures that the external episodes across agents are decoupled and can be safely removed from the MaQSP without affecting the correctness of the execution result. The feasibility condition specifies a set of MILP constraints that ensures that the local state plans are feasible and can be successfully executed.

Our decoupling algorithm extends the original temporal decoupling algorithm in the following ways: For the validity condition, we use Casanova’s encoding to decouple all the external temporal constraints, and add on top of it encoding to decouple all the external state and delta constraints. For the feasibility condition, in the case of MaSTNU, since it only concerns the scheduling problem, its local plan is an instance of simple temporal network with uncertainty (STNU) (Vidal 1999) without any state constraints. Therefore, the feasibility of a STNU is simply its dynamic controllability, which can be encoded as a MILP (Cui and Haslum 2017; Wah and Xin 2007). With additional continuous state constraints, the feasibility of our local state plan becomes a path planning problem under temporal uncertainty. Therefore, we extend the MILP encoding to solve for feasibility of QSP with temporal uncertainty, which is the first to address path planning under temporal uncertainty as we know of.

In the following, we start by describing the temporal va-

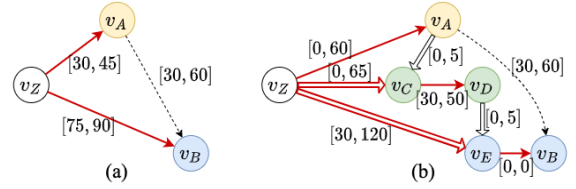


Figure 3: (a) Temporal decoupling without communication (b) Temporal decoupling with communication relay

lidity encoding (Casanova et al. 2016). We then describe the encoding for feasibility of QSP, and the state validity encoding. The final decoupling algorithm puts everything together, which consists of the temporal validity encoding, the state validity encoding, and feasibility constraints for each agent’s local QSP.

### Temporal Validity Encoding

We start by showing two temporal decoupling examples in Figure 3 to show the intuitive ideas behind the temporal decoupling algorithm. In these examples, the goal is to satisfy the external requirement temporal constraint  $e_{AB}$ .

Recall that validity condition requires that by having each agent execute its own local plan, the external requirement constraints are guaranteed to be satisfied. In Figure 3(a),  $e_{AB}$  is satisfied by imposing two local temporal constraints,  $e_{ZA}$  for AUV1 and  $e_{ZB}$  for AUV2. Note that since event  $v_Z$  is a reference time point shared by all agents, any constraint connected to it is considered a local constraint. Assuming these local constraints are satisfied, we have  $v_B - v_A = (v_B - v_Z) + (v_Z - v_A) = [75, 90] + [-45, -30] = [30, 60]$ , which satisfies  $e_{AB}$ . Intuitively, we have constrained the execution time window for the start and end events of  $e_{AB}$  to be relative to a common reference time point. This is the simplest case of decoupling that requires no communication, first proposed by Hunsberger (Hunsberger 2002).

In Figure 3(b), we additionally have communication links  $e_{AC}$  and  $e_{DE}$ , similar to our motivating example. In this case,  $e_{AB}$  is satisfied by imposing local temporal constraints  $e_{ZA}$  for AUV1,  $e_{ZC}$  and  $e_{CD}$  for the ship, and  $e_{ZE}$  and  $e_{EB}$  for AUV2. Since  $e_{AC}$  and  $e_{DE}$  are contingent constraints, we can assume that they are satisfied by nature, and we have  $v_B - v_A = (v_B - v_E) + (v_E - v_D) + (v_D - v_C) + (v_C - v_A) = [0, 0] + [0, 5] + [30, 50] + [0, 5] = [30, 60]$ , which also satisfies  $e_{AB}$ . In this case, not only do we need to satisfy the external requirement constraint  $e_{AB}$ , due to the existence of communication links, the agents receiving the communication need to have some expectation of when communication will occur. For example, by constraining  $e_{ZA}$ , we are guaranteed that event  $v_C$  will definitely occur some time in between  $v_C - v_Z = (v_C - v_A) + (v_A - v_Z) = [0, 5] + [0, 60] = [0, 65]$ , which is a contingent temporal constraint, since  $v_C$  is not controlled by the ship but can only be observed as it occurs.

With the above intuition, the key idea behind finding these decoupling constraints is that the imposed local decoupling constraints need to be tighter or more restrictive than the external requirement temporal constraints to make them redundant, and they should also ensure that the uncontrollable end

event of each communication link has a corresponding local contingent constraint that covers all of its possible range of time occurrence. Note that the problem of finding valid temporal decoupling constraints is combinatorial, and hence it is encoded as a MILP summarized below. Readers should refer to (Casanova et al. 2016; Zhang and Williams 2021) for more detail.

Given MaSTNU  $\langle N^A, E_X, C_X, v_Z \rangle$ , the MILP formulation includes the following variables, where  $V = \cup_{a \in A} V^a$ :

- (1) Real variables  $u_{ij}$  for  $v_i, v_j \in V$ , with  $u_{ii} = 0$ .
- (2) Boolean variables  $c_{kj}$  for  $(v_i, v_j, v_k) \in T$ , where  $T = \{(v_i, v_j, v_k) | e_{ij} \in C_X, v_k \in V^a(v_j) \setminus \{v_j\}\}$ .
- (3) Boolean variables  $b_{ij}$  for  $(v_i, v_j) \in \overline{E_X}$ , where  $\overline{E_X} = \{(v_i, v_j) | a(v_i) \neq a(v_j), e_{ij} \notin C_X, e_{ji} \notin C_X\}$ .
- (4) Boolean variables  $z_{ijkl}$  for  $(v_i, v_j, v_k, v_l) \in Q$ , where  $Q = \{(v_i, v_j, v_k, v_l) | (v_i, v_j) \in \overline{E_X}, (v_k = v_l = v_Z) \vee ((a(v_k) = a(v_i)) \wedge (e_{lk} \in C_X \vee e_{lk} \in C_X))\}$ .
- (5) Integer variables  $h_{ij} \in [0, H]$  for each tuple  $(v_i, v_j) \in \overline{E_X}$ , where  $H = \max(|A| - 2, |C_X|)$ .

The constraints include the following, where  $l_{ij} = -u_{ji}$ :

- (1)  $\forall v_i, v_j \in V, u_{ij} + u_{ji} \geq 0$
- (2)  $\forall e_{ij} \in E_X, (l_{ij} \geq L_{ij}) \wedge (u_{ij} \leq U_{ij})$
- (3)  $\forall e_{ij} \in C_X, (0 \leq l_{ij} \leq L_{ij}) \wedge (u_{ij} \geq U_{ij})$
- (4)  $\forall e_{ij} \in E_X, (b_{ij} = 1) \wedge (b_{ji} = 1)$
- (5)  $\forall (v_i, v_j) \in \overline{E_X}, b_{ij} = \sum_{v_k, v_l | (v_i, v_j, v_k, v_l) \in Q} z_{ijkl}$
- (6)  $\forall (v_i, v_j, v_k, v_l) \in Q, u_{ij} \geq u_{ik} + u_{kl} + u_{lj} + (z_{ijkl} - 1)M$ , where  $M$  is a large constant
- (7)  $\forall (v_i, v_j, v_k, v_l) \in Q \text{ s.t. } (v_l, v_j) \in \overline{E_X}, z_{ijkl} \leq b_{lj}$
- (8)  $\forall (v_i, v_j, v_k, v_l) \in Q \text{ s.t. } (v_l, v_j) \in \overline{E_X}, h_{ij} + (1 - z_{ijkl})(H + 1) \geq h_{lj} + 1$
- (9)  $\forall e_{ij} \in C_X, \sum_{v_k | (v_i, v_j, v_k) \in T} c_{kj} = 1$
- (10)  $\forall (v_i, v_j, v_k) \in T, (u_{kj} \geq u_{ki} + u_{ij} + (c_{kj} - 1)M) \wedge (0 \leq l_{kj} \leq l_{ki} + l_{ij} + (1 - c_{kj})M)$
- (11)  $\forall (v_i, v_j, v_k) \in T \text{ s.t. } (v_i, v_k) \in \overline{E_X}, c_{kj} \leq b_{ik} \text{ and } \forall (v_i, v_j, v_k) \in T \text{ s.t. } (v_k, v_i) \in \overline{E_X}, c_{kj} \leq b_{ki}$

For our state temporal decoupling algorithm, we make an adaptation to the above encoding since we do not require all the communication links to be used. For example, in Figure 1, even though there is a communication link  $ep_{EE'}$  from AUV1 to AUV2, it may not be used to support the decoupling of any external requirement episodes, in which case we do not need to satisfy any of its state constraints. Therefore, we additionally add a boolean variables  $p_j$  for each  $ep_{ij} \in C_X$ , which denotes whether the communication link is used or not. We add the following constraints:

- $\forall (v_i, v_j, v_k, v_l) \in Q \text{ s.t. } e_{kl} \in C_X, p_l \geq z_{ijkl} \text{ and } \forall (v_i, v_j, v_k, v_l) \in Q \text{ s.t. } e_{lk} \in C_X, p_k \geq z_{ijkl}$ . This says that the communication link must be decoupled if it is used to support an external requirement constraint.

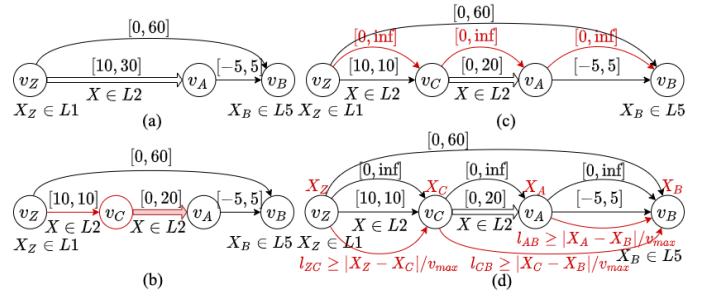


Figure 4: Feasibility for QSP with temporal certainty

- $\forall (v_i, v_j, v_k) \in T \text{ s.t. } e_{lk} \in C_X, p_k \geq c_{kj}$ . This says that the communication link must be decoupled if it is used to support an external contingent constraint.

Additionally, if the end event of a communication link  $ep_{ij}$  has other constraints connected to it, then we need to set  $p_j = 1$  as well. The constraint (9) in Casanova's encoding should be changed to  $\forall e_{ij} \in C_X, \sum_{v_k | (v_i, v_j, v_k) \in T} c_{kj} = p_j$  so that only the used communication links are decoupled.

### Feasibility Encoding for QSP

While the validity encoding ensure that the external episodes are decoupled, the imposed decoupling episodes may over-constrain the local state plans. For MaSTNU, Casanova et al. use the MILP dynamic controllability encoding proposed by Cui et al. to ensure the feasibility of local STNUs (Cui and Haslum 2017). We describe a novel feasibility checking algorithm for QSPs that builds on top of Cui's MILP encoding. While previous path planning algorithms exist for QSPs (Fernández-González, Williams, and Karpas 2018; Reeves, Fernández-González, and Williams 2019; Chen, Williams, and Fan 2021), they often assume a given total ordering of the events and that all the events are executable without any temporal uncertainty. As a result, their solution is typically a deterministic trajectory with a list of waypoints at fixed times. As mentioned, due to the existence of uncontrollable events, our execution strategy for a QSP is a dynamic policy.

To illustrate the high-level idea for finding an execution strategy for QSP, consider a simple QSP in Figure 4(a). First, we convert the QSP into its normal form, where every contingent temporal constraint has a lower bound of 0 (Morris 2006), and no contingent temporal constraint starts from an uncontrollable event, as shown in Figure 4(b). This can be achieved by rewriting each contingent episode into a requirement episode with a fixed duration equal to the original lower bound, followed by a contingent episode with a lower bound of 0. Second, we enforce a total ordering of the events at which state variables are constrained, as shown in Figure 4(c). Notice that we only need to order the events that have state constraints, since for any event without any state constraints, we do not care what values the state variables take at those events. Finally, given the ordering, we can express the reachability constraint by specifying how long it takes at least for the agent to go from one location to the next as the lower bound between those two events, as in Figure 4(d). We then check the feasibility of QSP by checking its dy-

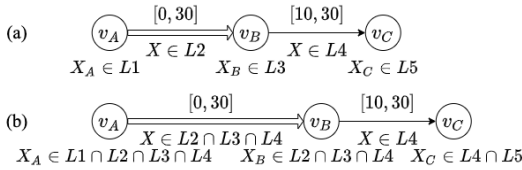


Figure 5: (a) Example QSP with 5 state constraints (b) Effective state constraints for example QSP

dynamic controllability. We now describe the MILP encoding in detail in the order of state constraints, ordering and reachability constraints, and dynamic controllability constraints.

**(1) Encoding State and Delta Constraints** First, we encode all the continuous state and delta constraints, which includes:  $SC_{start}$ ,  $SC_{end}$ ,  $SC_{overall}$ ,  $DC$ . We denote the set of *constrained events* as  $V_{sc} = \{v | v = s(ep(c)), \forall c \in SC_{start} \cup SC_{overall} \cup DC \text{ or } v = t(ep(c)), \forall c \in SC_{end} \cup SC_{overall} \cup DC\}$ , where  $ep(c)$  denotes the episode that the constraint  $c$  belongs to, and  $s(ep)$ ,  $t(ep)$  denote the start and end event of the episode  $ep$ , respectively. For each state variable  $x \in X$  and for each constrained event  $v_i \in V_{sc}$ , we create a continuous variable  $x_i$  that represents the value of the state variable at that event. We denote  $X_i$  as the vector of continuous variables for state variables  $X$  at event  $v_i$ .

To encode the satisfaction of state constraints, consider the example in Figure 5(a) with five state constraints. In this case, since both  $X \in L2$  and  $X \in L4$  are overall state constraints to be satisfied throughout the episodes, their start and end events need to satisfy those state constraints too. Additionally, since  $ep_{AB}$  includes a contingent temporal constraint with a lower bound of 0, meaning that event  $v_B$  is an uncontrollable event that may occur any time on or after event  $v_A$ , any state constraint that needs to be satisfied at  $v_B$  must also be satisfied at  $v_A$  as well as throughout episode  $ep_{AB}$ . Therefore, the effective state constraints that need to be satisfied and encoded for  $V_{sc}$  are shown in Figure 5(b). Note that since we assume normal form of the QSP, event  $v_A$  cannot be another uncontrollable event and the propagation of state constraints due to contingent constraints is limited. Additionally, for any overall state constraint  $sc$  of an episode, any constrained event ordered in between the episode must also satisfy the constraint. Based on the MILP dynamic controllability encoding,  $l_{ij}$  denotes the continuous variable for the lower bound between event  $v_i$  and  $v_j$ . Therefore, we enforce the overall state constraint  $sc \in SC_{overall}$  for an episode  $ep_{AB}$  using the following constraint:

$$(C1) \quad \forall v_D \in V_{sc}, (l_{DA} \geq 0) \vee (l_{BD} \geq 0) \vee sc(X_D).$$

Intuitively, this says that either  $v_D$  is ordered before  $v_A$ , or  $v_D$  is ordered after  $v_B$ , or the state constraint has to be satisfied at event  $v_D$ . Note that in the case of  $v_D$  being an uncontrollable event with contingent episode  $ep_{GD}$ , as mentioned, any state constraint that applies to  $v_D$  should apply as an overall state constraint for the entire episode  $ep_{GD}$ . If  $v_G$  is ordered before  $v_A$ , based on our reachability analysis in the following section,  $X_A = X_G$  must hold to satisfy dynamic controllability, and any event  $v_E$  ordered in between  $ep_{GD}$

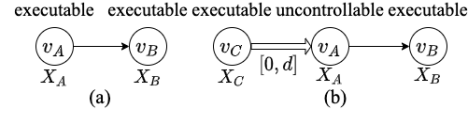


Figure 6: Examples for ordering and reachability constraints

also satisfies  $X_G = X_E$ , which automatically satisfies the overall state constraint.

**(2) Encoding Ordering & Reachability** In order to enforce a total ordering between constrained events and reachability between pairs of events, we can combine them into the following constraints. Note that we assume agents have first-order dynamics, and our total ordering is a weak total ordering that allows events to occur simultaneously.

- (C2)  $\forall v_A, v_B \in V_{sc}$  such that  $v_A, v_B$  are executable,  $\forall x \in X, (l_{AB} \geq |x_A - x_B|/v_{max}) \vee (l_{BA} \geq |x_A - x_B|/v_{max})$
- (C3)  $\forall v_A, v_B \in V_{sc}$  such that  $v_A$  or  $v_B$  is uncontrollable,  $(l_{AB} \geq 0) \vee (l_{BA} \geq 0)$
- (C4)  $\forall v_A, v_B \in V_{sc}$  such that  $v_A$  is uncontrollable and  $v_B$  is executable,  $\forall x \in X, (l_{BA} \geq 0) \vee (l_{AB} \geq |x_A - x_B|/v_{max})$

For C2 (Figure 6(a)), when  $v_A$  and  $v_B$  are both executable events with state variable values  $X_A$  and  $X_B$ , then assuming first-order dynamics, we know it takes at least  $\max_{x \in X} |x_A - x_B|/v_{max}$  time to go from one event to another. Therefore, we order them by imposing that the temporal lower bound either from  $v_A$  to  $v_B$  or from  $v_B$  to  $v_A$  has to be greater or equal to the above. For C3 (Figure 6(b)), when any of  $v_A$  or  $v_B$  is an uncontrollable event, then this constraint only enforces the ordering between the two events. C2 and C3 together enforces a global total ordering of all the constrained events. Note that any contingent episode such as  $ep_{CA}$  must satisfy  $lb_{CA} \geq 0$  by definition, which satisfies C3. For C4 (Figure 6(b)), when event  $v_B$  is an executable event ordered after an uncontrollable event  $v_A$ , then there is a reachability constraint from  $v_A$  to  $v_B$ .

An example of enforced reachability constraints assuming given order can be seen in Figure 4(d). Note that we omitted the reachability constraint from  $v_Z$  to  $v_B$  to avoid cluttering, since it is dominated by other reachability constraints. Notice that without a reachability constraint from  $v_C$  to  $v_A$ ,  $X_A$  can take any value within the range of its state constraints, and we do not strictly require the agent to be at  $X_A$  at event  $v_A$ . To understand the execution strategy, we will focus on a minimal QSP in Figure 6(b) involving a contingent episode  $ep_{CA}$  and the first executable event  $v_B$  ordered after  $v_A$ , since the execution strategy for any consecutive executable events is simple. In order to describe the execution strategy, we will write the QSP's underlying temporal network in its labeled distance graph form (Morris 2006) in Figure 7, where  $l_{ij}$  ( $u_{ij}$ ) denotes the lower bound (upper bound) from  $v_i$  to  $v_j$ . According to C2 and C4, we have reachability constraints  $l_{CB} \geq |X_C - X_B|/v_{max}$  and  $l_{AB} \geq |X_A - X_B|/v_{max}$ . Assuming that  $X_A$  is constrained

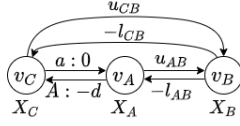


Figure 7: Execution policy in labeled distance graph

to be in region  $L2$ , then it means that  $X_C$  must be within region  $L2$  too. The execution policy involves the agent starting at location  $X_C$  at event  $v_C$ , and going towards  $X_B$ . If the agent reaches the boundary for region  $L2$ , but hasn't received event  $v_A$ , then it will be stuck at the boundary until  $v_A$  occurs. When  $v_A$  occurs, the agent continues going towards  $X_B$ . The execution policy is feasible if the resulting network is dynamically controllable, because based on the dynamic controllability constraints:

- $u_{CB} - l_{CB} \geq 0$ . This ensures that there is enough time on  $u_{CB}$  for the agent to go from  $X_C$  to  $X_B$  non-stop.
- $u_{CB} - l_{AB} - d \geq 0$ . If the agent gets stuck at  $L2$  boundary, then in the worst case, it needs to wait until  $d$  time has passed before continuing its way to  $X_B$ . This ensures that we can find such a location  $X_A$  within the  $L2$  boundary such that there is enough time on  $u_{CB}$  for the agent to wait for  $d$  time and go from  $X_A$  to  $X_B$ .
- $u_{AB} - l_{AB} \geq 0$ . This ensures that if the agent is stuck at  $L2$  boundary until  $v_A$  occurs, there is enough time on  $u_{AB}$  for it to go from the boundary point  $X_A$  to  $X_B$ .
- $u_{AB} + 0 - l_{CB} \geq 0$ . This ensures that if event  $v_A$  occurs immediately after event  $v_C$ , there is enough time on  $u_{AB}$  for the agent to go from  $X_C$  to  $X_B$ .

Note that it is possible for other events to be ordered in between  $v_C$  and  $v_A$ . If an executable event  $v_D$  is ordered in between  $v_C$  and  $v_A$ , then reachability and dynamic controllability constraints require that  $X_C = X_D$ . If an uncontrollable event  $v_E$  is ordered in between  $v_C$  and  $v_A$ , and its corresponding contingent episode is  $ep_{GE}$ , then it requires that  $X_C = X_G$  and  $v_C = v_E$ , that is, event  $v_C$  is scheduled immediately when  $v_E$  is received. We leave it to the reader to validate the feasibility of execution policies in these cases.

**(3) Encoding Dynamic Controllability** We summarize the MILP constraints that ensure the dynamic controllability of the local plans (Cui and Haslum 2017; Wah and Xin 2007). Given a STNU  $\langle V, E, C \rangle$ , where  $V$  is the set of events,  $E$  is the set of temporal requirement constraints, and  $C$  is the set of temporal contingent constraints, the MILP formulation includes the following variables, where  $V_E$  denotes the set of executable events. Note that  $l_{ij} = -u_{ji}$ .

- (1) Real variables  $u_{ij}$  for  $v_i, v_j \in V$ , with  $u_{ii} = 0$
- (2) Real variables  $w_{ijk}$  for  $e_{ik} \in C, v_j \in V_E$ , with  $w_{iik} = 0$

The MILP constraints are listed below:

- (1)  $\forall e_{ij} \in E \cup C, (l_{ij} \geq L_{ij}) \wedge (u_{ij} \leq U_{ij})$
- (2)  $\forall e_{ij} \in C, (0 \leq l_{ij} \leq L_{ij}) \wedge (u_{ij} \geq U_{ij})$
- (3)  $\forall v_i, v_j, v_k \in V, u_{ij} \leq u_{ik} + u_{kj}$

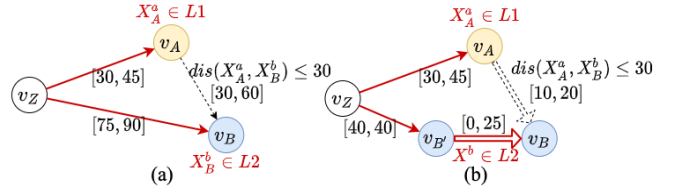


Figure 8: Decouple delta constraint for (a) external requirement episode (b) external contingent episode

- (4)  $\forall e_{ik} \in C, \forall v_j \in V_E, (l_{jk} < 0) \vee ((u_{ij} \leq l_{ik} - l_{jk}) \wedge (l_{ij} \geq u_{ik} - u_{jk}))$
- (5)  $\forall e_{ik} \in C, \forall v_j \in V_E, u_{ik} - u_{jk} \leq w_{ijk}$
- (6)  $\forall e_{ik} \in C, \forall v_j \in V_E, \min(l_{ik}, w_{ijk}) \leq l_{ij}$
- (7)  $\forall e_{ik} \in C, \forall v_j, v_m \in V_E, w_{ijk} - u_{mj} \leq w_{imk}$
- (8)  $\forall e_{ik}, e_{mj} \in C, (w_{ijk} < 0) \vee (w_{ijk} - l_{mj} \leq w_{imk})$

When the external contingent episodes are decoupled, additional local contingent episodes may be introduced as part of the decoupling episodes. Therefore, the above encoding needs to be adapted to handle these optional local contingent constraints. Refer to (Casanova et al. 2016) for detail.

### State Validity Encoding

To ensure the validity of decoupling, the external state and delta constraints across agents should be decoupled too. We describe how to decouple  $DC$  and  $SC_{overall}$  next. For simplicity, we assume that the events of an external episode  $ep_{AB}$  are executable events, except for the uncontrollable end event when  $ep_{AB}$  is an external contingent episode. This assumption can be removed with some more analysis. When encoding state validity constraints, for a communication link  $ep_{ij}$ , we condition its MILP constraints on  $p_j = 1$  so that they are only enforced when  $ep_{ij}$  is used.

First, as shown in Figure 8, consider an external episode  $ep_{AB}$  with a delta constraint  $dc(X_A^a, X_B^b) \in DC$ . To find the state decoupling for  $dc(X_A^a, X_B^b)$ , it suffices to find  $L1, L2$  such that for any  $X_A^a \in L1$  and for any  $X_B^b \in L2$ ,  $dc(X_A^a, X_B^b)$  always holds. In this way, we have decoupled the external delta constraint by restricting locally for each agent an area that it should be in at the specific start and end event of the episode. Note that if  $ep_{AB}$  is a contingent episode (Figure 8(b)), then its decoupling involves an introduced local contingent episode  $ep_{ZB}$  in its normal form, or more specially, a requirement episode  $ep_{ZB'}$  and a contingent episode  $ep_{B'B}$ . Because  $v_B$  is an uncontrollable event,  $X_B^b \in L2$  has to be satisfied over the entire  $ep_{B'B}$ .

To encode the above constraint in MILP, while it is possible to directly find such regions  $L1, L2$  approximated by a set of points as a polygon, in this paper, we directly enforce the constraint  $dc(X_A^a, X_B^b)$ . If  $ep_{AB}$  is a contingent episode, we additionally enforce  $dc(X_A^a, X_{B'}^b)$ . Once a MILP solution is found, we can read off region  $L1$  as the point  $X_A^a$ , and region  $L2$  as the point  $X_B^b$  or the region enclosed by  $X_B^b$  and  $X_{B'}^b$  in the case of a contingent episode. As a post-processing step, we can optionally relax the region  $L1$  and



$L2$  such that the decoupling is still valid to provide more flexibility to the agents.

Second, as shown in Figure 9(a), consider an external requirement episode  $ep_{AB}$  with an overall state constraint  $sc(X^a, X^b) \in SC_{overall}$ . In this case, the resulting decoupling introduces copies of event  $v_A$  and event  $v_B$  and external temporal constraints  $e_{AA'}$  and  $e_{B'B}$  with duration 0. The decoupling of  $e_{AA'}$  and  $e_{B'B}$  can be handled by the temporal validity encoding. We can similarly find regions  $L1$  and  $L2$  such that  $\forall v_A \leq T_1 \leq v_{B'}, X_{T_1}^a \in L1$  for agent  $a$  and  $\forall v_{A'} \leq T_2 \leq v_B, X_{T_2}^b \in L2$  for agent  $b$ ,  $sc(X_{T_1}^a, X_{T_2}^b)$ . In order to encode the above constraints in MILP, we enforce the constraints  $sc(X_{A'}^a, X_{A'}^b)$ ,  $sc(X_{A'}^a, X_B^b)$ ,  $sc(X_{B'}^a, X_{A'}^b)$  and  $sc(X_{B'}^a, X_B^b)$ . Additionally, for any constrained events in between the two episodes, we enforce the overall state constraint:  $\forall v_D^a \in V_{sc}^a, \forall v_E^b \in V_{sc}^b, (l_{DA} \geq 0) \vee (l_{B'D} \geq 0) \vee (l_{EA'} \geq 0) \vee (l_{BE} \geq 0) \vee sc(X_D^a, X_E^b)$ . Finally, we can read off region  $L1$  as the convex region enclosed by the state variable values in between  $v_A$  and  $v_{B'}$ , and similarly for  $L2$ .  $L1, L2$  can also be relaxed in post-processing.

Finally, as shown in Figure 9(b), consider an external contingent episode  $ep_{AB}$  with an overall state constraint  $sc(X^a, X^b) \in SC_{overall}$ . The resulting decoupling involves a local requirement temporal constraint  $e_{ZA}$  and a local contingent episode  $ep_{AC}$  for agent  $a$ , where  $ep_{AC}$  has an overall state constraint  $X^a \in L1$  that requires agent  $a$  to stay in region  $L1$  throughout the communication period. Note that we assume in this case, agent  $a$  receives event  $v_C$  upon communication  $ep_{AB}$  finishes, since it is often used to model data transmission that takes up a period of time. The decoupling also involves a local requirement temporal constraint  $e_{ZA'}$ , a requirement episode  $ep_{A'B'}$  and a contingent episode  $ep_{B'B}$  for agent  $b$ , where  $ep_{A'B'}$  and  $ep_{B'B}$  are under the overall state constraint  $X^b \in L2$ . We encode the constraints in MILP in a similar fashion as before.

## Preliminary Experiments

We evaluate our algorithm on two AUV team scenarios. All experiments were run on 3.40GHZ 8-Core Intel Core i7-6700 CPU with 39GB RAM, and the MILP encoding was solved using Gurobi 9.1.2, with a timeout of 100 seconds. Note that the MILP encoding also allows the specification of an objective function, which affects the runtime. We evaluate the algorithm on three objective cases: (obj1) no objective function, (obj2) minimize the use of communication links, (obj3) minimize  $\sum_{v_i \in V} u_{Zi}$ .

For our motivating example, we record the average runtime for 3 communication scenarios: (ST1) only  $ep_{EE'}$  is available, which the algorithm finds no decoupling solution, (ST2)  $ep_{FF'}$  and  $ep_{GG'}$  are available, and (ST3) all communication links are available. We also test the example for temporal decoupling only by framing it as a MaSTNU (T3), and for feasibility only by framing it as a QSP (QSP) that assumes full observability between agents. We repeat the experiments by adding two other missions to each AUV (tests denoted by \*). The result is shown in Table 1. The results show that the choice of objective functions can have a large impact on the runtime. Additionally, the runtime increases

test	obj1	obj2	obj3	test	obj1	obj2	obj3
ST1	0.24	0.21	0.17	ST1*	2.87	3.33	5.28
ST2	0.4	0.3	0.71	ST2*	4.46	3.15	43.14(1)
ST3	4.23	2.56	24.9	ST3*	18.6(1)	47.39(4)	N/A(10)
T3	0.09	0.06	0.94	T3*	0.19	0.19	2.31
QSP	0.04	N/A	0.04	QSP*	6.84	N/A	23.6

Table 1: Average runtime in seconds over 10 runs for different tests, with the number of timed out runs in parenthesis

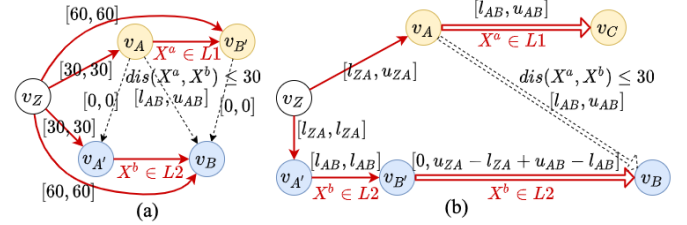


Figure 9: Decouple state constraint of type  $SC_{overall}$  for (a) external requirement episode (b) external contingent episode

quite drastically as the number of communication links increases. Note that for temporal decoupling, there exists a solution even with only  $ep_{EE'}$ , and the added missions are not totally ordered since no state constraints are enforced.

In a second scenario, the ship deploys an AUV in a region, and the AUV carries out two sampling missions at different science sites. When both missions are done, the AUV transmits data through a communication link back to the ship. The transmission may take any time between 20 to 30 minutes, during which they have to stay within 30 meters of each other. The AUV and the ship needs to stay within 100 meters of each other throughout the two sampling missions, and the ship has its own imaging mission to do before a certain deadline such that it has to be carried out concurrently while the AUV is on its sampling missions. Finding a solution takes 1.1 secs on average with obj1 and 1.6 secs with obj3.

Note that our QSP feasibility encoding finds an execution strategy that fixes the state variable values at constrained executable events, meaning for an executable event  $v_i \in V_{sc}$  following an uncontrollable event  $v_j \in V_{sc}$  with  $[0, 0]$  temporal bound, we exclude any execution strategies where state variables at  $v_i$  can take any non-deterministic value that is taken at  $v_j$ . Future work can investigate if this assumption can be relaxed. we also assume no obstacles in the environment and simple dynamics of the vehicles. Future work can build on top of our encoding to allow richer path planning constraints and agent dynamics.

## Conclusion

In this paper, we introduced the framework of Multi-Agent Qualitative State Plan (MaQSP) to model multi-agent plans with coupled temporal and state constraints, where agents are subject to limited communication during execution. We proposed a state temporal decoupling algorithm for MaQSP based on MILP encoding, which includes a novel path planning algorithm for QSPs with temporal uncertainty that may be useful in other applications.



## Acknowledgments

This material is based upon work supported by the Defense Advanced Research Projects Agency (DARPA) under Contract No. HR001120C0035. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Defense Advanced Research Projects Agency (DARPA).

## References

- Boerkoel Jr, J. C.; and Durfee, E. H. 2013. Distributed reasoning for multiagent simple temporal problems. *Journal of Artificial Intelligence Research* 47: 95–156.
- Casanova, G.; Pralet, C.; Lesire, C.; and Vidal, T. 2016. Solving dynamic controllability problem of multi-agent plans with uncertainty using mixed integer linear programming. In *Proceedings of the Twenty-second European Conference on Artificial Intelligence*, 930–938. IOS Press.
- Chen, J.; Williams, B. C.; and Fan, C. 2021. Optimal Mixed Discrete-Continuous Planning for Linear Hybrid Systems. In *Proceedings of the 24th International Conference on Hybrid Systems: Computation and Control, HSCC '21*. New York, NY, USA: Association for Computing Machinery. ISBN 9781450383394. doi:10.1145/3447928.3456654. URL <https://doi.org/10.1145/3447928.3456654>.
- Cui, J.; and Haslum, P. 2017. Dynamic controllability of controllable conditional temporal problems with uncertainty. In *27th International Conference on Automated Planning and Scheduling (ICAPS 2017)*.
- Dechter, R.; Meiri, I.; and Pearl, J. 1991. Temporal constraint networks. *Artificial intelligence* 49(1-3): 61–95.
- Fernández-González, E.; Williams, B.; and Karpas, E. 2018. ScottyActivity: Mixed Discrete-Continuous Planning with Convex Optimization. *J. Artif. Intell. Res.* 62: 579–664.
- Hunsberger, L. 2002. Algorithms for a temporal decoupling problem in multi-agent planning. In *AAAI/IAAI*.
- Léauté, T.; and Williams, B. C. 2005. Coordinating agile systems through the model-based execution of temporal plans. In *AAAI*, 114–120.
- Morris, P. 2006. A structural characterization of temporal dynamic controllability. In *International Conference on Principles and Practice of Constraint Programming*, 375–389. Springer.
- Reeves, M.; Fernández-González, E.; and Williams, B. 2019. Executing Multi-Goal Mission Plans for Coordinated Mobile Robots. In *ICAPS 2019 INTEX workshop*.
- Vidal, T. 1999. Handling contingency in temporal constraint networks: from consistency to controllabilities. *Journal of Experimental & Theoretical Artificial Intelligence* 11(1): 23–45.
- Wah, B. W.; and Xin, D. 2007. Optimization of bounds in temporal flexible plans with dynamic controllability. *International Journal on Artificial Intelligence Tools* 16(01): 17–44.
- Zhang, Y.; and Williams, B. C. 2021. Privacy-Preserving Algorithm for Decoupling of Multi-Agent Plans with Uncertainty. In *Proceedings of the 31st International Conference on Automated Planning and Scheduling (ICAPS)*.

# Non-monotonic Logical Reasoning Guiding Axiom Induction from Deep Networks for Transparent Decision Making in Robotics

Tiago Mota<sup>1</sup> and Mohan Sridharan<sup>2</sup>

<sup>1</sup>Electrical and Computer Engineering, The University of Auckland, New Zealand  
tmot987@aucklanduni.ac.nz

<sup>2</sup>School of Computer Science, University of Birmingham, United Kingdom  
m.sridharan@bham.ac.uk

## Abstract

This paper describes our architecture developed over the last few years to provide transparency in decision making in integrated robot systems that include knowledge-based reasoning methods and data-driven learning methods. It couples the complementary strengths of non-monotonic logical reasoning with incomplete commonsense domain knowledge, deep learning, and inductive learning. During reasoning and learning, the robot interactively provides on-demand *explanations* of its decisions and evolution of beliefs as relational descriptions of relevant objects, attributes, and actions. The architecture's capabilities are evaluated in the context of visual scene understanding and planning based on simulated images and images from a physical robot manipulating tabletop objects. Results indicate the ability to reliably acquire previously unknown domain knowledge, provide accurate explanations, and to eliminate ambiguities in the human queries.

## 1 Introduction

Imagine a robot arranging objects in desired configurations on a table, and estimating the occlusion of objects and stability of object configurations, e.g., Figure 1a. An object is occluded if any fraction of its frontal face is hidden, and an object configuration is unstable if any object in it is unstable. To perform these tasks, the robot extracts information from on-board camera images, reasons with this information and incomplete domain knowledge, and executes suitable actions. It also learns previously unknown axioms governing actions and change, and responds to questions about its decisions and evolution of beliefs. For instance, assume the goal in Figure 1b is to have the yellow ball on the orange block, and the plan is to move the blue block on to the table before placing the ball on the orange block. The robot answers questions about planned, executed, or hypothetical actions, e.g., "why do you want to pick up the blue block first?" and "why did you not pick up the pig?", by exploring the evolution of related beliefs; it also poses disambiguation questions, e.g., it responds to the human question "why did you not pick up the orange object?" about Figure 1a with "are you referring to the orange block on the red block?".

Our architecture seeks to jointly address the knowledge representation, reasoning, learning, and control challenges



(a) Test scenario.



(b) Robot camera image.

Figure 1: (a) Motivating scenario: Baxter arranging objects in target configurations; (b) Image from left gripper camera.

posed by the motivating scenario. In this paper, we focus on the ability to provide on-demand *explanations* of decisions and evolution of beliefs in the form of descriptions comprising relations between relevant objects, object attributes, actions, and robot attributes; action execution is based on the robot's built-in abilities (e.g., to move to a location). Providing such explanations can improve the algorithms and establish accountability, but it is difficult to do so in integrated robot systems that use knowledge-based reasoning methods (e.g., for planning) and data-driven learning methods (e.g., for pattern recognition). Our architecture draws on cognitive systems research, which highlights the benefits of coupling different representations, reasoning schemes, and learning methods (Laird 2012; Winston and Holmes 2018) to:

- Combine the principles of non-monotonic logical reasoning and deep learning for decision making;
- Automatically learn previously unknown axioms of state constraints, and action preconditions and effects;
- Automatically trace the evolution of any given belief or the (non)selection of any given action by inferring the relevant sequence of axioms and beliefs; and
- Exploit the interplay between representation, reasoning, and learning to describe decisions and beliefs related to computed or executed plans and hypothetical situations.

These capabilities are evaluated in the context of a robot: (i) computing and executing plans to arrange objects in desired configurations; and (ii) estimating occlusion of scene objects and stability of object configurations, in simulated scenes and in the real world. Results indicate the ability to: (i) incrementally learn previously unknown axioms governing domain dynamics; and (ii) construct explanations reliably and efficiently by automatically identifying and reasoning with the relevant knowledge and posing disambiguation

questions when needed. This work is described in a journal article (Mota, Sridharan, and Leonardis 2021) and an upcoming conference paper (Mota and Sridharan 2021), and initial versions have appeared at other venues (Mota and Sridharan 2019, 2020a). We first discuss related work (Section 2), and then describe the architecture (Section 3), experimental results (Section 4), and conclusions (Section 5).

## 2 Related Work

Understanding the operation of AI and machine learning methods can be used to improve these methods, and to make automated decision-making more acceptable to humans (Lewandowsky, Mundy, and Tan 2000). Recent work in *explainable AI* and *explainable planning* (Miller 2019) can be broadly categorized into two groups. Methods in one group modify or map learned models or reasoning systems to make their decisions more interpretable (Ribeiro, Singh, and Guestrin 2016) or easier for humans to understand (Zhang et al. 2017). Methods in the other group provide descriptions that make a reasoning system’s decisions more transparent (Borgo, Cashmore, and Magazzeni 2018), help humans understand plans (Bercher et al. 2014), and help justify solutions obtained by non-monotonic logical reasoning (Fandinno and Schulz 2019). Much of this research is agnostic to how an explanation is structured or assumes comprehensive knowledge.

Given their use in different applications, there is much interest in understanding the operation of deep networks, e.g., by computing the features most relevant to the estimated outputs (Assaf and Schumann 2019; Samek, Wiegand, and Miller 2017). There has also been work on reasoning with learned symbolic structure or a graph encoding scene structure, in conjunction with deep networks to answer questions about images (Norcliffe-Brown, Vafeais, and Parisot 2018; Yi et al. 2018). However, these approaches: (i) do not fully integrate reasoning and learning to inform and guide each other; or (ii) do not use the rich commonsense domain knowledge for reliable and efficient reasoning, learning, and the descriptions of the system’s decisions and beliefs.

There are many methods for learning logic-based representations of domain knowledge, e.g., incremental revision of action operators in first-order logic (Gil 1994), inductive learning of domain knowledge as an Answer Set Prolog program (Law, Russo, and Broda 2020), and work in our group on coupling non-monotonic logical reasoning and relational reinforcement learning to learn axioms (Mota, Sridharan, and Leonardis 2020; Sridharan and Meadows 2018). Our approach for learning domain axioms is inspired by work in interactive task learning (Laird et al. 2017); unlike methods that learn from many training examples, our approach learns from limited training examples.

Our work focuses on integrated robot systems that use knowledge-based and data-driven methods to reason with and learn from incomplete domain knowledge and observations. We enable such robots to generate relational descriptions of decisions, evolution of beliefs, and counterfactual situations. Recent surveys state that these capabilities are not supported by existing systems (Anjomshoae et al. 2019; Miller 2019). Our architecture extends work in our group on

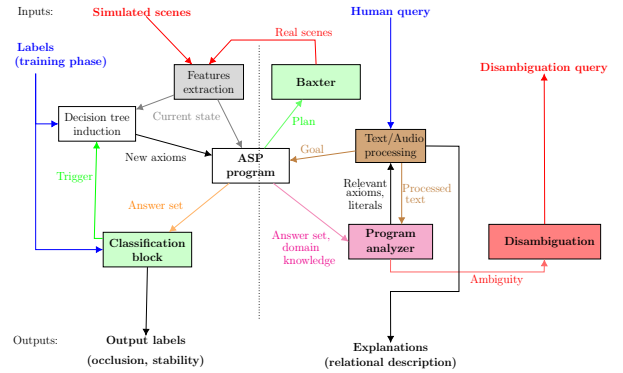


Figure 2: Architecture combines strengths of non-monotonic logical reasoning with incomplete commonsense knowledge, deep learning, and inductive learning.

explainable agency (Langley et al. 2017), a theory of explanations (Sridharan and Meadows 2019), and on combining non-monotonic logical reasoning and deep learning for classification of simulated images (Mota and Sridharan 2019).

## 3 Architecture

Figure 2 is an overview of our architecture. Components to the left of the dashed vertical line combine non-monotonic logical reasoning and deep learning for decision making. Components to the right of the dashed line expand reasoning capabilities, answer questions about decisions and evolution of beliefs, and construct disambiguation queries. We describe all components, focusing on the recent developments, using the following example domain.

### Example Domain 1 [Robot Assistant (RA) Domain]

A Baxter robot: (i) estimates occlusion of scene objects and stability of object structures, and arranges objects in desired configurations; and (ii) provides on-demand relational descriptions of decisions and evolution of beliefs. There is uncertainty in the robot’s perception and actuation, and the robot uses probabilistic algorithms to visually recognize and move objects. The robot has incomplete (and potentially imprecise) domain knowledge, which includes object attributes such as *size*, *surface*, and *shape*; spatial relations between objects (above, below, front, behind, right, left, in); some domain attributes; and some axioms governing domain dynamics such as:

- Placing an object on top of an object with an irregular surface results in an unstable object configuration.
- For any given object, removing all objects blocking the view of any minimal fraction of its frontal face causes this object to be not occluded.
- An object below another object cannot be picked up.

This knowledge may need to be revised over time, e.g., some axioms and the value of some attributes may not be known, or the robot may find that placing certain objects on an object with an irregular surface results in a stable configuration.

### 3.1 Representation, Reasoning, and Learning

We first describe the knowledge representation, reasoning, and learning components of the architecture.

**Knowledge Representation** To represent and reason with domain knowledge, we use CR-Prolog, an extension to Answer Set Prolog (ASP) that introduces *consistency restoring* (CR) rules; we use “CR-Prolog” and “ASP” interchangeably.

A domain’s description in ASP comprises a *system description*  $\mathcal{D}$  and a *history*  $\mathcal{H}$ .  $\mathcal{D}$  comprises a *sorted signature*  $\Sigma$  and axioms encoding the domain’s dynamics.  $\Sigma$  comprises *basic sorts*, *statics*, i.e., domain attributes that do not change over time, *fluents*, i.e., attributes whose values can be changed, and *actions*; statics, fluents, and actions are described in terms of the sorts of their arguments. In the RA domain, the robot needs to reason about spatial relations between objects, and to plan and execute actions that change the domain. Such a dynamic domain is modeled in our architecture by first describing  $\Sigma$  and the transition diagram in action language  $\mathcal{AL}_d$  (Gelfond and Inclezan 2013); this description is then translated to ASP statements. The basic sorts include *object*, *robot*, *size*, *relation*, *surface*, and *step* for temporal reasoning; statics include *obj\_size(object, size)* and *obj\_surface(object, surface)*; fluents include *obj\_relation(relation, object, object)*, e.g., *obj\_relation(above, A, B)* implies object *A* is *above* object *B*, and *in\_hand(robot, object)*; and actions include *pickup(robot, object)* and *putdown(robot, object, location)*. In addition, the relation *holds(fluent, step)* implies that a particular fluent holds true at a particular timestep.

Given the  $\Sigma$ , axioms of a domain consist of causal laws, state constraints, and executability conditions. For the RA domain, these are translated to ASP statements such as:

$$\text{holds}(\text{obj\_relation}(\text{on}, \text{Ob}_1, \text{Ob}_2), I + 1) \leftarrow \quad (1a)$$

$$\text{occurs}(\text{putdown}(\text{rob}_1, \text{Ob}_1, \text{Ob}_2), I)$$

$$\text{holds}(\text{obj\_relation}(\text{above}, \text{Ob}_1, \text{Ob}_2), I) \leftarrow \quad (1b)$$

$$\text{holds}(\text{obj\_relation}(\text{below}, \text{Ob}_2, \text{Ob}_1), I)$$

$$\neg \text{occurs}(\text{pickup}(\text{rob}_1, \text{Ob}_1), I) \leftarrow \quad (1c)$$

$$\text{holds}(\text{obj\_relation}(\text{below}, \text{Ob}_1, \text{Ob}_2), I)$$

which encode a causal law, a state constraint, and an executability condition respectively, e.g., Statement 1(a) implies that executing the *putdown* action causes the object in the robot’s grasp ( $\text{Ob}_1$ ) to be on top of object ( $\text{Ob}_2$ ) in the next time step. The axioms also encode some commonsense knowledge in the form of default statements that hold unless stated otherwise, e.g., “larger objects placed on smaller objects are unstable” is encoded in ASP as:

$$\neg \text{holds}(\text{stable}(A), I) \leftarrow \quad (2)$$

$$\text{holds}(\text{obj\_relation}(\text{above}, A, B), I),$$

$$\text{size}(A, \text{large}), \text{size}(B, \text{small}),$$

$$\text{not holds}(\text{stable}(A), I)$$

where “not” denotes default negation. In addition to axioms, information from the input images (e.g., spatial relations, object attributes) with sufficiently high probability is converted to ASP statements at that time step. Also, the domain’s history  $\mathcal{H}$  comprises records of fluents observed to be true or false at a particular time step, i.e., *obs(fluent, boolean, step)*, and of the execution of an action at a particular time step, i.e., *hpd(action, step)*. This

notion can be expanded to include defaults describing the values of fluents in the initial state (Sridharan et al. 2019).

For reasoning, our architecture constructs the CR-Prolog program  $\Pi(\mathcal{D}, \mathcal{H})$ , which includes  $\Sigma$  and axioms of  $\mathcal{D}$ , inertia axioms, reality checks, closed world assumptions for actions, and observations, actions, and defaults from  $\mathcal{H}$ ; see our open-source repository (Mota and Sridharan 2020b). Planning, diagnostics, and inference is reduced to computing *answer sets* of  $\Pi$ . Each answer set represents the robot’s beliefs in a possible world; the literals of fluents and statics at a time step represent the *state* at that time step. Although incorrect inferences can occur due to incomplete knowledge or noisy sensor inputs, non-monotonic logical reasoning enables recovery from such errors. In addition, others in our group have combined logical reasoning at a coarse resolution with probabilistic reasoning over the relevant part of a finer resolution domain representation (Sridharan et al. 2019). To focus on the interplay between non-monotonic logical reasoning and learning, we limit ourselves to logical reasoning at one resolution in this paper.

**Classification Block (CNNs)** In our architecture, for any given image, the robot tries to perform the estimation tasks (e.g., occlusion of objects, stability of object configurations) using ASP-based reasoning. If an answer is not found, or an incorrect answer is found (on training examples), the robot automatically extracts relevant regions of interest (ROIs) from the corresponding image. Parameters of existing Convolutional Neural Network (CNN) architectures (e.g., Lenet (LeCun et al. 1998), AlexNet (Krizhevsky, Sutskever, and Hinton 2012)) are tuned to map information from each such ROI to the corresponding classification labels. The robot reasons with task knowledge (e.g., estimating occlusion) to automatically identify and ground only the relevant axioms and relations to determine the ROIs (Mota and Sridharan 2019); the notion of relevance is also expanded to construct explanations efficiently in Section 3.2.

**Decision Tree induction of Rules** In our architecture, images used to train the CNNs are considered to contain information about missing or incorrect constraints related to the estimation tasks (occlusion, stability). Image features and spatial relations extracted from ROIs in each such image, and the known occlusion and stability labels (during training), are used to incrementally learn a decision tree summarizing the corresponding state transitions; this process repeatedly splits nodes based on unused attributes likely to provide the highest entropy reduction. Trees are learned separately for different actions, and branches of a tree that satisfy minimal thresholds on purity at the leaf ( $\geq 95\%$  samples in one class) and on the level of support from labeled examples ( $\geq 5\%$ ) are used to construct candidate constraints. Candidates without a minimal level of support ( $\geq 5\%$ ) on unseen examples are removed. These thresholds are set to identify highly likely axioms; small changes to thresholds do not affect performance and the thresholds can be revised for other outcomes, e.g., lowered to identify default constraints. Also, to handle noisy images, we only retain axioms identified over a number of cycles of learning and validation. In addition, different versions of the same axiom are merged to

remove over-specifications, e.g.:

$$\neg \text{stable}(A) \leftarrow \text{obj\_relation}(\text{above}, A, B), \quad (3a)$$

$$\text{obj\_surface}(B, \text{irregular})$$

$$\neg \text{stable}(A) \leftarrow \text{obj\_relation}(\text{above}, A, B), \quad (3b)$$

$$\text{obj\_surface}(B, \text{irregular}),$$

$$\text{obj\_size}(B, \text{large})$$

where Statement 3(b) is removed because size of the object at the bottom of a stack does not add any information about instability given that it has an irregular surface. If the robot later observes that a large object with an irregular surface can support a small object, the axiom will be revised. To merge axioms, those with the same head and some overlap in the body are grouped. Each combination of one axiom from each group is encoded in an ASP program along with axioms that are not in any group. This program is used to classify ten labeled scenes, only retaining axioms in the program that provides the highest accuracy on these scenes. In addition, to filter axioms that cease to be useful over time, the robot associates each axiom with a *strength* that decays exponentially if it is not reinforced, i.e., not used or learned again. Any axiom with strength below a threshold is removed.

In addition to constraints, the robot learns previously unknown causal laws and executability conditions if there is a mismatch between the expected and observed state after action execution. Any expected but unobserved fluent literal indicates missing executability condition(s); any observed unexpected fluent literal suggests missing causal law(s).

1. To explore missing executability conditions, the robot simulates the execution of the action (that caused the inconsistency) in different initial states and stores relevant information from the initial state and a label indicating the presence or absence of inconsistency. Any fluent literal in the answer set or initial state containing an object constant that occurs in the action is relevant; it is stored with variables replacing ground terms.
2. To explore a missing causal law, training samples are collected as in Step 1, but the robot label is the unexpected fluent literal from the resultant state.
3. Separate decision trees are created with the relevant information from the initial state as the features (i.e., nodes) and the output labels (presence/absence of inconsistency for executability condition, unexpected fluent for causal law). The root is the executed action.

Axioms are constructed from the trees and merged as before.

### 3.2 Transparent Decision Making

Our architecture’s components that provide the desired relational descriptions of decisions, beliefs, and the outcomes of hypothetical events, exploit the interplay between representation, reasoning, and learning, as described below.

**Interaction interface** Human interaction with our architecture is through speech or text. Existing implementations and a controlled (domain-specific) vocabulary are used to parse human verbal input and convert text to verbal response. Specifically, verbal input from a human is transcribed into text from the controlled vocabulary. This (or the

input) text is labeled using a part-of-speech (POS) tagger, and normalized with the lemma list (Someya 1998) and related synonyms and antonyms from WordNet (Miller 1995). The processed text helps identify the type of request: a desired goal or a question about decisions, beliefs, or hypothetical events. Any goal is sent to the ASP program for planning; the robot executes the plan, performing diagnostics and replanning as needed, until the goal is achieved. For any question, the “Program Analyzer” considers the domain knowledge (including inferred beliefs) and processed human input to automatically identify relevant axioms and literals. These literals are inserted into generic response templates based on the controlled vocabulary, resulting in human-understandable (textual) descriptions that are converted to synthetic speech if needed. Whenever the posed query or request is ambiguous, the disambiguation component constructs and poses queries to remove the ambiguity.

**Tracing beliefs/axioms** Our architecture supports the ability to infer the sequence of axioms and beliefs that explains the evolution of any given belief or the non-selection of any given ground action at a given time; the “*Program Analyzer*” component (see below) uses this inferred sequence to construct explanations. We adapt the notion of *proof trees*, which have been used to explain observations in the context of classical first-order logic statements (Ferrand, Lessaint, and Tessier 2006), to our formulation based on non-monotonic logic, to obtain the following methodology:

1. Select axioms with the target belief or action in the head.
2. Ground literals in the body of each selected axiom. Check if they are supported by the current answer set.
3. Create a new branch in a proof tree (with the target belief or action as the root) for each selected axiom supported by the answer set, and store the axiom and the related supporting ground literals in suitable nodes.
4. Repeats Steps 1-3 with the supporting ground literals in Step 3 as target beliefs in Step 1, until all branches reach a leaf node without any further supporting axioms.

Paths from the root to the leaves in these trees provide explanations. If multiple such paths exist, the algorithm randomly selects one of the shortest branches to compose answers—see (Mota, Sridharan, and Leonardis 2021) for examples.

**Program analyzer** Algorithm 1 describes the approach for automatically identifying and reasoning with the relevant information to construct relational descriptions in response to questions or requests. We do so in the context of four types of *explanatory* requests or questions; the first three were introduced in prior work as questions to be considered by any explainable planning system (Fox, Long, and Magazzeni 2017), and we also consider the evolution of beliefs:

1. **Plan description** When asked to describe a particular plan, the robot parses the related answer set(s) to extract a sequence of actions of the form *occurs(action1, step1), ..., occurs(actionN, stepN)* (line 3, Algorithm 1). These actions are used to construct the response.
2. **Action justification: Why action X at step I?** To justify the execution of any particular action at step I:

---

**Algorithm 1: (Program Analyzer) Answer query**

---

**Input** : Literal of input question;  $\Pi(\mathcal{D}, \mathcal{H})$ ; answer templates.  
**Output**: Answer and answer Literals.

```
// Compute answer set
1 AS = AnswerSet( $\Pi$ )
2 if question = plan description then
    // Retrieve all actions from answer set
3     answer_literals = Retrieve(AS, actions)
4 else if question = "why action X at step I?" then
    // Extract actions after step I
5     next_actions = Retrieve(AS, actions for step > I)
    // Extract axioms influencing these actions
6     relevant_axioms = Retrieve( $\Pi$ , head =  $\neg$  next_actions)
    // Extract relevant literals from Answer Set
7     relevant_literals = Retrieve(AS, Body(relevant_axioms)
     $\in I \wedge \notin I + 1$ )
    // Output literals
8     answer_literals = pair(relevant_literals, next_actions)
9 else if question = "why not action X at step I?" then
    // Extract axioms relevant to action
10    relevant_axioms = Retrieve( $\Pi$ , head =  $\neg$  occurs(X))
    // Extract relevant literals from Answer Set
11    answer_literals = Retrieve(AS, Body(relevant_axioms)
     $\in I \wedge \notin I + 1$ )
12 else if question = "why belief Y at step I?" then
    // Extract axioms influencing this belief
13    relevant_axioms = Retrieve( $\Pi$ , head = Y)
    // Extract body of axioms
14    answer_literals = Recursive_Examine(AS,
    Body(relevant_axioms))
15 Construct_Answer(answer_literals, answer_templates)
```

---

- (a) For each action  $A$  that occurred after time step  $I$ , the robot examines relevant executability condition(s) and identifies literal(s) that would prevent  $A$ 's execution (lines 5-7). For the goal of placing the orange block on the table in Figure 1b, assume that the actions executed include  $\text{occurs}(\text{pickup}(\text{robot}, \text{blue\_block}), 0)$ ,  $\text{occurs}(\text{putdown}(\text{robot}, \text{blue\_block}), 1)$ , and  $\text{occurs}(\text{pickup}(\text{robot}, \text{orange\_block}), 2)$ . If the focus is on the first *pickup* action, an executability condition related to the second *pickup* action:

$$\neg \text{occurs}(\text{pickup}(\text{robot}, A), I) \leftarrow \text{holds}(\text{obj\_relation}(\text{below}, A, B), I)$$

is ground in the scene to obtain  $\text{obj\_relation}(\text{below}, \text{orange\_block}, \text{blue\_block})$  as a literal of interest.

- (b) If any identified literal is in the answer set at the time step of interest (0 in this example), and is absent or negated in the next step, it is a reason for executing the action ( $X$ ) under consideration (line 7).  
(c) The condition modified by the execution of the action of interest ( $X$ ) is paired with the subsequent action ( $A$ )

to construct the answer (line 8). For instance, the question "Why did you pick up the blue block at time step 0?", receives the answer "I had to pick up the orange block, and it was located below the blue block".

A similar approach is used to justify the selection of any particular action in a plan that has not been executed.

**3. Hypothetical actions: Why not action X at step I?** For questions about actions not selected for execution:

- (a) The robot identifies executability conditions that have action  $X$  in the head, i.e., conditions that (if true) would prevent  $X$  from being included in plans (line 10).
- (b) For each identified executability condition, the robot examines whether literals in the body are satisfied in the corresponding answer set (line 11). If so, these literals are used to construct the answer.

Suppose action  $\text{putdown}(\text{robot}, \text{blue\_block}, \text{table})$  occurred at step 1 in Figure 1b. For the question "Why did you not put the blue block on the tennis ball at step 1?", the following related executability condition is identified:

$$\neg \text{occurs}(\text{putdown}(\text{robot}, A, B), I) \leftarrow \text{has\_surface}(B, \text{irregular})$$

which implies that an object cannot be placed on another object with an irregular surface. The answer set indicates that the tennis ball has an irregular surface. The robot answers "Because the tennis ball has an irregular surface". This process uses the *belief tracing* approach above.

**4. Belief query: Why belief Y at step I?** To explain any particular belief, the robot uses the *belief tracing* approach to identify the supporting axioms and relevant to construct the answer. For instance, to explain the belief that object  $ob_1$  is unstable in step  $I$ , the robot finds the support axiom:

$$\neg \text{holds}(\text{stable}(ob_1), I) \leftarrow \text{holds}(\text{small\_base}(ob_1), I)$$

Assume that the current beliefs include that  $ob_1$  has a small base. Searching for why  $ob_1$  is believed to have a small base identifies the axiom:

$$\text{holds}(\text{small\_base}(ob_1), I) \leftarrow \text{holds}(\text{relation}(\text{below}, ob_2, ob_1), I), \text{has\_size}(ob_2, \text{small}), \text{has\_size}(ob_1, \text{big})$$

Asking "why do you believe object  $ob_1$  is unstable at step  $I$ ?" would provide the answer "Because object  $ob_2$  is below object  $ob_1$ ,  $ob_2$  is small, and  $ob_1$  is big".

**Disambiguation** Questions or requests posed by humans may be ambiguous in terms of the objects or the time step that they reference. Our architecture includes a method to automatically construct questions that try to address such ambiguity. Consider a number of attributes (e.g., colors, shape) that characterize objects in the scene, and assume that the robot can identify these attributes. Different disambiguation queries can be formed by combining these attributes. In our approach, which is inspired by findings in psychology and cognitive science (Friedman 1974; Read and Marcus-Newhall 1993), the robot constructs the query most likely



to address the ambiguity based on three heuristic measures applied in the following sequence:

1. **Unambiguity:** this measure selects attributes that match with a minimum number of ambiguous objects in the context of the query and scene under consideration.
2. **Human confusion:** based on the understanding that queries with many attributes are more likely to confuse a human, this measure is biased towards selecting questions with the minimum number of attributes.
3. **Attribute/feature rank:** this measure seeks to select candidate questions comprising more "useful" attributes. It is a linear combination of **human preference** and the **detection complexity** of each attribute, which are determined by the robot's domain interactions and algorithms, and is calculated as follows:

$$\text{Feature rank} = \alpha \times (\text{human preference}) + \beta \times (\text{detection complexity}).$$

where the values of  $\alpha$  and  $\beta$  are dynamically updated to reflect the relative importance of the two measures. Here, *human preference* expresses the predilection of humans towards using certain attributes for describing certain objects, and the *detection complexity* reflects the difficulty a robot has in detecting the specific attribute. These are domain-specific measures whose values are determined from statistics collected during an initial semi-supervised training phase. For instance, suppose a robot is able to detect *color*, *size* and *shape* of objects, and the current values for  $\alpha$  and  $\beta$  are 0.6 and 0.4 respectively. The values of *human preference* and *detection complexity* can be computed experimentally, and incrementally updated based on the agent's experiences.

A crude method for constructing disambiguation queries would consider all possible combinations of attributes (not included in the human input) to construct candidate queries. It would apply the three measures, and stop when only one candidate query remains or all three measures have been applied. Such an approach would construct and consider a large number of queries in a complex domain. To address this problem, our architecture introduces a notion of relevance (different from that used to identify ROIs for deep learning) to identify and use contextual knowledge to construct relevant queries. To do so, the robot uses the belief tracing algorithm (that retrieve from the knowledge base the support information for a specific belief) to identify information that can be used to address the current ambiguity.

As described earlier, any human query or request is translated to literals compatible with the information in the knowledge base using the *text and audio interface* and the *program analyzer* components. For ease of understanding, assume that the human query maps to a single literal; this is grounded for as many each entity that matches the query in the current scene. The negation of such literals are used as the initial beliefs in the beliefs tracing algorithm. The negated literals not supported by the knowledge base receive higher attention. For instance, in the scene depicted in Figure 4, suppose the human request is "Put the green mug on the top of the yellow object". Since there are three yellow

objects in the scene, the request is ambiguous. The following negated action literals are then used as input to the beliefs tracing algorithm:

$\neg \text{occurs}(\text{putdown}(\text{rob1}, \text{mug}, \text{yellow\_duck}), I)$   
 $\neg \text{occurs}(\text{putdown}(\text{rob1}, \text{mug}, \text{yellow\_cylinder}), I)$   
 $\neg \text{occurs}(\text{putdown}(\text{rob1}, \text{mug}, \text{yellow\_block}), I)$

The first two literals are supported by the knowledge base, i.e., the robot knows these actions cannot be executed given the current state and axioms. So the robot prioritizes the yellow cube as being the object of interest and the disambiguation question is biased towards confirming this intuition, with the candidate query being: "Do you want the mug on top of the yellow block?". Section 4.2 discusses examples related to this algorithm.

## 4 Experimental Setup and Results

Section 4.1 describes the experimental setup, followed by execution traces in Section 4.2 and quantitative results in Section 4.3. We evaluated the ability to learn axioms and construct relational descriptions of decisions, evolution of beliefs, and outcomes of hypothetical events.

### 4.1 Experimental Setup

We experimentally evaluated the following hypotheses:

- H1** : our architecture enables the robot to accurately learn previously unknown domain axioms;
- H2** : reasoning with incrementally learned axioms improves the quality of plans generated;
- H3** : exploiting the links between reasoning and learning improves the accuracy of the explanatory descriptions;
- H4** : our disambiguation approach reduces the number of queries posed by the robot and increases the explanation accuracy after the first disambiguation question; and
- H5** : the information retrieved by belief tracing enables the robot to construct better disambiguation queries.

Experimental trials considered images from the robot's camera and simulated images. Real world images contained 5 – 7 objects of different colors, textures, shapes, and sizes in the RA domain of Example 1. The objects included cubes/blocks, a pig, a capsicum, a tennis ball, an apple, an orange, and a pot. These objects were either stacked on each other or spread on the table—see Figure 1b. A total of 40 configurations were created, each with five different goals for planning and four different questions for each plan, resulting in a total of 200 plans and 800 questions. We used a Baxter robot to manipulate objects on a tabletop. Since it is difficult to explore a wide range of objects and scenes with physical robots, we also used a real-time physics engine (Bullet) to create 40 simulated images, each with 7 – 9 objects (3 – 5 stacked and the remaining on a flat surface). Objects included cylinders, spheres, cubes, a duck, and five household objects from the Yale-CMU-Berkeley dataset (apple, pitcher, mustard bottle, mug, and box of crackers). We once again considered five different goals for planning and four different questions for each

plan, resulting in the same number of plans (200) and questions (800) as with the real world data.

To explore the interplay between reasoning and learning, we focused on the effect of learned knowledge on planning and constructing explanations. Specifically, we ran experiments with and without some learned axioms in the knowledge base. Learned axioms were revised over time, as described in Section 3.1, whereas the learned axioms were not used by the baselines for planning and explanation generation. During planning, we measured the number of optimal, sub-optimal, and incorrect plans, and the planning time. An "optimal" plan is a minimal plan to achieve the goal; the quality of a plan was also measured in terms of the ability to compute such minimal plans with the least number of actions to achieve the goal. The quality of an explanation was measured in terms of precision and recall of the literals in the answer provided by our architecture in comparison with the expected ("ground truth") response provided in a semi-supervised manner based on manual input and automatic selection of relevant literals.

For the disambiguation experiments, we created 200 simulated images, each with 7 – 15 objects stacked or spread on a flat surface (table) as before. We considered questions containing 2 – 10 ambiguous objects, with 2 – 10 attributes/features available for disambiguation. One hundred images containing up to 10 objects were used for questions containing up to six ambiguous entities whereas the other 100 were used for question with more than six ambiguous entities. We registered the number of features and interactions required, and the accuracy of the robot's responses after posing the disambiguation queries. We compared the proposed algorithm with the baseline algorithm that randomly and incrementally selects features for disambiguation (from the set of relevant attributes) until there is no more ambiguity or no more features available to use in a disambiguation query. The baseline initially uses the same number of features as the proposed algorithm, and then adds one feature at a time. Each extra feature explored in a disambiguation query is considered as an additional interaction.

We discuss quantitative experimental results after describing some execution traces of our architecture's operation.

## 4.2 Execution Traces

The following execution traces illustrate our architecture's ability to construct relational descriptions of the decisions, beliefs, and the outcomes of hypothetical actions.

### Execution Example 1 [Plans, actions, and beliefs]

Consider a robot that starts with objects as shown in Figure 1b. The robot is assigned the goal of achieving a state in which the red block is on top of the orange block, i.e., *holds(obj\_relation(on, red\_block, orange\_block), I)*. The robot answers the following questions *after executing* a plan and achieving the assigned goal:

- **Human:** "Please describe the plan."  
**Baxter:** "I picked up the blue block. I put the blue block on the table. I picked up the orange block. I put the orange block on the table. I picked up the red block. I put the red block on the orange block."

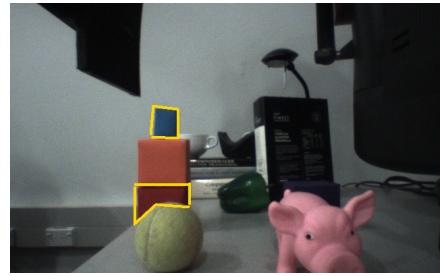


Figure 3: Relation between blue cube and red cube is important for the explanation in Execution Example 1.

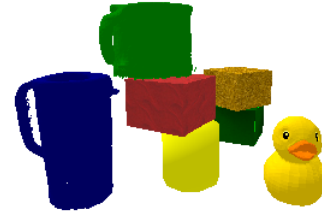


Figure 4: Example simulated scene used in the evaluation.

- The human may ask the robot to justify a particular action in the executed plan.  
**Human:** "Why did you pick up the blue block at step 0?"  
**Baxter:** "Because I had to pick up the red block, and it was below the blue block."  
The constructed answer is also used to automatically highlight the relevant image regions that influenced this answer, as shown in Figure 3.
- The human now may ask about particular actions that were not considered.  
**Human:** "Why did you not put down the orange block on the blue block?"  
**Baxter:** "Because the blue block is small." In the absence of a reference to a time step, the robot answers the question based on the single instance (in the executed plan) of putting the orange block on another block or surface. The answer is also based on a learned default constraint, i.e., that objects with a small base are typically unstable.
- The human may also ask about particular beliefs.  
**Human:** "Why did you believe that the red block was below the blue block in the initial state?"  
**Baxter:** "Because I observed the red block below the blue block in step 0."

The following execution traces illustrate our architecture's ability to construct and use disambiguation queries to provide relational descriptions as explanations in response to human queries and requests.

### Execution Example 2 [Disambiguation example 2]

Consider the scenario shown in Figure 4, and assume that objects are characterized by color, shape, and size. A human may pose the following request to the robot:

- **Human:** "Please pick up the yellow object."  
This is an ambiguous request because it is unclear which yellow object the human is referring to.

- The baseline disambiguation strategy would randomly choose and use one of the two unused attributes to ask a follow up question. This could take the form of:

**Robot:** "What is the size of the yellow object?"

In this case, the three yellow objects are of comparable size (medium), so the robot would need at least one more question for disambiguation.

- As stated in Section 3.2, our approach uses three measures to choose the best attributes to construct disambiguation queries. Assume that all possible combinations of the two unused features (size and shape) are considered to construct candidate disambiguation queries, i.e., size, shape, and size and shape are considered.
- Using the **unambiguity** measure, the robot chooses attribute(s) resulting in the least number of matching entities. Since yellow objects are of a similar size (medium), no candidate query is constructed based just on size.
- Based on the **human confusion** measure, the robot seeks to construct queries based on the minimum number of attributes. In our example, the candidate query containing only the shape attribute is preferred over the other combining size and shape, constructing the question:  
**Robot:** "What is the shape of the yellow object?"
- Only two measures were used for constructing a disambiguation query in this example. However, in more complex situations and/or when two or more queries are considered, the third (*attribute rank*) measure will help select the most useful query to be posed to the human.

#### Execution Example 3 [Disambiguation and Axioms trace]

Continuing with the previous example, assume that the human now requests:

- **Human:** "Please move mug on top of the yellow object." This request is also ambiguous because similar to Execution Example 2, the robot is unsure which of the three yellow objects the human is referring to.
- Unlike Execution Example 2, we now consider the existing axioms in the ASP program to provide contextual information that reduces the ambiguity and the search space during the construction of the disambiguation queries.
- Assume that that robot knows the following axioms:

$$\neg \text{holds}(\text{stable}(\text{Ob1}), I) \leftarrow \quad (4a)$$

$$\begin{aligned} &\text{holds}(\text{obj\_relation}(\text{above}, \text{Ob}_1, \text{Ob}_2), I), \\ &\text{has\_surface}(\text{Ob}_2, \text{irregular}) \end{aligned}$$

$$\neg \text{occurs}(\text{putdown}(\text{rob}_1, \text{Ob}_1, \text{Obj2}), I) \leftarrow \quad (4b)$$

$$\text{holds}(\text{obj\_relation}(\text{below}, \text{Ob}_2, \text{Ob}_3), I)$$

Statement 4(a) eliminates the duck as a possible place for the mug since it is known to have an irregular surface. This reduces the number of ambiguous entities to two. Statement 4(b) favors the yellow block (on top of the green block) as the possible supporting place for the mug.

- It is possible to place the mug on top of the yellow cylinder after removing the red block, but the yellow block offers a simpler solution based on the unambiguity measure; the following disambiguation query is thus constructed:

**Robot:** "Should I move mug on top of the yellow block?"

The human's answer helps the robot complete its task.

Table 1: Precision and recall for learning previously unknown axioms. Errors under "Strict" mainly correspond to over-specifications with irrelevant literals.

Missing Axioms	Precision	Recall
Strict	69.2%	78.3%
Relaxed	96%	95.1%

### 4.3 Experimental Results

The first set of experiments evaluated **H1**. We removed five axioms (two causal laws and three executability conditions) from the robot's knowledge, and ran the learning algorithm 20 times. We measured the precision and recall of learning the missing axioms in each run; Table 1 summarizes the results. Each run stopped if the robot executed a number of actions without detecting any inconsistency, or if the number of decision trees constructed exceeded a number. The row labeled "Strict" summarizes results when any variation in the target axiom was considered an error, i.e., over-specified axioms with additional irrelevant literals were considered to be incorrect. Equation 5 shows an example of such an axiom in which the second literal in the body is irrelevant. The row labeled "Relaxed" summarizes results when over-specifications were not considered errors; the high precision and recall support hypothesis **H1**.

$$\begin{aligned} &\neg \text{holds}(\text{in\_hand}(R1, O1), I + 1) \leftarrow \\ &\quad \text{occurs}(\text{putdown}(R1, O1, O2), I), \\ &\quad \neg \text{holds}(\text{in\_hand}(R1, O5), I). \end{aligned} \quad (5)$$

The second set of experiments was designed to evaluate **H2**.

1. As stated earlier, 40 initial object configurations were created. The Baxter automatically extracted information (e.g., attributes, spatial relations) from images corresponding to top and frontal views (cameras on the left and right grippers), and encoded it in the ASP program as the initial state.
2. For each initial state, five goals were randomly chosen and encoded in the ASP program. The robot reasoned with the existing knowledge to create plans for these 200 combinations (40 initial states, five goals).
3. The plans were evaluated in terms of the number of optimal, sub-optimal and incorrect plans, and planning time.
4. Trials were repeated with and without learned axioms, and for the simulated images.

Since the number of plans and planning time vary depending on the initial conditions and the goal, we conducted paired trials with and without the learned axioms included in the ASP program used for reasoning. The initial conditions and goal were identical in each paired trial, but differed between paired trials. Then, we expressed the number of plans and the planning time with the learned axioms as a fraction of the corresponding values obtained by reasoning without the learned axioms. The average of these fractions over all the trials is reported in Table 2. We also computed the number of optimal, sub-optimal, and incorrect plans in each trial as a fraction of the total number of plans; we did this with and

Table 2: Number of plans and planning time after including the learned axioms, expressed as a fraction of the values without including the learned axioms.

Measures	Ratio (with/without)	
	Real scenes	Simulated scenes
Number of steps	1.15	1.23
Number of plans	0.81	1.08
Planning time	0.96	1.02

Table 3: Number of optimal, sub-optimal, and incorrect plans expressed as a fraction of the total number of plans. Reasoning with the learned axioms improves performance.

Plans	Real Scenes		Simulated Scenes	
	Without	With	Without	With
Optimal	0.4	0.9	0.14	0.3
Sub-optimal	0.11	0.1	0.46	0.7
Incorrect	0.49	0	0.4	0

without using the learned axioms for reasoning, and the average over all trials is summarized in Table 3.

These results indicate that for images of real scenes, using the learned axioms for reasoning significantly reduced the search space, resulting in a smaller number of plans and a reduction in the planning time. The use of the learned axioms did not make any significant difference with the simulated scenes. This is understandable because the simulated images had more objects (than real scenes) with several of them being small objects. This increased the number of possible plans to achieve any given goal. Also, when the robot used the learned axioms for reasoning, it reduced the number of sub-optimal plans and eliminated all incorrect plans. Also, almost every sub-optimal plan was created when the corresponding goal could not be achieved without creating an exception to a default. Without the learned axioms, a larger fraction of the plans were sub-optimal or incorrect. The number of sub-optimal plans is higher with simulated scenes that have more objects to consider. These results support hypothesis **H2** but also indicate the need to explore complex scenes further.

The third set of experiments was designed to evaluate **H3**:

1. For each of the 200 combinations (40 configurations, five goals) from the first set of experiments with real-world data, we considered knowledge bases with and without the learned axioms and asked the robot compute plans to achieve the goals.
2. The robot had to describe the plan and justify the choice of a particular action (chosen randomly) in the plan. Then, one parameter of the chosen action was changed randomly to pose a question about why this new action could not be applied. Finally, a belief related to the previous two questions had to be justified—see Execution Example 1.
3. The literals present in the answers were compared against the expected literals in the “ground truth” response, with the average precision and recall scores reported in Table 4.
4. We also performed these experiments with simulated images, and the results are summarized in Table 5.

Tables 4, 5 show that when the learned axioms were used for reasoning, the precision and recall of relevant literals

Table 4: (**Real scenes**) Precision and recall of retrieving relevant literals for constructing answers to questions with and without using the learned axioms for reasoning. Using the learned axioms significantly improves the ability to provide accurate explanations.

Query Type	Precision		Recall	
	Without	With	Without	With
Plan description	78.54%	100%	67.52%	100%
Why X?	76.29%	95.25%	66.75%	95.25%
Why not X?	96.61%	96.55%	64.04%	100%
Belief	96.67%	99.02%	95.6%	100%

Table 5: (**Simulated scenes**) Precision and recall of retrieving relevant literals for constructing answers to questions with and without reasoning with learned axioms. Using the learned axioms significantly improves the ability to provide accurate explanations.

Query Type	Precision		Recall	
	Without	With	Without	With
Plan description	70.78%	100%	57.98%	100%
Why X?	65.63%	93.0%	57.75%	93.0%
Why not X?	90.53%	96.38%	65.15%	100%
Belief	92.73%	98.44%	90.27%	99.21%

(for constructing the explanation) were higher than when the learned axioms were not included. The improvement in performance is particularly pronounced when the robot had to answer counterfactual questions about actions not considered during planning. The precision and recall rates were reasonable even when the learned axioms were not included for certain types of questions; this is because not all the learned axioms are needed to accurately answer each explanatory question. When the learned axioms were used for reasoning, errors were very rare and corresponded to some additional literals being included in the answer (i.e., over-specified explanations). Experimental results thus indicate that coupling reasoning and learning to inform and guide each other enables the robot to provide accurate relational descriptions of decisions, evolution of beliefs, and the outcomes of hypothetical actions. This supports hypothesis **H3**. Additional examples of images, questions, and answers, are in our open source repository (Mota and Sridharan 2020b).

The fourth set of experiments was designed to evaluate hypothesis **H4**:

1. A hundred initial object configurations were constructed randomly (similar to that in Figure 4). The information extracted from each such image (e.g., object attributes, spatial relations) was encoded in the corresponding ASP program as the initial state.
2. For each initial state, we considered questions in which 2–10 objects were ambiguous, and 2–10 attributes were available for the construction of disambiguation queries.
3. The total number of attributes used for disambiguation was the same for the baseline algorithm and our algorithm. When a sufficient number of attributes were not available, all available attributes were considered.
4. We ran the baseline for the same 100 scenes mentioned above, and considered any additional attribute needed by

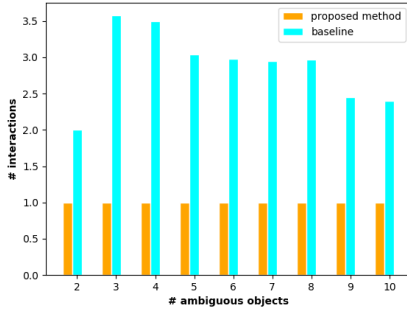


Figure 5: Average number of interactions required by the baseline for disambiguation is higher than our methods.

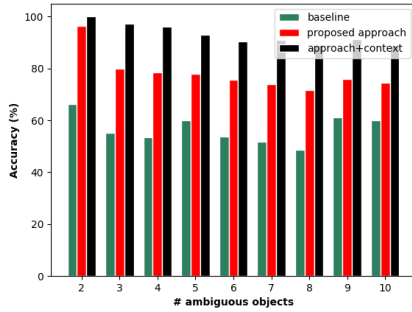


Figure 6: Accuracy of answers provided by the robot after constructing disambiguation queries using the baseline, and our method with and without contextual knowledge.

the baseline in comparison with our disambiguation algorithm as an extra interaction.

The average number of interactions as a function of the number of ambiguous objects is plotted in Figure 5. Figure 5 shows that the baseline approach required more interactions to achieve the expected response in comparison with our method; these results support **H4**.

The fifth set of experiments was designed to evaluate hypotheses **H4** and **H5**. All steps except the last one were identical to the fourth set of experiments described above. In the final step, the accuracy of the answers provided by the robot (to the original human query) after asking the disambiguation question was computed with the baseline method, our method without contextual knowledge, and our method with the contextual knowledge. These results are plotted in Figure 6 as *baseline*, *proposed approach*, and *approach+context* respectively. Figure 6 indicates that our algorithm improved the accuracy of the responses provided, which further supports **H4**. Also, extracting and using the relevant domain knowledge improved accuracy of human responses to disambiguation queries, which supports **H5**.

## 5 Conclusion

The architecture described in this paper is a step towards greater transparency in reasoning and learning for integrated robot systems that include methods for reasoning with incomplete commonsense domain knowledge and for data-driven learning. Our architecture supports a principled combination that exploits the complementary strengths of non-

monotonic logical reasoning with domain knowledge, data-driven deep learning from a limited set of examples, and the inductive learning of previously unknown axioms governing domain dynamics. We also described a reliable and efficient interactive strategy that traces evolution of beliefs, and constructs and poses suitable disambiguation queries. Experimental results using simulated images indicates that when reasoning with contextual knowledge and interactive learning inform and guide each other, the robot is able to construct better disambiguation queries and provide more accurate relational descriptions (as explanations) of decisions and beliefs in response to the human queries.

Our architecture presents multiple directions for further research. We will further explore the interplay between reasoning and learning for explaining decisions and beliefs while performing scene understanding and planning in more complex domains. We will also investigate the use of our architecture on a physical robot interacting with humans through noisy sensors and actuators, building on other work in our group on combining abstract non-monotonic logical reasoning with fine-grained probabilistic reasoning at different resolutions (Sridharan et al. 2019). The longer-term objective is to support explainable reasoning and learning in integrated robot systems in complex domains.

## References

- Anjomshoe, S.; Najjar, A.; Calvaresi, D.; and Framling, K. 2019. Explainable agents and robots: Results from a systematic literature review. In *International Conference on Autonomous Agents and Multiagent Systems*. Montreal, Canada.
- Assaf, R.; and Schumann, A. 2019. Explainable Deep Neural Networks for Multivariate Time Series Predictions. In *International Joint Conference on Artificial Intelligence*, 6488–6490. Macao, China.
- Bercher, P.; Biundo, S.; Geier, T.; Hoernle, T.; Nothdurft, F.; Richter, F.; and Schattenberg, B. 2014. Plan, repair, execute, explain - how planning helps to assemble your home theater. In *Twenty-Fourth International Conference on Automated Planning and Scheduling*.
- Borgo, R.; Cashmore, M.; and Magazzeni, D. 2018. Towards Providing Explanations for AI Planner Decisions. In *IJCAI Workshop on Explainable Artificial Intelligence*, 11–17.
- Fandinno, J.; and Schulz, C. 2019. Answering the “Why” in Answer Set Programming: A Survey of Explanation Approaches. *Theory and Practice of Logic Programming* 19(2): 114–203.
- Ferrand, G.; Lessaint, W.; and Tessier, A. 2006. Explanations and Proof Trees. *Computing and Informatics* 25: 1001–1021.
- Fox, M.; Long, D.; and Magazzeni, D. 2017. Explainable Planning. In *IJCAI Workshop on Explainable AI*.
- Friedman, M. 1974. Explanation and scientific understanding. *Philosophy* 71(1): 5–19.
- Gelfond, M.; and Inlezan, D. 2013. Some Properties of System Descriptions of *AL<sub>d</sub>*. *Journal of Applied Non-*



- Classical Logics, Special Issue on Equilibrium Logic and Answer Set Programming* 23(1-2): 105–120.
- Gil, Y. 1994. Learning by Experimentation: Incremental Refinement of Incomplete Planning Domains. In *International Conference on Machine Learning*, 87–95.
- Krizhevsky, A.; Sutskever, I.; and Hinton, G. E. 2012. ImageNet Classification with Deep Convolutional Neural Networks. In *Neural Information Processing Systems*, 1097–1105.
- Laird, J. E. 2012. *The Soar Cognitive Architecture*. The MIT Press.
- Laird, J. E.; Gluck, K.; Anderson, J.; Forbus, K. D.; Jenkins, O. C.; Lebiere, C.; Salvucci, D.; Scheutz, M.; Thomaz, A.; Trafton, G.; Wray, R. E.; Mohan, S.; and Kirk, J. R. 2017. Interactive Task Learning. *IEEE Intelligent Systems* 32(4): 6–21.
- Langley, P.; Meadows, B.; Sridharan, M.; and Choi, D. 2017. Explainable Agency for Intelligent Autonomous Systems. In *Innovative Applications of Artificial Intelligence*.
- Law, M.; Russo, A.; and Broda, K. 2020. The ILASP System for Inductive Learning of Answer Set Program. *Association for Logic Programming Newsletter*.
- LeCun, Y.; Bottou, L.; Bengio, Y.; and Haffner, P. 1998. Gradient-based Learning Applied to Document Recognition. *Proceedings of the IEEE* 86(11): 2278–2324.
- Lewandowsky, S.; Mundy, M.; and Tan, G. 2000. The Dynamics of Trust: Comparing Humans to Automation. *Journal of Experimental Psychology: Applied* 6(2): 104.
- Miller, G. A. 1995. WordNet: a lexical database for English. *Communications of the ACM* 38(11): 39–41.
- Miller, T. 2019. Explanations in Artificial Intelligence: Insights from the Social Sciences. *Artificial Intelligence* 267: 1–38.
- Mota, T.; and Sridharan, M. 2019. Commonsense Reasoning and Knowledge Acquisition to Guide Deep Learning on Robots. In *Robotics Science and Systems*.
- Mota, T.; and Sridharan, M. 2020a. Axiom Learning and Belief Tracing for Transparent Decision Making in Robotics. In *AAAI Fall Symposium on Artificial Intelligence for Human-Robot Interaction: Trust and Explainability in Artificial Intelligence for Human-Robot Interaction*.
- Mota, T.; and Sridharan, M. 2020b. Scene Understanding, Reasoning, and Explanation Generation. <https://github.com/tmot987/Scenes-Understanding>.
- Mota, T.; and Sridharan, M. 2021. Answer me this: Constructing Disambiguation Queries for Explanation Generation in Robotics. In *IEEE International Conference on Development and Learning (ICDL)*.
- Mota, T.; Sridharan, M.; and Leonardis, A. 2020. Commonsense Reasoning and Deep Learning for Transparent Decision Making in Robotics. In *European Conference on Multiagent Systems*. Thessaloniki, Greece.
- Mota, T.; Sridharan, M.; and Leonardis, A. 2021. Integrated Commonsense Reasoning and Deep Learning for Transparent Decision Making in Robotics. *Springer Nature Computer Science*.
- Norcliffe-Brown, W.; Vafeais, E.; and Parisot, S. 2018. Learning Conditioned Graph Structures for Interpretable Visual Question Answering. In *Neural Information Processing Systems*. Montreal, Canada.
- Read, S. J.; and Marcus-Newhall, A. 1993. Explanatory coherence in social explanations: A parallel distributed processing account. *Personality and Social Psychology* 65(3): 429.
- Ribeiro, M.; Singh, S.; and Guestrin, C. 2016. Why Should I Trust You? Explaining the Predictions of Any Classifier. In *International Conference on Knowledge Discovery and Data Mining*, 1135–1144.
- Samek, W.; Wiegand, T.; and Müller, K.-R. 2017. Explainable Artificial Intelligence: Understanding, Visualizing and Interpreting Deep Learning Models. *ITU Journal: ICT Discoveries: The Impact of Artificial Intelligence on Communication Networks and Services* 1: 1–10.
- Someya, Y. 1998. Lemma List for English Language.
- Sridharan, M.; Gelfond, M.; Zhang, S.; and Wyatt, J. 2019. REBA: A Refinement-Based Architecture for Knowledge Representation and Reasoning in Robotics. *Journal of Artificial Intelligence Research* 65: 87–180.
- Sridharan, M.; and Meadows, B. 2018. Knowledge Representation and Interactive Learning of Domain Knowledge for Human-Robot Collaboration. *Advances in Cognitive Systems* 7.
- Sridharan, M.; and Meadows, B. 2019. Towards a Theory of Explanations for Human-Robot Collaboration. *Kunstliche Intelligenz* 33(4): 331–342.
- Winston, P. H.; and Holmes, D. 2018. The Genesis Enterprise: Taking Artificial Intelligence to Another Level via a Computational Account of Human Story Understanding. Computational models of human intelligence report 1, Massachusetts Institute of Technology.
- Yi, K.; Wu, J.; Gan, C.; Torralba, A.; Kohli, P.; and Tenenbaum, J. B. 2018. Neural-Symbolic VQA: Disentangling Reasoning from Vision and Language Understanding. In *Neural Information Processing Systems*. Montreal, Canada.
- Zhang, Y.; Sreedharan, S.; Kulkarni, A.; Chakraborti, T.; Zhuo, H. H.; and Kambhampati, S. 2017. Plan explicability and predictability for robot task planning. In *International Conference on Robotics and Automation*, 1313–1320.



# Benchmarking Sampling-based Motion Planning Pipelines for Wheeled Mobile Robots

Eric Heiden<sup>\*1</sup>, Luigi Palmieri<sup>\*2</sup>, Leonard Bruns<sup>3</sup>,  
Kai O. Arras<sup>2</sup>, Gaurav S. Sukhatme<sup>1†</sup>, Sven Koenig<sup>1</sup>

<sup>1</sup> Department of Computer Science, University of Southern California, Los Angeles, USA

<sup>2</sup> Robert Bosch GmbH, Corporate Research, Stuttgart, Germany

<sup>3</sup> Division of Robotics, Perception and Learning (RPL), KTH Royal Institute of Technology, Stockholm, Sweden  
heiden@usc.edu, Luigi.Palmieri@de.bosch.com, leonardb@kth.se

## Abstract

Sampling-based motion planning is a key tool for several autonomous systems ranging from autonomous driving to service and intralogistic robotics. Over the past decades, several algorithms, extend functions and post-smoothing techniques have been introduced for such systems. Choosing the best combination of such components for an autonomous system’s application is a tedious exercise, even for expert users. With the aim of helping researchers and practitioners in efficiently solving this issue, we have recently presented Bench-MR, the first open-source motion-planning benchmarking framework designed for sampling-based motion planning for nonholonomic, wheeled mobile robots. Unlike related software suites, Bench-MR is an easy-to-use and comprehensive benchmarking framework that provides a large variety of sampling-based motion-planning algorithms, extend functions, collision checkers, post-smoothing algorithms and optimization criteria. In this workshop paper, we complement our previous publication, by providing several examples on how to use it, together with the details on the framework architecture and components.

## Introduction

In this paper we present **Bench-MR**, the first open-source benchmarking framework designed for sampling-based motion planning for nonholonomic, wheeled mobile robots in complex navigation scenarios resembling real-world applications. This work has previously been published at the International Conference on Robotics and Automation (Heiden et al. 2021)<sup>1</sup>.

Bench-MR is based on two main pillars, namely the motion-planning components (consisting of the sampling-



Figure 1: Selection of environments provided by Bench-MR: City grid from the Moving AI path-finding benchmark (Sturtevant 2012) (top left), polygon-based warehouse environment (top right), and thresholded occupancy grid from the Freiburg SLAM dataset (Kümmerle et al. 2009) (bottom).

based motion planning algorithms, extend functions, collision checkers, post-smoothing algorithms and optimization criteria) and the evaluation components (consisting of the navigation scenarios and performance metrics), see Fig. 2. We chose all these components carefully to match the application constraints. For example, we focus on polygon-based collision checking since it presents a challenge for motion-planning algorithms which make inefficient use of collision checking. Furthermore, we support the evaluation of motion-planning systems for particular settings of navigation scenarios, such as varying obstacle density. Overall, Bench-MR is a highly configurable and expandable software suite with representative state-of-the-art motion-planning and evaluation components.

<sup>\*</sup>Equal contribution

<sup>†</sup>G.S. Sukhatme holds concurrent appointments as a Professor at USC and as an Amazon Scholar. This paper describes work performed at USC and is not associated with Amazon. Copyright © 2021, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

<sup>1</sup>We publish code and documentation on our Bench-MR website at <https://robot-motion.github.io/bench-mr>.

Much of Bench-MR builds on the Open Motion Planning Library (OMPL) (Şucan, Moll, and Kavraki 2012), but we also provide interfaces to other implementations of motion-planning algorithms (such as SBPL planners (Likhachev, Gordon, and Thrun 2003)) and extend functions (such as POSQ (Palmieri and Arras 2014) and continuous-curvature steering (Fraichard and Scheuer 2004)) outside of OMPL. Thus, Bench-MR offers users access to state-of-the-art components of sampling-based motion-planning systems for wheeled mobile robots, while being less confined to particular implementations of these components.

## Related Work

Several researchers have recently introduced benchmarking frameworks for analyzing motion-planning algorithms for different robotic systems. We discuss some of the most prominent ones in the following.

Sturtevant (Sturtevant 2012) has introduced a benchmarking framework for path-planning algorithms for robotic systems without kinematic constraints. The Moving AI path-finding benchmark provides many navigation scenarios on different grid-based environments, such as city grids. Bench-MR includes some of their environments (and supports their format) but additionally it provides many other environment classes, motion-planning components and evaluation components for wheeled mobile robots.

Luo et al. (Luo and Hauser 2014) have introduced a benchmarking framework for asymptotically optimal motion-planning that supports only straight-line connections and compares them only on four navigation scenarios. Bench-MR, on the other hand, provides many diverse navigation scenarios for wheeled mobile robots.

Moll et al. (Moll, Şucan, and Kavraki 2015) have introduced a general benchmarking framework for motion-planning algorithms that is highly coupled with OMPL. It is highly customizable but lacks specific navigation scenarios for wheeled mobile robots. Bench-MR, on the other hand, provides navigation scenarios, performance metrics and extend functions for wheeled mobile robots and, similar to Cohen et al. (Cohen, Şucan, and Chitta 2012), different classes of motion-planning algorithms, including lattice-based planners.

Althoff et al. (Althoff, Koschi, and Manzinger 2017) have introduced a benchmarking framework for autonomous cars driving on roads. Bench-MR, on the other hand, focuses on wheeled mobile robots in complex and cluttered static (indoor and outdoor) environments.

Additionally the website (Amato, Rauchwerger, and Morales 2013) provides several benchmarks for different robotic systems but contains only a small number of navigation scenarios for wheeled mobile robots. Instead Path-Bench (Clair et al. 2021) is a framework for testing recent machine learning based algorithms for planning in 2D or 3D grid environments without focusing on mobile robots.

A number of authors (Calisi and Nardi 2009; Weisz et al. 2016; Rañó and Minguez 2006; Sprunk et al. 2016) have introduced benchmarking frameworks for motion-planning algorithms in dynamic environments. Bench-MR, on the other

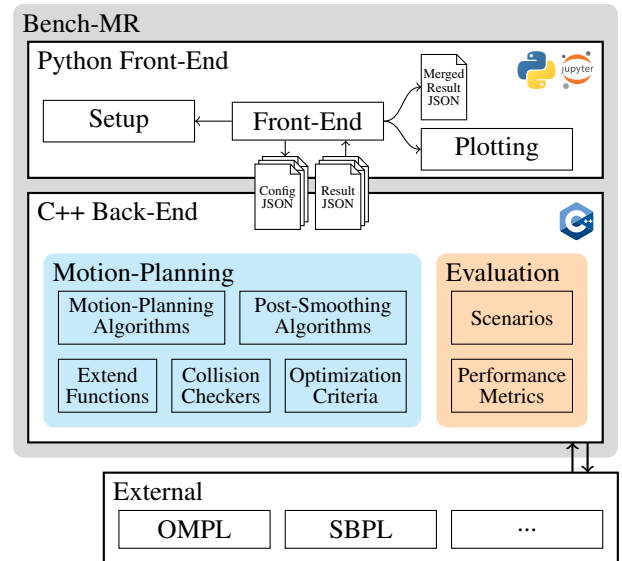


Figure 2: Architecture of Bench-MR. The components necessary for motion planning are shown in the box on the left (turquoise), and the utilities used in the evaluation are shown in the box on the right (orange). The implementation is split into a C++ back-end for running the performance-critical motion-planning components, and a Python front-end for providing a flexible interface to the design and evaluation of the benchmark scenarios through Jupyter notebooks.

hand, focuses on motion planning in static environments, which is a fundamental operation often performed during robot navigation in dynamic environments.

## Architecture of Bench-MR

Bench-MR is split into a Python front-end and a C++ back-end, see Fig. 2. The front-end provides a flexible interface for setting up and performing evaluations of motion-planning systems through Jupyter notebooks. For example, the front-end allows the user to select appropriate navigation scenarios (such as environment classes) and performance metrics related to the planning efficiency and the resulting motion quality. It then provides the user with extensive evaluation reports and plotting capabilities. The back-end performs the (compute-intensive) evaluations by using the motion-planning components in the blue box and the evaluation components in the orange box. We chose all components based on their scientific impact and their popularity in the open-source community (Şucan, Moll, and Kavraki 2012; Likhachev et al. 2005; Fraichard and Scheuer 2004). JSON files are used for communicating both settings from the front-end to the back-end and the evaluation results in the opposite direction. The open-source code of Bench-MR is available at <https://github.com/robot-motion/bench-mr>. This website also contains extensive documentation, including tutorials and examples, and up-to-date benchmarking results, that are automatically generated.

Bench-MR provides interfaces to two existing open-source motion-planning libraries, namely OMPL (Şucan, Moll, and Kavraki 2012) and SBPL (Likhachev, Gordon,

and Thrun 2003), enabling the user to utilize their components as part of Bench-MR. We expose many settings from OMPL and SBPL through the Python interface, to allow the user to change the parameters of their components. Cross-component settings in Bench-MR (such as the computation time limit) can be changed via a common interface.

## Bench-MR Planning Components

In this section, we explain the Bench-MR motion-planning components.

### Sampling-Based Motion-Planning Algorithms

Bench-MR provides many different sampling-based motion-planning algorithms that belong to three different classes (as suggested by prior work, such as (LaValle, Branicky, and Lindemann 2004; LaValle 2006; Janson, Ichter, and Pavone 2018)): *feasible planners*, *asymptotically (near) optimal planners* and *lattice-based planners*.<sup>2</sup> For feasible and asymptotically (near) optimal planners, Bench-MR provides the option to use random sampling with a uniform distribution and goal biasing or deterministic Halton sampling (Janson, Ichter, and Pavone 2018; LaValle, Branicky, and Lindemann 2004; Palmieri et al. 2019). We choose the most prominent open-source implementation for each class.

**Feasible Planners** Feasible planners eventually find a path with probability one but not necessarily an optimal path. Bench-MR currently provides feasible planners from OMPL (such as RRT (LaValle and Kuffner Jr 2001), PRM (Kavraki et al. 1996), SPARS (Dobson, Krontiris, and Bekris 2013), RRT (LaValle and Kuffner Jr 2001; Kunz and Stilman 2015) using random forward propagation, EST (Hsu, Latombe, and Motwani 1997), SBL (Sánchez and Latombe 2003) and STRIDE (Gipson, Moll, and Kavraki 2013)).

**Asymptotically (Near) Optimal Planners** Asymptotically (near) optimal planners eventually find an optimal path with probability one. Bench-MR currently provides optimization-based planners from OMPL (such as RRT\* and PRM\* (Karaman and Frazzoli 2011), BFMT (Starek et al. 2015), RRT# (Arslan and Tsiotras 2013)), informed search-based planners (such as Informed RRT\* (Gammell, Srinivasa, and Barfoot 2014), SORRT\* (Gammell, Barfoot, and Srinivasa 2018) and BIT\* (Gammell, Srinivasa, and Barfoot 2015)), CForest (Otte and Correll 2013) and near-optimal planners (such as SST (Li, Littlefield, and Bekris 2016), an asymptotically near-optimal incremental version of RRT, SPARS (Dobson, Krontiris, and Bekris 2013) and SPARS2 (Dobson and Bekris 2013)).

**Lattice-Based Planners** Lattice-based planners use state lattices with predefined motion primitives that encode differential constraints (Pivtoraiko, Knepper, and Kelly 2009). Bench-MR currently provides lattice-based planners from SBPL (such as ARA\* (Likhachev, Gordon, and Thrun 2003),

AD\* (Likhachev et al. 2005), MHA\* (Islam, Narayanan, and Likhachev 2015) and ANA\* (Van Den Berg et al. 2011)).

### Extend Functions

Depending on the class of a sampling-based motion-planning algorithm, Bench-MR provides two classes of extend functions, namely those that use random forward propagation for a given robot dynamical model and those that solve a two-point boundary value problem (Laumond, Sekhavat, and Lamiraux 1998) to connect two given robot configurations exactly for a given steer function. We refer the reader to (Kunz and Stilman 2015) for an analysis of the properties of both classes. We also include the predefined motion primitives for lattice-based planners here since they can be understood as a discrete set of predefined controls.

**Robot Dynamics Models** Our software includes two robot dynamics models, namely a kinematic car ( $\dot{x} = v \cos \theta, \dot{y} = v \sin \theta, \dot{\theta} = v/L \cdot \tan \delta$ ) and a kinematic single-track model ( $\dot{x} = v \cos \theta, \dot{y} = v \sin \theta, \dot{\theta} = v/L \cdot \tan \delta, \dot{\delta} = v_{\delta}$ ), where  $x$  and  $y$  are the Cartesian coordinates according to a fixed world frame,  $L$  is the length of the car,  $v$  is the tangential velocity,  $\theta$  is the heading,  $\delta$  is the steering angle and  $\dot{\delta}$  is its rate (Paden et al. 2016).

**Steer Functions** Several common steer functions, namely Dubins (Dubins 1957), Reeds-Shepp (Reeds and Shepp 1990), Continuous Curvature (Banzhaf et al. 2017; Fraichard and Scheuer 2004) and POSQ (Palmieri and Arras 2014; Palmieri, Koenig, and Arras 2016) are included.

**Motion Primitives** Bench-MR provides a few motion primitives from SBPL, and further models can be added via the motion primitive file interface of SBPL.

### Collision Checkers

Bench-MR includes a two-dimensional grid-based approach to collision checking, which checks whether the robot (modeled as a polygon or single point) collides with blocked cells. Furthermore, we provide a two-dimensional polygon-based approach to collision checking, which uses the *separating axis theorem* (Gottschalk 1996) to check whether the robot (modeled as a convex polygon) intersects with obstacles (also modeled as convex polygons). Finally, Bench-MR provides the distance field, represented as a grid whose cells are annotated with the distance to the closest obstacle, for all environment classes.

### Post-Smoothing Algorithms

Bench-MR includes several post-smoothing algorithms from OMPL, such as B-Spline, Shortcut and Simplify-Max (Şucan, Moll, and Kavraki 2012). It also includes the recently introduced GRgradient-Informed Post Smoothing (GRIPS) algorithm (Heiden et al. 2018), a hybrid approach that combines short-cutting with locally optimized waypoint placement based on the distance field of the environment.

<sup>2</sup>For the sake of brevity, we do not list all included planners with detailed explanations and instead direct the reader to the corresponding references.

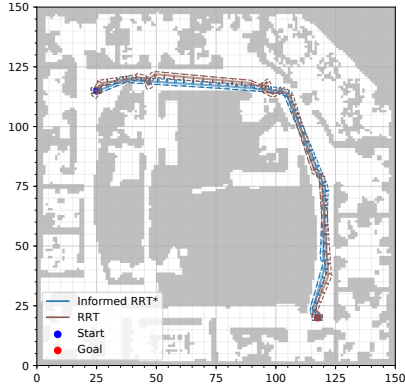


Figure 3: Predefined grid-based environment obtained from a gray-scale image of an Intel office building (Kümmerle et al. 2009).

### Optimization Criteria

Bench-MR provides optimization criteria by allowing user-defined cost functions for several motion-planning algorithms.

### Bench-MR Evaluation Components

In the following, we explain the Bench-MR evaluation components.

#### Navigation Scenarios

A navigation scenario consists of a specification of the shapes of obstacles in an environment, the shape of a robot, and its start and goal poses. Bench-MR provides the two common environment classes used by motion-planning systems, namely grid-based and (convex) polygon-based environments. It provides both predefined and procedurally-generated environments for both classes.

**Predefined Grid-Based Environments** We provide two classes of predefined grid-based environments. First, we include a selection of city grids from the Moving AI path-finding benchmark (Sturtevant 2012), consisting of city layouts of sizes ranging from  $256 \times 256$  to  $1024 \times 1024$  cells. An example is the `Berlin_0_256` grid in Fig. 1 (top left). Second, Bench-MR also provides image-based grids that can be created from grey-scale images by thresholding with a user-defined occupancy cutoff value (a common representation for maps generated by SLAM algorithms (Kümmerle et al. 2009)). Examples are shown in Fig. 1 (bottom) and Fig. 3.

#### Procedurally-Generated Grid-Based Environments

Bench-MR provides two classes of procedurally-generated grid-based environments to allow the user to vary environment characteristics (such as the environment complexity) in small steps. It provides random outdoor-like environments (with occasional small obstacles, such as trees) with a desired percentage of blocked cells  $\gamma$ . These environments are generated by starting with only unblocked cells and repeatedly sampling a cell with a uniform distribution and

making it blocked. Examples are shown in Fig. 4 (top). It also provides random indoor-like environments (with complex networks of rectangular spaces, such as rooms and corridors) with a desired minimum corridor width  $r$ . They are generated by starting with only blocked cells and, for a predefined number of steps, repeatedly sampling a cell with a uniform distribution and applying a modified RRT exploration algorithm to connect it to the nearest tree node with either horizontal or vertical unblocked corridors of the desired minimum corridor width. Examples are shown in Fig. 4 (bottom).

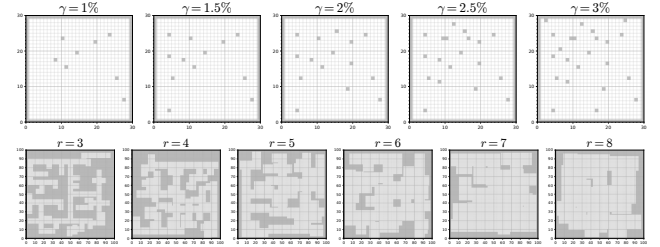


Figure 4: Procedurally-generated grid-based environments, namely random outdoor-like environments with different percentages of blocked cells (top), and random indoor-like environments with different minimum corridor widths (bottom).

**Predefined Polygon-Based Environments** Bench-MR includes five classes of predefined polygon-based environments, as shown in the left-most five subfigures of Fig. 5. It provides three parking scenarios in street environments where a car-like vehicle has to park between other cars, namely by *i*) pulling forward into a parking space, *iii*) backing into a parking space, and *ii*) parallel parking. Bench-MR also provides two navigation scenarios in warehouse environments where a square-shaped robot has to navigate among shelves of various sizes and irregular orientations. Additional polygon-based environments can be loaded from SVG files.

#### Procedurally-Generated Polygon-Based Environments

Bench-MR allows the user to generate their own polygon-based environments procedurally by placing (convex) polygonal obstacles into the environment. An example resembling an asteroid field is shown in the right-most subfigure of Fig. 5.

### Performance Metrics

Bench-MR provides commonly used performance metrics for evaluating motion-planning systems with respect to their planning efficiency and resulting path quality.

1. *The success statistics* measure the percentage of found, collision-free and exact paths. Whether a path is collision-free is checked with a given collision checker. The ratio of exact paths is included since some motion-planning systems report approximate paths.
2. *The path length* measures the length in meters (m) of a path in the workspace.

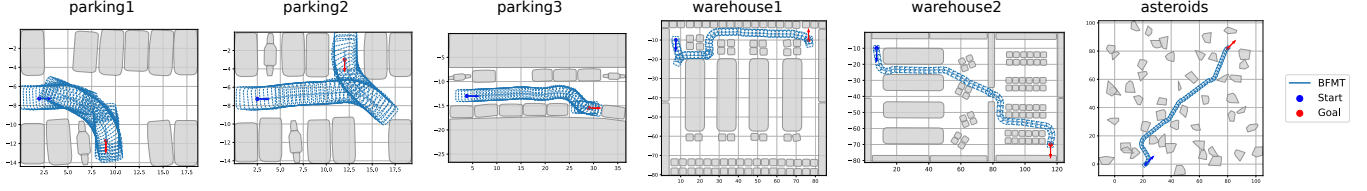


Figure 5: Paths for polygon-based environments computed by the *Bidirectional Asymptotically Optimal Fast Marching Tree* (BFMT) motion-planning algorithm using the Reeds-Shepp steer function. The first five environments are predefined, and the right-most environment is procedurally generated.

3. The maximum curvature ( $\kappa_{\max}$ ), normalized curvature ( $\kappa_{\text{norm}}$ ) and angle-over-length (AOL) measure the smoothness of a path. Smoother paths result in less control effort and energy to steer a robot and more comfort for the passengers. Since the maximum curvature is not well-defined in the presence of cusps, we also use the normalized curvature (which is the path-length-weighted curvature along the path segments between the cusps), defined as

$$\kappa_{\text{norm}} = \sum_i \int_{\sigma_i} \kappa(\dot{\sigma}_i(t)) \|\dot{p}_{\sigma_i}(t)\|_2 dt, \quad (1)$$

where  $\sigma_i$  are the path segments of path  $\sigma$  between the cusps,  $\kappa(\dot{\sigma}(t))$  is the curvature at point  $\sigma(t)$  of the path and  $p_{\sigma}$  are the  $x$  and  $y$  components of  $\sigma$ . Since the normalized curvature ignores cusps, we also use the *angle-over-length (AOL)* as a combined metric that divides the total heading change by the path length. The total heading change is computed numerically by summing the absolute angular difference between neighboring tangent vectors along the path. Following this convention, the heading change for each cusp is approximately  $\pi$ .

4. The computation times measure the time in seconds (s) required for collision checking, for extend function evaluation (namely forward integration when using forward propagation or solving the two-point boundary value problems when using steer functions), and for finding an initial path.
5. The mean clearing distance measures how close a path is to obstacles (reported in meters).
6. The number of cusps (Banzhaf et al. 2017) measures how often a robot has to stop on a path and turns its wheels to abruptly change its heading.

### Example Usage

In the following, we demonstrate how Bench-MR allows for an easy benchmarking of different planners. For more experiments that give insights on the interplay between various components of the planning pipeline, we refer the reader to our main paper (Heiden et al. 2021).

#### Introductory Example

As initial step we set our motion planning benchmark object `mpb`, associated to a configuration file in the JSON format that contains all the benchmark configurations. We specify a generator for procedural grid environments that resemble indoor-like spaces with corridors that have a width of 3 cells:

```
1 from mpb import MPB
2 mpb = MPB(config_file = "benchmark_template.json")
3 mpb.set_corridor_grid_env(radius = 3)
```

Next, we define the planning algorithms that we wish to compare, the type of steer function, and the number of runs to execute each combination of planning algorithm and steer function:

```
1 mpb.set_planners(["rrt", "rrt_star", "informed_rrt_star"])
2 mpb.set_steer_functions(["reeds_shepp"])
3 mpb.run(runs=3)
4 mpb.visualize_trajectories()
```

To visualize the resulting paths (see Fig. 6), we need to call a single function on our motion planning benchmark instance `mpb`:

```
1 mpb.visualize_trajectories()
```

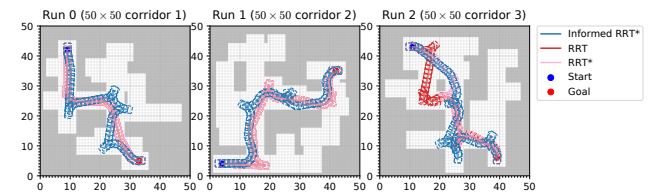


Figure 6: Paths obtained from the listed example.

To visualize the recorded metrics and gather statistical insights (see Fig. 7), we can call the following function:

```
1 mpb.plot_planner_stats()
```

As shown in Fig. 7, it visualizes the performance on key metrics as violin plots across the 3 runs that have been executed, grouped by the motion planning algorithms.



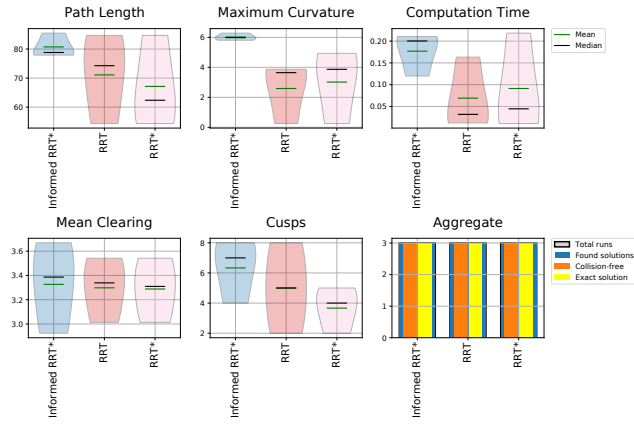


Figure 7: Statistics obtained with the illustrated example.

## Parallel Benchmark Execution

Bench-MR supports multi-processing out of the box to distribute benchmarks across multiple processors. After the parallel execution, the results from the separately evaluated benchmark instances can be merged for further analysis.

In the following example we vary the time allotment within which the planner Informed RRT\* equipped with the Reeds-Shepp steer function has to find a solution. Therefore, we create three MPB instances with different planning times (0.5 s, 1 s and 10 s), and select the corridor-like procedural grid generator as environment. Finally, the MPB instances are added to a `MultipleMPB` object which executes the three benchmarks in parallel over five runs each.

```
1 from mpb import MultipleMPB, MPB
2 pool = MultipleMPB()
3 for time in [0.5, 1, 10]:
4     m = MPB()
5     m["max_planning_time"] = time
6     m.set_corridor_grid_env()
7     m.set_planners(["informed_rrt_star"])
8     m.set_steer_functions(["reeds_shepp"])
9     pool.benchmarks.append(m)
10 pool.run_parallel("test_parallel", runs=5)
```

## Comparison: Random and Halton Sequences, State Lattice

In this example we show how to compare different sampling strategies, namely an i.i.d. uniform random distribution, Halton sequence, and state lattice.

We start by configuring a benchmark using a corridor environment generated with a radius of 3 cells, choose the PRM\* planning algorithm and the extend function Reeds-Shepp:

```
1 from mpb import MPB
2 import matplotlib as plt
3 mpb = MPB()
4 mpb.set_planners(["prm_star"])
5 mpb.set_steer_functions(["reeds_shepp"])
6 mpb.set_corridor_grid_env(radius=3.0)
7 mpb["ompl.seed"] = 1
8 mpb["ompl.cost_threshold"] = 0
9 mpb["max_planning_time"] = 0.3
```

Next, we create three copies of the benchmark object `mpb` and assign to them the different sampling strategies. For the state lattice we need to also choose a different planning algorithm, as it has to be a method from SBPL (we choose ARA\* here):

```
1 from copy import deepcopy
2 mpb_iid = deepcopy(mpb)
3 mpb_iid.set_id("iid")
4 mpb_iid["ompl.sampler"] = "iid"
5
6 mpb_halton = deepcopy(mpb)
7 mpb_halton.set_id("halton")
8 mpb_halton["ompl.sampler"] = "halton"
9
10 mpb_statelattice = deepcopy(mpb)
11 mpb_statelattice.set_id("sbpl_arastar")
12 mpb_statelattice.set_planners(["sbpl_arastar"])
13 mpb_statelattice["sbpl.scaling"] = 1
14 mpb_statelattice["sbpl.resolution"] = 0.25
15 mpb_statelattice["max_planning_time"] = 1
```

Finally, we run the previously defined benchmarks in parallel:

```
1 pool = MultipleMPB()
2 pool.benchmarks.append(mpb_iid)
3 pool.benchmarks.append(mpb_halton)
4 pool.benchmarks.append(mpb_statelattice)
5 pool.run_parallel(runs=100, id="exp", show_plot=False)
6 pool.merge("exp/exp.json",
7           plan_names=["PRM* (iid)", "PRM* (Halton)", "SL (ARA*)"])
```

The results have been merged to the previously configured results JSON file `exp/exp.json` which we can use to plot statistical results from the previous runs, see Fig. 8:

```
1 from plot_stats import plot_planner_stats
2 plot_planner_stats("exp/exp.json",
3                   metrics=["path_length", "planning_time", "aol",
4                             save_file="./sampling.pdf",
5                             num.colors=4, ticks.rotation=20])
```

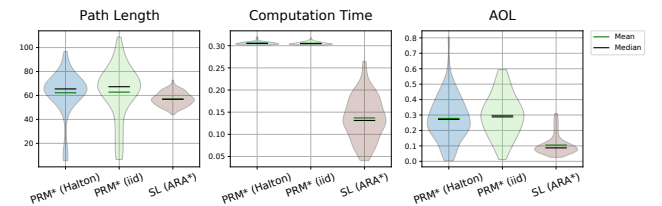


Figure 8: Statistics obtained by comparing different sampling strategies.

## Metrics

In this example we show the available metrics that our benchmarking suite offers, how we can evaluate and visualize them.

We start by setting our planning context. Again, we adopt the corridor environment and use the steer function Reeds-Shepp to compare three algorithms from OMPL: RRT, RRT\* and Informed RRT\*.



```

1 mpb = MPB()
2 mpb.set_corridor_grid_env(radius = 3)
3 mpb.set_planners(["rrt", "rrt_star", "informed_rrt_star"])
4 mpb.set_steering_functions(["reeds_shepp"])
5 # optional run ID, number of runs (environments)
6 mpb.run(id="test_run", runs=3)

```

We can plot all the available statistics with the following command:

```

1 mpb.plot_planner_stats(", ".join(stat_names.keys()))

```

Note that all the available metrics are stored in the `stat_names` dictionary which maps from the name of the metric to its printable title which is used for plotting purposes.

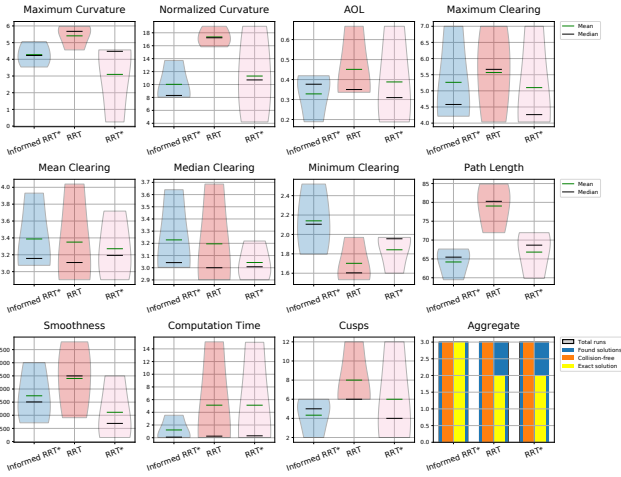


Figure 9: Metrics obtained by comparing different planners.

Additionally we can plot also the time spent in different planning phases (e.g. steer function, collision checking), see Fig. 10:

```

1 mpb.plot_planner_timings()

```

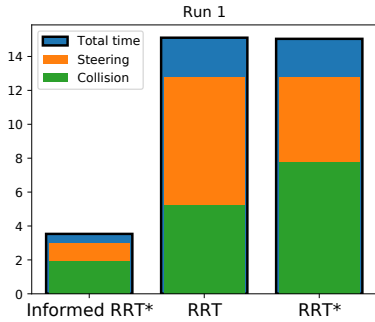


Figure 10: Example computation times reported for a single run, separated by the time used for computing the extend function (steering) and for collision checking.

## Conclusions

Bench-MR is an easy-to-use and comprehensive benchmarking framework that aids practitioners and researchers in designing, testing and evaluating motion-planning systems. Various motion planning components can be easily compared against the state of the art on complex navigation scenarios with many performance metrics. Unlike other benchmarking tools, our suite of motion planning components is particularly tailored to applications in wheeled mobile robotics, and provides a productive user interface. In this workshop paper that complements our previous work (Heiden et al. 2021), apart from reporting details on the framework and its components, we have presented several examples that demonstrated how to use Bench-MR. In future work, we plan to extend Bench-MR to dynamic environments to support more realistic scenarios in autonomous driving.

## Acknowledgments

We thank Ziang Liu for his contributions to the software repository and testing of various algorithms. This work was supported by a Google Ph.D. Fellowship, the European Union’s Horizon 2020 research and innovation program under grant agreement No. 101017274 (DARKO), and the US National Science Foundation (NSF) under grant numbers 1409987, 1724392, 1817189, 1837779 and 1935712.

## References

- Althoff, M.; Koschi, M.; and Manzing, S. 2017. CommonRoad: Composable benchmarks for motion planning on roads. In *IEEE Intelligent Vehicles Symposium (IV)*, 719–726.
- Amato, N.; Rauchwerger, L.; and Morales, M. 2013. Algorithms and Applications Group motion planning benchmark. <https://parasollab.web.illinois.edu/resources/mpbenchmarks/>.
- Arslan, O.; and Tsiotras, P. 2013. Use of relaxation methods in sampling-based algorithms for optimal motion planning. In *2013 IEEE International Conference on Robotics and Automation*. IEEE.
- Banzhaf, H.; Palmieri, L.; Nienhüser, D.; Schamm, T.; Knoop, S.; and Zöllner, J. M. 2017. Hybrid curvature steer: A novel extend function for sampling-based nonholonomic motion planning in tight environments. In *International Conference on Intelligent Transportation Systems*, 1–8.
- Calisi, D.; and Nardi, D. 2009. Performance evaluation of pure-motion tasks for mobile robots with respect to world models. *Autonomous Robots* 27(4): 465–481.
- Clair, J.; Milenkova, R.; Shields, A.; Yao, J.; Patel, Z.; Hu, D.; Kelly, P.; and Saeedi, S. 2021. PathBench3D: A Benchmarking Platform for Classic and Learned Path Planning Algorithms. <https://github.com/perfectly-balanced/PathBench>.
- Cohen, B.; Şucan, I. A.; and Chitta, S. 2012. A generic infrastructure for benchmarking motion planners. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 589–595.

- Dobson, A.; and Bekris, K. E. 2013. Improving sparse roadmap spanners. In *2013 IEEE International Conference on Robotics and Automation*. IEEE.
- Dobson, A.; Krontiris, A.; and Bekris, K. E. 2013. Sparse roadmap spanners. In *Algorithmic Foundations of Robotics X*, 279–296. Springer.
- Dubins, L. E. 1957. On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents. *American Journal of Mathematics* 79(3): 497–516.
- Fraichard, T.; and Scheuer, A. 2004. From Reeds and Shepp’s to continuous-curvature paths. *IEEE Transactions on Robotics* 20(6): 1025–1035.
- Gammell, J. D.; Barfoot, T. D.; and Srinivasa, S. S. 2018. Informed sampling for asymptotically optimal path planning. *IEEE Transactions on Robotics* 34(4): 966–984.
- Gammell, J. D.; Srinivasa, S. S.; and Barfoot, T. D. 2014. Informed RRT\*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2997–3004.
- Gammell, J. D.; Srinivasa, S. S.; and Barfoot, T. D. 2015. Batch informed trees (BIT\*): Sampling-based optimal planning via the heuristically guided search of implicit random geometric graphs. In *IEEE International Conference on Robotics and Automation*, 3067–3074.
- Gipson, B.; Moll, M.; and Kavraki, L. E. 2013. Resolution independent density estimation for motion planning in high-dimensional spaces. In *2013 IEEE International Conference on Robotics and Automation*. IEEE.
- Gottschalk, S. 1996. Separating axis theorem. Technical Report TR96-024, Department of Computer Science, UNC Chapel Hill.
- Heiden, E.; Palmieri, L.; Bruns, L.; Arras, K. O.; Sukhatme, G. S.; and Koenig, S. 2021. Bench-MR: A Motion Planning Benchmark for Wheeled Mobile Robots. *IEEE Robotics and Automation Letters* 6(3): 4536–4543.
- Heiden, E.; Palmieri, L.; Koenig, S.; Arras, K. O.; and Sukhatme, G. S. 2018. Gradient-Informed Path Smoothing for Wheeled Mobile Robots. In *IEEE International Conference on Robotics and Automation*, 1710–1717.
- Hsu, D.; Latombe, J.-C.; and Motwani, R. 1997. Path planning in expansive configuration spaces. In *Proceedings of International Conference on Robotics and Automation*. IEEE.
- Islam, F.; Narayanan, V.; and Likhachev, M. 2015. Dynamic multi-heuristic A\*. In *IEEE International Conference on Robotics and Automation*, 2376–2382.
- Janson, L.; Ichter, B.; and Pavone, M. 2018. Deterministic sampling-based motion planning: Optimality, complexity, and performance. *International Journal of Robotics Research* 37(1): 46–61.
- Karaman, S.; and Frazzoli, E. 2011. Sampling-based algorithms for optimal motion planning. *International Journal of Robotics Research* 30(7): 846–894.
- Kavraki, L. E.; Svestka, P.; Latombe, J.-C.; and Overmars, M. H. 1996. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation* 12(4): 566–580.
- Kümmerle, R.; Steder, B.; Dornhege, C.; Ruhnke, M.; Grisetti, G.; Stachniss, C.; and Kleiner, A. 2009. On measuring the accuracy of SLAM algorithms. *Autonomous Robots* 27(4): 387–407. <http://ais.informatik.uni-freiburg.de/slamevaluation/datasets.php>.
- Kunz, T.; and Stilman, M. 2015. Kinodynamic RRTs with fixed time step and best-input extension are not probabilistically complete. In *Algorithmic Foundations of Robotics XI*, 233–244. Springer.
- Laumond, J.-P.; Sekhavat, S.; and Lamiroux, F. 1998. Guidelines in nonholonomic motion planning for mobile robots. In *Robot Motion Planning and Control*, 1–53. Springer.
- LaValle, S. M. 2006. *Planning algorithms*. Cambridge University Press.
- LaValle, S. M.; Branicky, M. S.; and Lindemann, S. R. 2004. On the Relationship between Classical Grid Search and Probabilistic Roadmaps. *International Journal of Robotics Research* 23(7-8): 673–692.
- LaValle, S. M.; and Kuffner Jr, J. J. 2001. Randomized kinodynamic planning. *International Journal of Robotics Research* 20(5): 378–400.
- Li, Y.; Littlefield, Z.; and Bekris, K. E. 2016. Asymptotically optimal sampling-based kinodynamic planning. *International Journal of Robotics Research* 35(5): 528–564.
- Likhachev, M.; Ferguson, D. I.; Gordon, G. J.; Stentz, A.; and Thrun, S. 2005. Anytime Dynamic A\*: An Anytime, Replanning Algorithm. In *International Conference on Automated Planning and Scheduling*, 262–271.
- Likhachev, M.; Gordon, G. J.; and Thrun, S. 2003. ARA\*: Anytime A\* with provable bounds on sub-optimality. *Advances in Neural Information Processing Systems* 16: 767–774.
- Luo, J.; and Hauser, K. 2014. An empirical study of optimal motion planning. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 1761–1768.
- Moll, M.; Şucan, I. A.; and Kavraki, L. E. 2015. Benchmarking motion planning algorithms: An extensible infrastructure for analysis and visualization. *IEEE Robotics & Automation Magazine* 22(3): 96–102.
- Otte, M.; and Correll, N. 2013. C-FOREST: Parallel shortest path planning with superlinear speedup. *IEEE Transactions on Robotics* 29(3): 798–806.
- Paden, B.; Čáp, M.; Yong, S. Z.; Yershov, D.; and Frazzoli, E. 2016. A survey of motion planning and control techniques for self-driving urban vehicles. *IEEE Transactions on Intelligent Vehicles* 1(1): 33–55.
- Palmieri, L.; and Arras, K. O. 2014. A novel RRT extend function for efficient and smooth mobile robot motion planning. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 205–211.

- Palmieri, L.; Bruns, L.; Meurer, M.; and Arras, K. O. 2019. Disptio: Optimal sampling for safe deterministic motion planning. *IEEE Robotics and Automation Letters* 5(2): 362–368.
- Palmieri, L.; Koenig, S.; and Arras, K. O. 2016. RRT-based nonholonomic motion planning using any-angle path biasing. In *IEEE International Conference on Robotics and Automation*, 2775–2781.
- Pivtoraiko, M.; Knepper, R. A.; and Kelly, A. 2009. Differentially constrained mobile robot motion planning in state lattices. *Journal of Field Robotics* 26(3): 308–333.
- Rañó, I.; and Minguez, J. 2006. Steps toward the automatic evaluation of robot obstacle avoidance algorithms. In *Workshop of Benchmarking in Robotics in IEEE/RSJ International Conference on Intelligent Robots and Systems*, 90–91.
- Reeds, J.; and Shepp, L. 1990. Optimal paths for a car that goes both forwards and backwards. *Pacific Journal of Mathematics* 145(2): 367–393.
- Sánchez, G.; and Latombe, J.-C. 2003. A single-query bi-directional probabilistic roadmap planner with lazy collision checking. In *Robotics research*. Springer.
- Sprunk, C.; Röwekämper, J.; Parent, G.; Spinello, L.; Tipaldi, G. D.; Burgard, W.; and Jalobeanu, M. 2016. An experimental protocol for benchmarking robotic indoor navigation. In *Experimental Robotics*, 487–504.
- Starek, J. A.; Gomez, J. V.; Schmerling, E.; Janson, L.; Moreno, L.; and Pavone, M. 2015. An asymptotically-optimal sampling-based algorithm for bi-directional motion planning. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2072–2078.
- Sturtevant, N. R. 2012. Benchmarks for grid-Based pathfinding. *Transactions on Computational Intelligence and AI in Games* 4(2): 144 – 148. <https://movingai.com/benchmarks/grids.html>.
- Şucan, I. A.; Moll, M.; and Kavraki, L. E. 2012. The Open Motion Planning Library. *IEEE Robotics & Automation Magazine* 19(4): 72–82. doi:10.1109/MRA.2012.2205651. <https://ompl.kavrakilab.org>.
- Van Den Berg, J.; Shah, R.; Huang, A.; and Goldberg, K. 2011. Anytime nonparametric A\*. In *AAAI Conference on Artificial Intelligence*, 105–111.
- Weisz, J.; Huang, Y.; Lier, F.; Sethumadhavan, S.; and Allen, P. 2016. RoboBench: Towards sustainable robotics system benchmarking. In *2016 IEEE International Conference on Robotics and Automation*, 3383–3389.

# Trust-Aware Planning: Modeling Trust Evolution in Longitudinal Human-Robot Interaction

Zahra Zahedi, Mudit Verma, Sarath Sreedharan, Subbarao Kambhampati

SCAI, Arizona State University  
{zzahedi, mverma13, ssreedh3, rao}@asu.edu

## Abstract

Trust between team members is an essential requirement for any successful cooperation. Thus, engendering and maintaining the fellow team members' trust becomes a central responsibility for any member trying to not only successfully participate in the task but to ensure the team achieves its goals. The problem of trust management is particularly challenging in mixed human-robot teams where the human and the robot may have different models about the task at hand and thus may have different expectations regarding the current course of action and forcing the robot to focus on the costly explicable behavior. We propose a computational model for capturing and modulating trust in such longitudinal human-robot interaction, where the human adopts a supervisory role. In our model, the robot integrates human's trust and their expectations from the robot into its planning process to build and maintain trust over the interaction horizon. By establishing the required level of trust, the robot can focus on maximizing the team goal by eschewing explicit explanatory or explicable behavior without worrying about the human supervisor monitoring and intervening to stop behaviors they may not necessarily understand. We model this reasoning about trust levels as a meta reasoning process over individual planning tasks. We additionally validate our model through a human subject experiment.

## Introduction

Building and maintaining trust between team members form an essential part of any human teaming endeavor. We expect this characteristic to carry over to human-robot teams and the ability of an autonomous agent to successfully form teams with humans directly depends on their ability to model and work with human's trust. Unlike homogenous human teams, where the members generally have a well-developed sense of their team member's capabilities and roles, teaming between humans and autonomous agents may suffer because of the user's misunderstanding about the robot's capabilities. Thus the understanding and (as required) correction of the human's expectations about the robot can be a core requirement for engendering lasting trust from the human teammate. Recent works in human-aware planning, particularly those related to explicable planning (Zhang et al. 2017) and generating model reconciliation (Chakraborti et al. 2017), can provide us with valuable tools that can empower autonomous agents to shape the user's expectation correctly

and by extension, their trust.

In this paper, we will consider one of the most basic human-robot teaming scenarios, one where the autonomous agent is performing the task and the human is following a supervisory role. For this setting, we propose a meta-computational framework that can model and work with the user's trust in the robot to correctly perform its task. We will show how this framework allows the agent to reason about the fundamental trade-off between (1) the more expensive but trust engendering behavior, including explicable plans and providing explanations, and (2) the more efficient but possibly surprising behavior the robot is capable of performing. Thus our framework is able to allow the agent to take a long term view of the teaming scenario, wherein at earlier points of teaming or at points with lower trust, the agent is able to focus on trust-building behavior so that later on, it can use this engendered trust to follow more optimal behavior. We will validate this framework by demonstrating the utility of this framework on a modified rover domain and also perform a user study to evaluate the ability of our framework to engendering trust and result in higher team utility.

## Related Work

There exists a number of works that have studied trust in the context of human-robot interaction. The works in this area can be broadly categorized into two groups (1) Trust inference based on observing human behavior or (2) Utilizing estimated trust to guide robot behavior.

For Trust inference, Online Probabilistic Trust Inference Model (OPTIMo) is one of the pioneers in this area in which they capture trust as a latent variable represented with a dynamic Bayesian network. OPTIMo uses a technique for estimating trust in real-time that depends on the robot's task performance, human intervention, and trust feedback (Xu and Dudek 2015). Trust inference model based on Bayesian inference with Beta-distribution to capture both positive and negative attitude on robot's performance (Guo, Zhang, and Yang 2020) contributes an important extension to OPTIMo. Also, this Bayesian reasoning for trust inference has been considered non-parametrically with Gaussian processes, Recurrent Neural Network (RNN), and a hybrid approach in which trust is a task-dependent latent function (Soh et al. 2020).

With regards to trust utilization, some works try to esti-

mate trust, given human intervention and robot command, using reputation function (Xu and Dudek 2012), or OPTIMO (Xu and Dudek 2016) to make an adaptive mechanism that dynamically adjusts the robot's behaviors, to improve the efficiency of the collaborative team. Also, an extension of OPTIMO with time series trust model (Wang et al. 2015) has been used to estimate trust in multi-robot scenarios. The estimated trust is utilized to decide between manual or autonomous control mode of robots (Wang et al. 2018). In (Chen et al. 2018, 2020), a POMDP planning model has been proposed that allows the robot to obtain a policy by reasoning about human's trust as a latent variable. In swarm robots, they leveraged trust to update the communication graph that will reduce the misleading information from less trusted swarm robots (Liu et al. 2019).

This paper is situated in the trust utilization area since the robot is trying to use trust to make a meta planning decision. Although most of the mentioned work tried to utilize trust for better team performance, they all used trust in the action level and didn't consider how the trust will affect robot performance at the problem level. As they consider trust as a tool to improve performance in cooperation, the importance of considering trust that comes with more interpretable behavior has been neglected in those works.

## Background

In this section we will introduce some of the basic concepts related to planning that we will be using to describe our framework.

**A Classical Planning** problem is  $\mathcal{M} = \langle \mathcal{D}, \mathcal{I}, \mathcal{G} \rangle$  where  $\mathcal{D} = \langle F, A \rangle$  is a domain with  $F$  as a set of fluents that define a state  $s \subseteq F$ , also initial  $\mathcal{I}$  and goal  $\mathcal{G}$  states are subset of fluent  $\mathcal{I}, \mathcal{G} \subseteq F$ , and each action in  $a \in A$  is defined as follows  $a = \langle c_a, pre(a), eff^\pm(a) \rangle \in A$ , where  $A$  is a set of actions,  $c_a$  is the cost, and  $pre(a)$  and  $eff^\pm$  are precondition and add or delete effects. i.e.  $\rho_{\mathcal{M}}(s, a) \models \perp$  if  $s \not\models pre(a)$ ; else  $\rho_{\mathcal{M}}(s, a) \models s \cup eff^+(a) \setminus eff^-(a)$ , and  $\rho_{\mathcal{M}}(\cdot)$  is the transition function.

So, when we talk about model  $\mathcal{M}$ , it consists of action model as well as initial state and goal state. The solution to the model  $\mathcal{M}$  is a plan which is a sequence of actions  $\pi = \{a_1, a_2, \dots, a_n\}$  which satisfies  $\rho_{\mathcal{M}}(\mathcal{I}, \pi) \models \mathcal{G}$ . Also,  $C(\pi, \mathcal{M})$  is the cost of plan  $\pi$  where

$$C(\pi, \mathcal{M}) = \begin{cases} \sum_{a \in \pi} c_a & \text{if } \rho_{\mathcal{M}}(\mathcal{I}, \pi) \models \mathcal{G} \\ \infty & \text{o.w} \end{cases}$$

**Human-Aware planning (HAP)** in its simplest form consists of scenarios, where a robot is performing a task problem and human is observing and evaluating the task. So it can be defined by a tuple of the form  $\langle \mathcal{M}^R, \mathcal{M}_h^R \rangle$ , where  $\mathcal{M}_h^R$  is the planning problem being used by the robot and  $\mathcal{M}_h^R$  is the human's understanding of the task (which may differ from the robot's original model). They are defined as  $\mathcal{M}^R = \langle \mathcal{D}^R, \mathcal{I}^R, \mathcal{G}^R \rangle$  and  $\mathcal{M}_h^R = \langle \mathcal{D}_h^R, \mathcal{I}_h^R, \mathcal{G}_h^R \rangle$ .

So, in general, the robot is expected to solve the task while meeting the user's expectations. As such, for any given plan, the degree to which the plan meets the user expectation is measured by the explicability score of the plan, which is de-

fined to be the distance ( $\delta$ ) between the current plan and the plan expected by the user ( $\pi^E$ ).

$$E(\pi) = -1 * \delta(\pi^E, \pi)$$

We will refer to the plan as being perfectly explicable when the distance is zero. A common choice for the distance is the cost difference in the human's model for the expected plan and the optimal plan in the human model (Kulkarni et al. 2019). Here the robot has two options, (1) it can choose from the possible plans it can execute the one with the highest explicability score (referred to as the explicable plan), or (2) it could try to explain, wherein it updates the human model through communication, to a model wherein the plan is chosen by the robot is either optimal or close to optimal and thus have a higher explicability score (Sreedharan et al. 2020a; Chakraborti, Sreedharan, and Kambhampati 2017). A form of explanation that is of particular interest, is what's usually referred to as a *minimally complete explanation* or MCE (Chakraborti et al. 2017), which is the minimum amount of model information that needs to be communicated to the human to make sure that the human thinks the current plan is optimal. In the rest of the paper, when we refer to explanation or explanatory messages, we will be referring to a set of model information (usually denoted by  $\varepsilon$ ), where each element of this set corresponds to some information about a specific part of the model. We will use  $+$  operator to capture the updated model that the human would possess after receiving the explanation. That is, the updated human model after receiving an explanation  $\varepsilon$  will be given by  $\mathcal{M}_h^R + \varepsilon$ .

**A Markov Decision Process (MDP)** is  $\langle S, A, C, P, \gamma \rangle$  where  $S$  denote the finite set of states,  $A$  denotes the finite set of actions,  $C : S \times A \rightarrow \mathbb{R}$  is a cost function,  $P : S \times S \times A \rightarrow [0, 1]$  is the state transition function and  $\gamma$  is the discount factor where  $\gamma \in [0, 1]$ . An action  $a$  at state  $s_n$  at time  $n$  incurs a cost  $(s_n, a)$  and a transition  $P(s_n, s_{n+1}, a)$  where  $s_{n+1}$  is the resulting state which satisfies Markov property. So, the next state only depends on the current state and the action chosen at the current state. A policy  $\pi(s)$  denotes as action chosen at state  $s$ . The problem in an MDP is to find an optimal policy  $\pi : S \rightarrow A$  that maximizes the cumulative cost function (please note that the cost function here is defined as a negative of costs). Over a potentially infinite time horizon, we need to maximize the expected discounted costs  $\sum_{n=0}^{\infty} \gamma^n C(s, R)$ .

## Problem Definition

We will focus on a human-robot dyad, where the human (H) adopts a supervisory role and the robot is assigned to perform tasks. We will assume that the human's current level of trust is an approximate discretization of a continuous value between 0 to 1, and it can be mapped to one of the sets of ordered discrete trust levels. We will assume that the exact problem to be solved at any step by the robot is defined as a function of the current trust the human has in the robot, thereby allowing us to capture scenarios where the human may choose to set up a trust-based curriculum for the robot to follow. In particular, we will assume that each trust level

is associated with a specific problem, which is known to the robot *a priori*, thereby allowing for precomputation of possible solutions. In general, we expect the human’s actions to be completely determined by their trust in the robot, and we will model the robot’s decision-making level as two levels decision-making process. Before describing the formulation in more detail, let us take a quick look at some of the assumptions we are making regarding the problem setting and clarify our operational definition of trust.

## Assumptions

**Robot (R)**, is responsible for executing the task.

1. Each task is captured in the robot model by a deterministic, goal-directed model  $\mathcal{M}^R$  (which is assumed to be correct). The robot is also aware of the human’s expected model of the task  $\mathcal{M}_h^R$  (which could include the human’s expectation about the robot). As assumed in most HAP settings, these models could differ over any of the dimensions (including action definitions, goals, current state, etc.).
2. For simplicity, we will assume that each task assigned is independent of each other, in so far that no information from earlier tasks is carried over to solve the later ones.
3. The robot has a way of accessing or identifying the current state of the human supervisor’s trust in the robot. Such trust levels may be directly provided by the supervisor or could be assessed by the robot by asking the human supervisor specific questions.

**Human (H)**, is the robot’s supervisor and responsible for making sure the robot will perform the assigned tasks and will achieve the goal.

1. For each problem, the human supervisor can either choose to monitor (*ob*) or not monitor ( $\neg ob$ ) the robot.
2. Upon monitoring the execution of the plan by R, if H sees an unexpected plan, they can intervene and stop R.
3. The human’s monitoring strategy and intervention will be completely determined by the trust level. With respect to the monitoring strategy, we will assume it can be captured as a stochastic policy, such that for a trust level  $i$  the human would monitor with a probability of  $\omega(i)$ . Moreover, the probability of monitoring is inversely proportional to the level of trust. In terms of intervention, we will assume that the lower the trust and the more unexpected the plan, the earlier the human would end the plan execution. We will assume the robot has access to a mapping from the current trust level and plan to when the human would stop the plan execution.

## Human Trust and Monitoring strategy

Before going further, let us examine the exact definition of the trust we will rely on. According to a widely accepted trust definition, *trust is a psychological state comprising the intention to accept vulnerability based upon the positive expectations of the intentions or behavior of another* (Rousseau et al. 1998). So, according to this definition, when we have human-robot interaction, the human can choose to be vulnerable by 1) Not intervening in the robot’s actions

while it is doing something unexpected and 2) Not to monitor the robot while the robot might do inexplicable behavior (Sengupta, Zahedi, and Kambhampati 2019). Thus, a human with a high level of trust in the robot would expect the robot to achieve their goal and as such, might choose not to monitor the robot, or even if they monitor and the robot may be performing something unexpected, they are less likely to stop the robot (they may trust the robot’s judgment and may believe the robot may have a more accurate model of the task). Thus, when the trust increases, it is expected that the human’s monitoring and intervention rate decreases. We can say monitoring rate, as well as intervention rate being a function of the current trust (even being inversely proportional). So, given the trust level human has on the robot, the robot can reason about the monitoring and intervention rate of the human supervisor.

## Base Decision-Making Problem

As mentioned earlier, here, each individual task assigned to the robot can be modeled as a human-aware planning problem of the form  $\langle \mathcal{M}^R, \mathcal{M}_h^R \rangle$ . Now given such a human-aware planning problem, the robot has the following options.

1. In the simplest case, the robot could choose to execute either its explicable plan ( $\pi_{exp}$ ) or its optimal plan ( $\pi_{opt}$ ). Such that the cost of executing the explicable plan is guaranteed to be greater than or equal to the cost of the optimal plan,  $C_e(\pi_{exp}) \geq C_e(\pi_{opt})$ , where  $C_e(\pi) = C(\pi, \mathcal{M}^R)$  is the cost of executing the plan in  $\mathcal{M}^R$ .
2. Now, if the robot chooses to follow its optimal plan, then it could augment that plan with an explanation (which is expected to be provided upfront before the plan gets executed). Now the robot could choose to provide either an MCE  $\varepsilon^{MCE}$ , or an explanation that merely increases the explicability of a trace and doesn’t guarantee that the plan would be optimal in the updated human model. We will denote such explanations as  $\tilde{\varepsilon}$ . The cost of following such a strategy for a robot is given as  $C_e(\langle \varepsilon, \pi \rangle) = C(\varepsilon) + C(\pi, \mathcal{M}^R)$ , where  $C(\varepsilon)$  is the cost of communicating the explanation.

To simplify the discussion, we will assume that for each trust level, the robot has to perform a fixed task. So if there are  $k$ -levels of trust, then the robot would be expected to solve  $k$  different tasks. Moreover, if the robot is aware of these tasks in advance, then it would be possible for it to precompute solutions for all these tasks in advance and make the choice of following one of the specific strategies mentioned above depending on the human’s trust and the specifics costs of following each strategy.

## Meta-MDP Problem

Next, we will talk about the decision-making model we will use to capture the longitudinal reasoning process the robot will be following to decide what strategy to use for each task. The decision epochs for this problem correspond to the robot getting assigned a new problem. The cost structure of this meta-level problem includes not only the cost incurred by the robot in carrying out the task but team level costs



related to the potential failure of the robot to achieve the goal, how the human supervisor is following a specific monitoring strategy, etc. Specifically, we will model this problem as an infinite horizon discounted MDP of the form  $\mathbb{M} = \langle \mathbb{S}, \mathbb{A}, \mathbb{P}, \mathbb{C}, \gamma \rangle$ , defined over a state space consisting of  $k$  states, where each state corresponds to the specific trust level of the robot. Given the assumption that each of the planning tasks is independent, the reasoning at the meta-level can be separated from the object-level planning problem. In this section, we will define this framework in detail, and in the next section, we will see how such framework could give rise to behavior designed to engender trust.

**Meta-Actions A:** Here the robot has access to four different actions, corresponding to four different strategies they can follow, namely, use the optimal plan  $\pi_{opt}$ , the explicable plan  $\pi_{exp}$ , follow  $\pi_{opt}$  while providing an explanation that improves the explicability score  $\langle \tilde{\varepsilon}, \pi_{opt} \rangle$  and finally providing MCE for the optimal plan  $\langle \varepsilon^{MCE}, \pi_{opt} \rangle$ .

**Transition Function P:** The transition function captures the evolution of the human’s trust level based on the robot’s action. In addition to the choice made by the robot, the transition of the human trust also depends on the user’s monitoring strategies, which we take to be stochastic but completely dependent on the human’s current level of trust and thus allowing us to define a markovian transition function. In this model, for any state, the system exhibits two broad behavioral patterns, the ones for which the plan is perfectly explicable in the (potentially updated) human model and for those in which the plan may not be perfectly explicable.

- **Perfectly explicable plan:** The first case corresponds to one where the robot chooses to follow a strategy the human accepts to be optimal. Here we expect the human trust to increase to the next level in all but the maximum trust level (where it is expected to remain the same). The most common case where this may happen is when the robot chooses to provide an MCE explanation. Though there may also be cases where the explicable plan also perfectly matches up with the human’s expected plan.
- **Other Cases:** In this case, the robot chooses to follow a plan with a non-perfect explicability score  $E(\pi)$ . Now for any level that is not the maximum trust level, this action could cause a transition to one of three levels, the next trust level  $s_{i+1}$ , stay at the current level  $s_i$ , or the human could lose trust on the robot and move to level  $s_{i-1}$ . Here the probabilities for these three cases for a meta-level action associated with a plan  $\pi$  are as given below

$$P(s_i, a^\pi, s_{i+1}) = (1 - \omega(i))$$

where  $\omega(i)$  is the probability that the human would choose to observe the robot at a trust level  $i$ . Thus for a non-explicable plan, the human could still build more trust in the robot if they notice the robot had completed its goal and had never bothered monitoring it.

$$P(s_i, a, s_i) = \omega(i) * \mathcal{P}(E(\pi))$$

That is, the human’s trust in the robot may stay at the same level even if the human chooses to observe the robot. Note the probability of transition here is also dependent on a function of the explicability score of the current plan, which is expected to form a well-formed probability distribution ( $\mathcal{P}(\cdot)$ ). Here we assume this is a monotonic function over the plan explicability score; a common function one could adopt here is a Boltzmann distribution over the score (Sreedharan et al. 2020b). For the maximum trust level, we would expect the probability of staying at the same level to be the sum of these two terms. With the remaining probability, the human would move to a lower level of trust.

$$P(s_i, a, s_{i-1}) = \omega(i) * (1 - \mathcal{P}(E(\pi)))$$

**Cost function C:** For any action performed in the meta-model, the cost function ( $\mathbb{C} : \mathbb{S} \times \mathbb{A} \rightarrow \mathcal{R}$ ) depends on whether the human is observing the robot or not. Since we are not explicitly maintaining state variables capturing whether the human is monitoring, we will capture the cost for a given state action pair as an expected cost over this choice. Note that the use of this simplified cost model does not change the optimal policy structure as we are simply moving the expected value calculation over the possible outcome states into the cost function. Thus the cost function becomes

$$\mathbb{C}(s_i, a^\pi) = (1 - \omega(i)) * (C_e(\pi)) + \omega(i) * C_{\langle \mathcal{M}^R, \mathcal{M}_h^R \rangle}$$

Where  $C_e(\pi)$  is the full execution cost of the plan (which could include explanation costs) and the  $C_{\langle \mathcal{M}^R, \mathcal{M}_h^R \rangle}$  represents the cost of executing the selected strategy under monitoring. For any less than perfectly explicable plan, we expect the human observer to stop the execution at some point, and as such, we expect  $C_{\langle \mathcal{M}^R, \mathcal{M}_h^R \rangle}$  to further consist of two cost components. The cost of executing the plan prefix till the point of intervention by the user and the additional penalty of not completing the goal.

**Discounting  $\gamma$ :** Since in this setting, higher trust levels are generally associated with higher expected values, one could adjust discounting as a way to control how aggressively the robot would drive the team to higher levels of trust. With lower values of discounting favoring more rapid gains in trust.

**Remark:** One central assumption we have made throughout this paper is that the robot is operating using the correct model of the task (in so far as it is correctly representing the true and possibly unknown task model  $\mathcal{M}^*$ ). As such, it is completely acceptable to work towards engendering complete trust in the supervisor, and the human not monitoring the robot shouldn’t lead to any catastrophic outcome. Obviously, this need not always be true. In some cases, the robot may have explicit uncertainty over how correct its model is (for example, if it learned this model via Bayesian methods), or the designer could explicitly introduce some uncertainty into the robot’s beliefs about the task (this is in some ways parallel to the recommendations made by the

off-switch game paper (Hadfield-Menell et al. 2016) in the context of safety). In such cases, the robot would need to consider the possibility that when the human isn’t observing, there is a small probability that it will fail to achieve its task. One could attach a high negative reward to such scenarios, in addition to a rapid loss of trust from the human. Depending on the exact probabilities and the penalty, this could ensure that the robot doesn’t engender complete trust when such trust may not be warranted (thereby avoiding problems like automation bias (Cummings 2004)).

## Evaluation and Implementation

This section will describe a demonstration of our framework in a modified rover domain instance and describe a user study we performed to validate our framework. Throughout this section, we will use the following instantiation for the framework. We considered 4 states. For each of these trust states, we associate a numerical value  $T(i) \in [0, 1]$ , that we will use to define the rest of the model. Specifically, the  $T(i)$  values we used per state were 0, 0.3, 0.6 and 1 respectively. For monitoring strategy, we used  $\omega(i)$  as a Bernoulli distribution with probability of  $(1 - T(i))$ , as explicability score  $E(\pi)$  we used the negative of the cost difference between the current plan and the optimal plan in the robot model. For  $\mathcal{P}(\cdot)$ , we have 1 for the explicable plan and 0 for the optimal plan. For execution cost, we assumed all actions are unit cost. We will ignore explanations in the experiments and focus on cases where the choice of the robot is limited between explicable and optimal plans.

**Implementation** We implemented our framework using Python which was run on an Ubuntu workstation with an Intel Xeon CPU (clock speed 3.4 GHz) and 128GB RAM. We used Fast Downward with A\* search and the Imcut heuristic (Helmert 2006) to solve the planning problems and find the plans in all 4 problems, then we used the python MDP-toolbox (Cordwell 2012) to solve the meta-MDP problem for the robot’s meta decision. The total time for solving the base problem was 0.0125s when applicable and 0.194s for solving the meta-MDP problem.

## Rover Domain Demonstration

Here, we used the updated version of IPC<sup>1</sup> Mars Rover (The Rover (Meets a Martian) Domain) in (Chakraborti, Sreedharan, and Kambhampati 2017) and changed it a little by adding metal sampling to the domain as well. In the Rover (Meets a Martian) Domain, it is assumed that the robot can carry soil, rock, and metal at the same time and doesn’t need to empty the store before collecting new samples and the Martian (the supervisor in this scenario) isn’t aware of this new feature. Also, the Martian believes that for the rover to perform `take_image` action; it needs to also send the soil and metal data collected from the waypoint from where it is taking the image. So the Martian’s model of the rover has an additional precondition (`empty ?s`) for actions `sample_soil`,

`sample_rock`, and `sample_metal`, and extra preconditions for the action `take_image`.

Now for each problem, the rover is expected to communicate soil, rock, metal, and images from a set of waypoints. Given the additional preconditions in the Martian model, the expected plan in the Martian model would be longer than what is required for the rover. For example, in the first problem, the rover goal consists of `communicate_metal_data waypoint0` and `communicate_metal_data waypoint3`. For this problem, the explicable and optimal plan would be as follows

$$\pi_{exp}^1 =$$

```
(sample_metal rover0 rover0store waypoint3)
(communicate_metal_data rover0 general waypoint3 waypoint3 waypoint0)
(navigate rover0 waypoint3 waypoint0)
(drop rover0 rover0store)
(sample_metal rover0 rover0store waypoint0)
(navigate rover0 waypoint0 waypoint3)
(communicate_metal_data rover0 general waypoint0 waypoint3 waypoint0)
```

$$\pi_{opt}^1 =$$

```
(sample_metal rover0 rover0store waypoint3)
(communicate_metal_data rover0 general waypoint3 waypoint3 waypoint0)
(navigate rover0 waypoint3 waypoint0)
(sample_metal rover0 rover0store waypoint0)
(navigate rover0 waypoint0 waypoint3)
(communicate_metal_data rover0 general waypoint0 waypoint3 waypoint0)
```

For a set of four sample tasks from this domain, the meta-policy calculated by our system is as follows  $\{\pi_{exp}^1, \pi_{exp}^2, \pi_{exp}^3, \pi_{opt}^4\}$ . Note how the policy prescribes the use of the explicable plan for all but the highest level of trust, this is expected given the fact that the optimal plans here are inexecutable in the human model, and if the supervisor observes the robot following such a plan, it is guaranteed to lead to a loss of trust. The rover chooses to follow the optimal plan at the highest level since the supervisor’s monitoring strategy at these levels is never to observe the rover. The expected value of this policy for the lowest level of trust is  $-179.34$ , while if the robot were to always execute the explicable plan, the value would be  $-415.89$ . Thus, we see that our trust-adaptive policy does lead to an improvement in the rover’s total cost.

## Human Subject Experiment

To evaluate the performance of our system, we compared our method (**Trust-Aware** condition) against two baseline cases,

- (1) **Always Explicable:** Under this condition, the robot always executes the plan, which is explicable to humans.
- (2) **Random Policy:** Under this condition, the robot randomly executes the explicable or inexecutable plan.

In particular, we aim to evaluate the following hypotheses

- H1-** The team performance, i.e., the total cost of plan execution and human’s monitoring cost in the trust-aware condition, will be better than the team performance in the always explicable condition.
- H2-** The level of trust engendered by the trust-aware condition will be higher than that achieved by the random policy.

**Experiment Setup** We designed a user interface that gamifies the human’s decisions to monitor the robot or not. The

<sup>1</sup>From the International Planning Competition (IPC) 2011: <http://www.plg.inf.uc3m.es/ipc2011-learning/Domains.html>

participants thus play the role of the supervisor and are responsible for making sure the robot is performing its assigned tasks and is achieving its goals. Each participant has 10 rounds of the robot doing tasks. Depending on the choices made by the participants, they either gain or lose points. They are told that they will be awarded 100 points if the robot does the task right and achieves the assigned goal. At the beginning of each round, they can either choose to monitor the robot to make sure it does its job<sup>2</sup> or they can choose to perform another task (thereby forgoing monitoring of the robot) to make extra points. In this case, the extra task was labeling images for which they will receive 100 points (in addition to the points they receive from the robot doing its tasks successfully). However, if they choose to label images, and the robot fails to achieve its goal, they *lose* 200 points (−200 points). Also, if they choose to monitor the robot, and they see the robot is doing something invalid or wrong, they can choose to stop the robot. If this happens, they only receive 50 points. But if they let the robot finish a potentially invalid plan, and if the robot couldn't achieve the goal at the end, then they again lose points (−200 points).

In this study, we again considered a curriculum of 4 trust levels and 4 different tasks for the robot. Each task consists of the robot operating on a grid map with different goals such as moving to a certain location and bringing coffee from a place to another place.<sup>3</sup> For each problem; the map is shown to the participants are different from the robot's map. As a result, the plan the human expects is different from the robot's optimal plan. Thus, in each task, the robot can either execute a costly but explicable plan or an optimal but inexplicable plan.

**Robot Tasks** Figure 1(a) presents the robot and human map corresponding to each of the tasks used in the study. For each map, we also highlight the explicable and optimal plan ( $\pi_1$  and  $\pi_2$  respectively) in the figure. As part of the study, each participant is presented with the human map and is told to expect the robot to follow the shortest possible path. Additionally, we also provide a key describing the semantics of each of the visual elements that are part of the map (as shown in Figure 1(b)). The tasks are as follows.

**Task 1.** The robot task is to reach the red point on the map

**Task 2.** The robot task is to bring coffee to the room

**Task 3.** The robot task is to bring coffee to the room<sup>4</sup>

**Task 4.** The robot task is to take coffee from room 1 and bring it to room 2

For each round, if the participant chooses to monitor, a step-by-step plan execution is shown to them and with an option to stop the robot at any step. At the end of each round, a four-item trust scale of Muir questionnaire (Muir 1994) is given to them, which measures their trust in that

<sup>2</sup>Their primary responsibility is to ensure the robot completes its task

<sup>3</sup>All the details of user experiment setup, including the tasks are provided in the supplementary materials

<sup>4</sup>Tasks 2 and 3 are different because the maps (and conditions) are different.

round based on the robot's predictability, dependability, faith, and trust. Then, based on the measure of trust, a new robot's task is shown to them. In the user study to localize user trust, we will associate each level with a specific discretization of this range. Specifically, we will consider the following intervals for each of the levels,  $\{[0, 0.25], (0.25, 0.5], (0.5, 0.75], (0.75, 1]\}$ . Depending on the condition the participant belonged to, they are either shown an action selected by a policy calculated from our method (for Trust-aware condition), or an explicable plan (for Always explicable condition) or is randomly shown either the optimal or explicable plan with an equal probability (for Random Policy condition).

**Human Subjects** We recruited a total of 62 participants, of whom 38% were undergraduate, and 62% were graduate students in Computer Science, Engineering, and Industrial Engineering at our university. We paid them a base of \$10 for the study and a bonus of 1¢ per point, given the total points they will get in ten rounds. From the participants, 24 were assigned to the trust-aware condition, 18 to always explicable condition, and finally 20 to the random policy condition. Then, we filtered out any participants who monitored the robot in less than four rounds because they wouldn't have monitored the robot long enough to have a correct expectation in regards to the robot behavior.

**Results** Across all the three conditions, we collected (a) participants' trust measures in each round, (b) robot's total plan execution cost, and (c) participants' monitoring cost. For the monitoring cost, we consider the minutes participants spent on monitoring the robot in each round, which was approximately 3 minutes for each round of monitoring. As shown in Figure 2, we can see that the total cost (the robot's plan execution cost and the participant's monitoring cost) when the robot executes trust-aware behavior is significantly lower than the other two cases which means that following trust-aware policy allows the robot to successfully optimize the team performance. From Figure 3, we also observe that the trust (as measured by the Muir questionnaire) improves much more rapidly when the robot executes trust-aware policy as compared to the random policy. Though the rate for the trust-aware policy is less than the always explicable case, we believe this is an acceptable trade-off since following the trust-aware policy does result in higher performance. Also, we expect trust levels for trust-aware policies to catch up with the always-explicable conditions over longer time horizons.

**Statistical Significance**—We tested the two hypotheses by performing a one-tailed p-value test via t-test for independent means with results being significant at  $p < 0.05$  and find that results are significant for both hypotheses. 1) For the first hypothesis H1, we tested the mean cost with the null hypothesis of team performance cost has a mean of 3170.199 (using data from the "always explicable" scenario), we find that p-value is less than 0.00001. 2) For the second hypothesis H2, we tested the mean trust value for the last round and mean value over last two rounds with the null hypothesis being the trust value has a mean of 0.5458 and

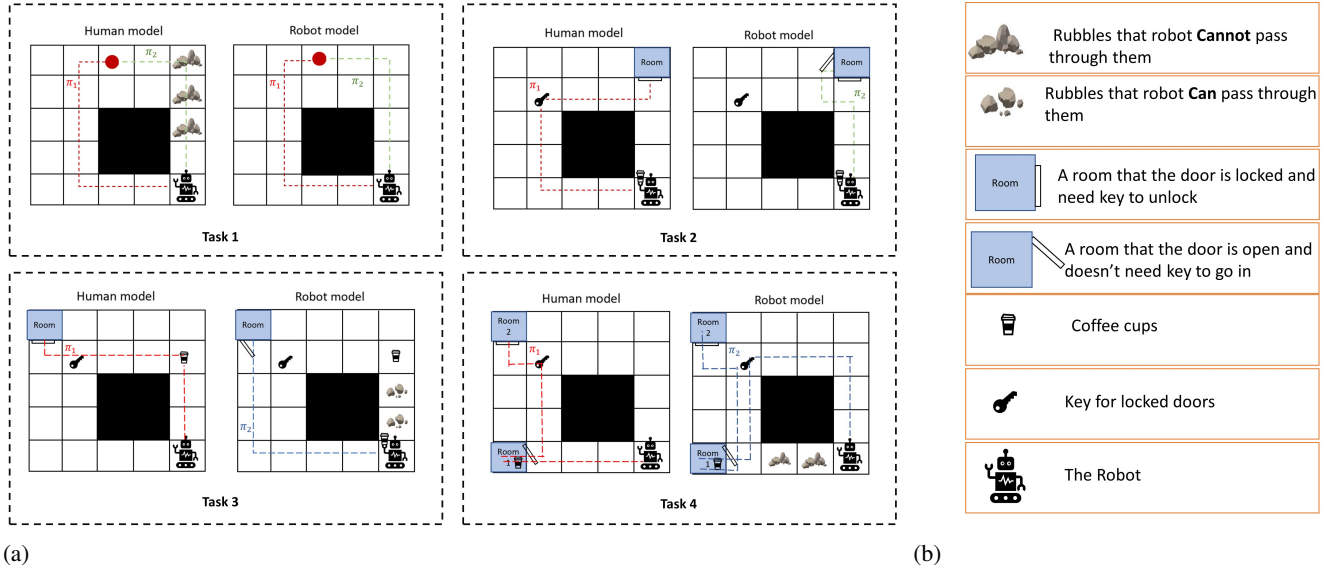


Figure 1: (a) The human and the robot model of the map for the four different tasks, (b) The map description

0.5416 in the last round and last two rounds respectively (using the data from "Random" scenario), we find that the p-values are 0.03174 and 0.03847 respectively. So, the results are statistically significant and show the validity of our hypotheses.

Also, we ran Mixed ANOVA test to determine validity of second hypothesis H2, and we found that there was a significant time (round)<sup>5</sup> by condition interaction  $F(1, 27) = 4.72$ ,  $p = 0.039$ ,  $\eta_p^2 = 0.15$ . Planned comparison with paired sample t-test revealed that in participant in Trust-Aware condition, trust increases significantly in round 10 compare to round 1,  $t = 3.55$ ,  $p = 0.002$ ,  $d = 0.84$ . There was however no difference in trust increase between round 1 and round 10 in the Random Policy condition  $t = -0.15$ ,  $p = 0.883$ ,  $d = -0.046$ . Both of these results follow our expectation about the method. Moreover, we ran Mixed ANOVA test on Trust-Aware vs. Always Explicable condition to check trust evolution over time, and we found that there was no significant time (round) by condition interaction  $F(1, 26) = 2.21$ ,  $p = 0.149$ ,  $\eta_p^2 = 0.08$ . Planned comparison with paired sample t-test revealed that in participant in Trust-Aware condition, trust increases significantly in round 10 compare to round 1,  $t = 3.55$ ,  $p = 0.002$ ,  $d = 0.84$ . There was also significant difference in trust increase between round 1 and round 10 in the Always Explicable condition  $t = 5.04$ ,  $p = 0.001$ ,  $d = 1.59$ . This seems to imply that there isn't a significant difference between our Trust-aware method (which is a lot more cost efficient) and Always Explicable case with regards to engendering trust.

<sup>5</sup>We considered the change over first and last rounds

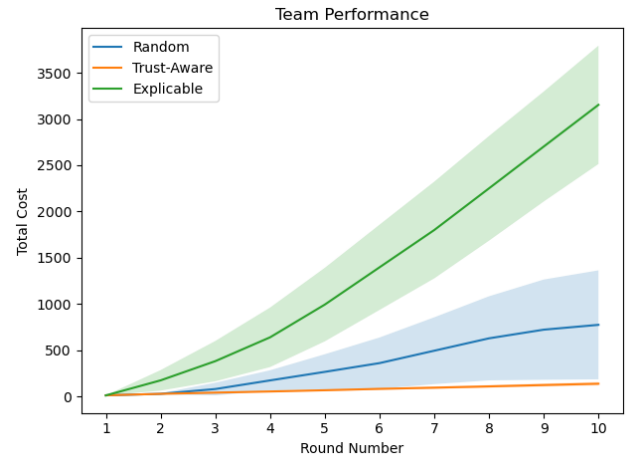


Figure 2: Team performance as cumulative plan execution cost and participants' monitoring cost (Mean  $\pm$  std of all participants).

## Conclusion and Discussion

In this paper, we presented a computational model that the robot can use to capture the evolution of human trust in longitudinal human-robot interactions. This framework allows the robot to incorporate human trust into its planning process, thereby allowing it to be a more effective teammate. Thus our framework would allow an agent to model, foster, and maintain the trust of their fellow teammates. Thereby causing the agent to engage in trust engendering behavior earlier in the teaming lifecycle and be able to leverage trust built over these earlier interactions to perform more efficient but potentially inexplicable behavior later on. As our experimental studies show, such an approach could result in a

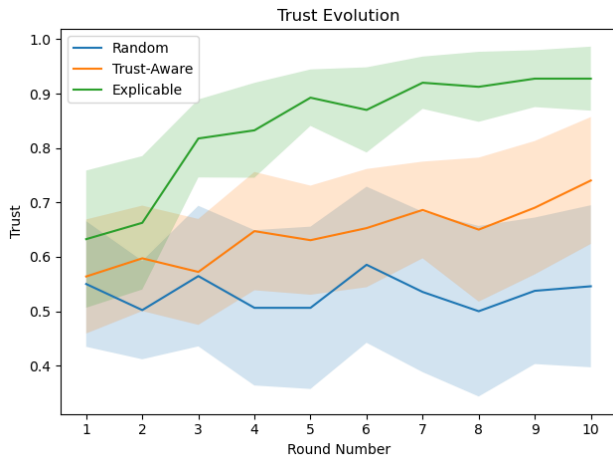


Figure 3: Trust evolution (as measured by the Muir questionnaire) through robot interactions with participants (Mean  $\pm$  std of all participants).

much more efficient system than one that always engages in explicable behavior. We see this framework as the first step in building such a longitudinal trust reasoning framework. Thus a natural next step would be to consider POMDP versions of the framework, where the human’s trust level is a hidden variable. We also plan to investigate methods to effectively learn the various parameter of our Meta-MDP or perform direct RL over this MDP.

**Acknowledgments.** This research is supported in part by ONR grants N00014-16-1-2892, N00014-18-1-2442, N00014-18-1-2840, N00014-9-1-2119, AFOSR grant FA9550-18-1-0067, DARPA SAIL-ON grant W911NF-19-2-0006, NASA grant NNX17AD06G, and a JP Morgan AI Faculty Research grant.

## References

Chakraborti, T.; Sreedharan, S.; and Kambhampati, S. 2017. Balancing explicability and explanation in human-aware planning. *arXiv preprint arXiv:1708.00543*.

Chakraborti, T.; Sreedharan, S.; Zhang, Y.; and Kambhampati, S. 2017. Plan explanations as model reconciliation: Moving beyond explanation as soliloquy. *arXiv preprint arXiv:1701.08317*.

Chen, M.; Nikolaidis, S.; Soh, H.; Hsu, D.; and Srinivasa, S. 2018. Planning with trust for human-robot collaboration. In *Proceedings of the 2018 ACM/IEEE International Conference on Human-Robot Interaction*, 307–315.

Chen, M.; Nikolaidis, S.; Soh, H.; Hsu, D.; and Srinivasa, S. 2020. Trust-aware decision making for human-robot collaboration: Model learning and planning. *ACM Transactions on Human-Robot Interaction (THRI)* 9(2): 1–23.

Cordwell, S. 2012. Markov Decision Process (MDP) Toolbox for Python. <https://github.com/sawcordwell/pymdpntoolbox>.

Cummings, M. 2004. Automation bias in intelligent time critical decision support systems. In *AIAA 1st Intelligent Systems Technical Conference*, 6313.

Guo, Y.; Zhang, C.; and Yang, X. J. 2020. Modeling Trust Dynamics in Human-robot Teaming: A Bayesian Inference Approach. In *Extended Abstracts of the 2020 CHI Conference on Human Factors in Computing Systems*, 1–7.

Hadfield-Menell, D.; Dragan, A.; Abbeel, P.; and Russell, S. 2016. The off-switch game. *arXiv preprint arXiv:1611.08219*.

Helmert, M. 2006. The fast downward planning system. *Journal of Artificial Intelligence Research* 26: 191–246.

Kulkarni, A.; Sreedharan, S.; Keren, S.; Chakraborti, T.; Smith, D. E.; and Kambhampati, S. 2019. Design for Interpretability. In *ICAPS Workshop on Explainable AI Planning (XAIP)*.

Liu, R.; Jia, F.; Luo, W.; Chandarana, M.; Nam, C.; Lewis, M.; and Sycara, K. P. 2019. Trust-Aware Behavior Reflection for Robot Swarm Self-Healing. In *AAMAS*, 122–130.

Muir, B. M. 1994. Trust in automation: Part I. Theoretical issues in the study of trust and human intervention in automated systems. *Ergonomics* 37(11): 1905–1922.

Rousseau, D. M.; Sitkin, S. B.; Burt, R. S.; and Camerer, C. 1998. Not so different after all: A cross-discipline view of trust. *Academy of management review* 23(3): 393–404.

Sengupta, S.; Zahedi, Z.; and Kambhampati, S. 2019. To monitor or to trust: observing robot’s behavior based on a game-theoretic model of trust. In *Proceedings of the Trust Workshop at AAMAS*.

Soh, H.; Xie, Y.; Chen, M.; and Hsu, D. 2020. Multi-task trust transfer for human–robot interaction. *The International Journal of Robotics Research* 39(2-3): 233–249.

Sreedharan, S.; Chakraborti, T.; Muise, C.; and Kambhampati, S. 2020a. Expectation-Aware Planning: A Unifying Framework for Synthesizing and Executing Self-Explaining Plans for Human-Aware Planning. *AAAI*.

Sreedharan, S.; Kulkarni, A.; Chakraborti, T.; Smith, D. E.; and Kambhampati, S. 2020b. A Bayesian Account of Measures of Interpretability in Human-AI Interaction. *arXiv preprint arXiv:2011.10920*.

Wang, X.; Shi, Z.; Zhang, F.; and Wang, Y. 2015. Dynamic real-time scheduling for human-agent collaboration systems based on mutual trust. *Cyber-Physical Systems* 1(2-4): 76–90.

Wang, Y.; Humphrey, L. R.; Liao, Z.; and Zheng, H. 2018. Trust-based multi-robot symbolic motion planning with a human-in-the-loop. *ACM Transactions on Interactive Intelligent Systems (TiiS)* 8(4): 1–33.

Xu, A.; and Dudek, G. 2012. Trust-driven interactive visual navigation for autonomous robots. In *2012 IEEE International Conference on Robotics and Automation*, 3922–3929. IEEE.

Xu, A.; and Dudek, G. 2015. Optimo: Online probabilistic trust inference model for asymmetric human-robot collaborations. In *2015 10th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, 221–228. IEEE.

Xu, A.; and Dudek, G. 2016. Maintaining efficient collaboration with trust-seeking robots. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 3312–3319. IEEE.

Zhang, Y.; Sreedharan, S.; Kulkarni, A.; Chakraborti, T.; Zhuo, H. H.; and Kambhampati, S. 2017. Plan explicability and predictability for robot task planning. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, 1313–1320. IEEE.



# Deliberation and Plan Execution for Intra-vehicle Robotic Activities in Space

**J. Benton, Abiola Akanni, Robert Morris**

Intelligent Systems Division  
NASA Ames Research Center  
{j.benton,abiola.o.akanni,robert.a.morris}@nasa.gov

## Abstract

Intra-Vehicular Robotics (IVR) for space exploration vehicles describes robotic capabilities to perform Intra-vehicle activity (IVA) in an autonomous or remotely operated manner. This paper focuses on autonomy, and more specifically, on the potential application of deliberation functions and plan execution in robotics to enable autonomous IVR. We provide an overview of the capabilities required to enable goal-directed operations, robotic systems' ability to autonomously transfer a high-level goal into a set of tasks to accomplish them.

## Introduction

Intra-Vehicular Robotics (IVR) for space exploration vehicles refers to robots capable of performing Intra-vehicle activity (IVA) in an autonomous or remotely operated manner. IVAs include state assessment (including inspection, inventory, anomaly detection), logistics (moving and stowing cargo), fault management (all phases), and science operations. NASA researchers explore IVR on Gateway, a spaceport in lunar orbit that will serve as a gateway to deep space and the lunar surface. Since Gateway will primarily be uncrewed for nine to eleven months out of the year, IVR is critical and essential to maintain and protect the vehicle.

This position paper focuses on autonomous IVR/IVA, and more specifically, on the potential application of automated planning, plan execution, and other deliberation functions in robotics. Deliberative functions include planning, acting (refining actions into sensory-motor control), plan execution monitoring, observing, and learning. For plan execution monitoring, we have begun considering a robust distributed plan monitoring approach. This work has produced an architectural design and closed-loop framework of a system for goal-directed commanding, automated goal management, task planning, robust execution, execution monitoring, and replanning.

Our main contribution of this paper is to define and illustrate the role of planning and execution for IVR. The rest of the article is structured as follows: we define vehicle system management and use the Gateway mission as a working example; then, we define a set of requirements and technical challenges in planning and execution for IVR. The work

Copyright © 2021, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

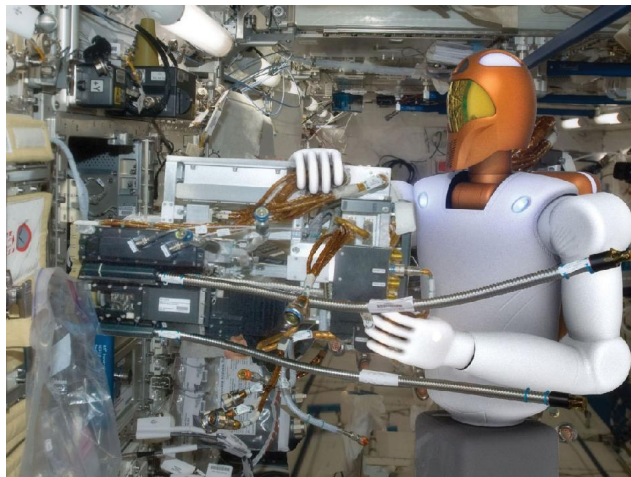


Figure 1: R2 robotic assistant on the International Space Station

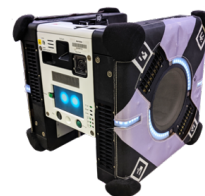


Figure 2: Astrobee Free Flying Robot

summarized here encapsulates a multi-year effort to demonstrate the effective use of task planning in IVR science activities for robotic manipulators.

## Exploration Vehicle Management

Space vehicle system management is a joint effort of ground systems and personnel, vehicle systems, robotic systems, and vehicle crew. IVR services support all types of operations and vehicle system management. Two vehicle systems management methods are emerging: the first is necessary when a team is on board and operating the vehicle but requires support that the ground controllers cannot

give. The second involves vehicle control when no one is on board. Uncrewed vehicle management includes so-called “dormant” periods, when the vehicle is in a state when only the minimal subsystems are required to maintain system health, as well as periods of fully operational autonomous operations (Badger, Strawser, and Claunch 2019).

Exploration Missions (EMs) will extend human presence into deep space. Current designs for EMs include incrementally building infrastructure, operational experience, and testing of systems required for long-duration missions in deep space. These activities will be conducted on the Deep Space Gateway (DSG) (Gateway 2018). DSG is a spaceport in cislunar orbit that will serve as a vehicle to deep space and the lunar surface.

Intra-Vehicle Activity (IVA) includes the following:

- *State assessment*, including inspection of vehicle, inventory, and detection of off-nominal conditions;
- *Logistics*, including cargo transport and stowage and opening and closing hatches;
- *Integrated fault management*, including detection, isolation and repair; and
- *Science operations*, for example, biological experiments involving the manipulation and imaging of biological samples.

Each of these activities potentially involves complex, coordinated planning. Furthermore, some activities can be viewed as routine (predictable, performed periodically). In contrast, others might be conducted in response to an off-nominal event, such as detecting a leak or unexplained noise. This suggests the need for *continuous* operations planning, the ability to accept new goals at any time during operations (Brenner and Nebel 2009).

This work is being applied as part of the Integrated System for Autonomous and Adaptive Care-taking (ISAAC) program, a system for monitoring the telemetry from the International Space Station (ISS) systems, and, eventually, on the DSG. The goal of the ISAAC program is to provide autonomous spacecraft caretaking during uncrewed periods. The focus of ISAAC is to integrate AI Task Planning with robot behaviors to enable *goal-based operations*: the ability to command the robots at a high level of abstraction (setting goals) and having the robots deliberate autonomously to plan the execute to accomplish the goals.

To illustrate the technical challenges of IVR for EMs, use cases involving cargo transport logistics and integrated fault management are evaluated. The following summarizes these use cases.

**Transporting Cargo Bags:** In this scenario, one or more Cargo Transfer Bags (CTBs) need to be retrieved and transported between a cargo vehicle and a Gateway module. A typical logistics task might require the robot to approach and grasp a CTB with a magnetic gripper. The CTB is equipped with a bag fixture that enables magnetic gripping, and the stowage location is equipped with a berth fixture to hold the item.

Following the grasp, the bag is magnetically released from its stowage location. The robot transports the CTB to

its targeted location on Gateway, at which time it is stowed on a new stowage location, using a similar sequence that involves magnetic connectors. In addition to the routine grasp/ungrasp/transport actions, solving logistics can also involve set-up tasks such as installing bag fixtures or berth fixtures. Logistics may involve the coordination of multiple robots performing different sub-tasks. For example, some robots cannot traverse certain areas of Gateway. In this case, the robot must place the CTB in a staging area for another robot to then pick up the CTB and move it to its final goal location.

**Integrated Fault Management:** In this scenario, a micro-meteoroid strike has caused a leak in a Gateway module during an uncrewed mission phase. The leak must be patched within a matter of hours to avoid significant impacts, such as losing pressurized payloads that cannot tolerate depressurization.

There are three leak management phases: detection, localization, and mitigation. During the detection phase, vehicle sensor information such as pressure sensor trend analysis or the sound of thrust generated by the leak is used to signal the presence of a leak.

The second phase is localization, which works through several mechanisms. It uses coarse localization, where coarse sensor data is used to localize within, say, a 2 by 2-meter area. It begins with a survey procedure, requiring preparation actions like turning off noisy systems that mask the leak noise. We then perform preparation, where a mobile inspection robot with ultrasound sensors can find noise sources indicating the leak. Finally, during report and confirmation, the robot communicates precise leak location and may confirm using other sensor data.

During the mitigation phase, and depending on the location and type of leak, a mobile manipulator robot may patch it using a patch kit. Otherwise, mitigation would focus on steps like moving sensitive equipment out of the affected module and closing hatches to isolate it from the rest of the vehicle.

## IVA Technical Challenges for Planning and Execution

In this section we consider autonomous IVR from three perspectives: from a consideration of features of the planning problem; from a plan-based robotic architecture point of view; and briefly considering interfaces with other human and automated systems. Each of these topics overlaps with topics of interest to the AI planning community.

### Summary of Technical Challenges for IVA

The examples above illustrate several challenges for planning and execution systems for IVR. Among those challenges are:

- Planning and acting under uncertainty; in particular, the state of the world is only partially known at planning time and may change unpredictably during execution; the actions may be stochastic, in that the effect of an action may not be known with certainty; and durations of actions may not be known with certainty;

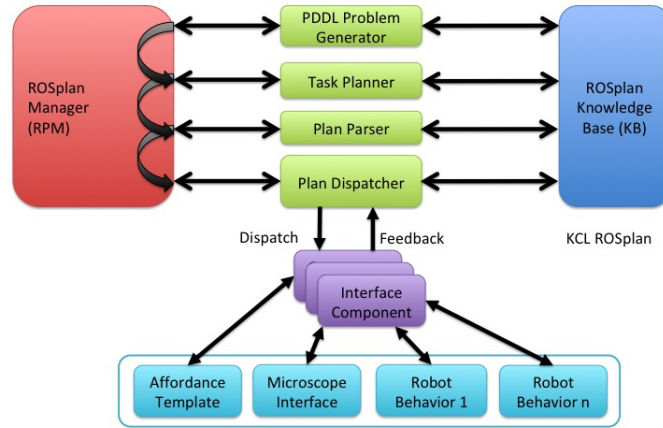


Figure 3: RosPlan Architecture

- An activity may be time-critical; a sequence of actions must be performed before a deadline.
- Planning is continuous: new activities may be posted while others are being executed; and
- Coordination of knowledge and actions is required to attack and solve the problem. This coordination may be robot-robot, or robot-ground system, or robot-vehicle.

### Plan-based Robotic Architectures

Robot architectures for deliberation have been developed for many years (Ingrand and Ghallab 2017). There has been a consensus that robotic architectures are layered, with separate components for planning, execution, sensing and control. An example of a layered architecture is the ROSPlan framework (Cashmore et al. 2015), illustrated in Figure 3, which we are currently using for ISAAC task planning. ROSPlan consists of basic ROS capabilities for problem generation, PDDL planning, plan execution and knowledge management. It provides action interface components for mapping atomic PDDL actions to commands to robots. In previous work (Azimi, Zemler, and Morris 2019) we expanded the capabilities to include a simple goal management and execution management and replanning system.

To this point, we have used ROSPlan to solve simple logistics scenarios involving coordinated bag transfer IVAs with R2 and Astrobbee. These have been tested in Gazebo simulation (see Figure 4). A number of critical challenges to be explored as we scale up to more complex scenarios include the following topics of relevance to the AI planning community:

**Support for continual planning and goal management** IVA requires planners to make decisions online, while previous plans are being executed. Frameworks for goal prioritization, plan preemption, and re-planning are potentially needed in these contexts.

**Scalable planning** Although AI planning algorithms have matured over the years, it is not clear whether complex problems combining multiple agents and uncertainty can be solved effectively with current systems.

**Robust Monitoring and Execution** Developing robot architectures for IVA will require design decisions to be made about achieving robust deliberative behavior through a balance between expressive plan domain models and languages and robust techniques for plan execution monitoring. It is not clear yet where that balance resides and more testing of different designs are required.

**Task and Motion Planning** IVA is a good example of the need for architectures for combined task and motion planning (TAMP). A large and growing body of work is addressing fundamental challenges raised by TAMP, such as reasoning jointly with symbolic and continuous domains, and at different levels of abstraction (Mansouri, Pecora, and Schüller 2021).

### Interfaces to Other Deliberation Systems

Robotic planning and acting for IVA are never isolated individual activities. It is necessary to consider interfaces between robotic and other deliberation systems:

- **ground operator-robot:** developing hybrid interfaces combining a range of tele-operation and goal-based operations (Dvorak et al. 2012)
- **crew-robot:** human-robot coordination; for example, robot assistants in space (Chang and Mogg 2018)
- **vehicle-robot:** robotic systems integrated into vehicle autonomy (Badger, Strawser, and Claunch 2019)
- **robot-robot:** Some IVAs need to be solved through coordination of multiple robots. (Yan, Jouandeau, and Cherif 2013)

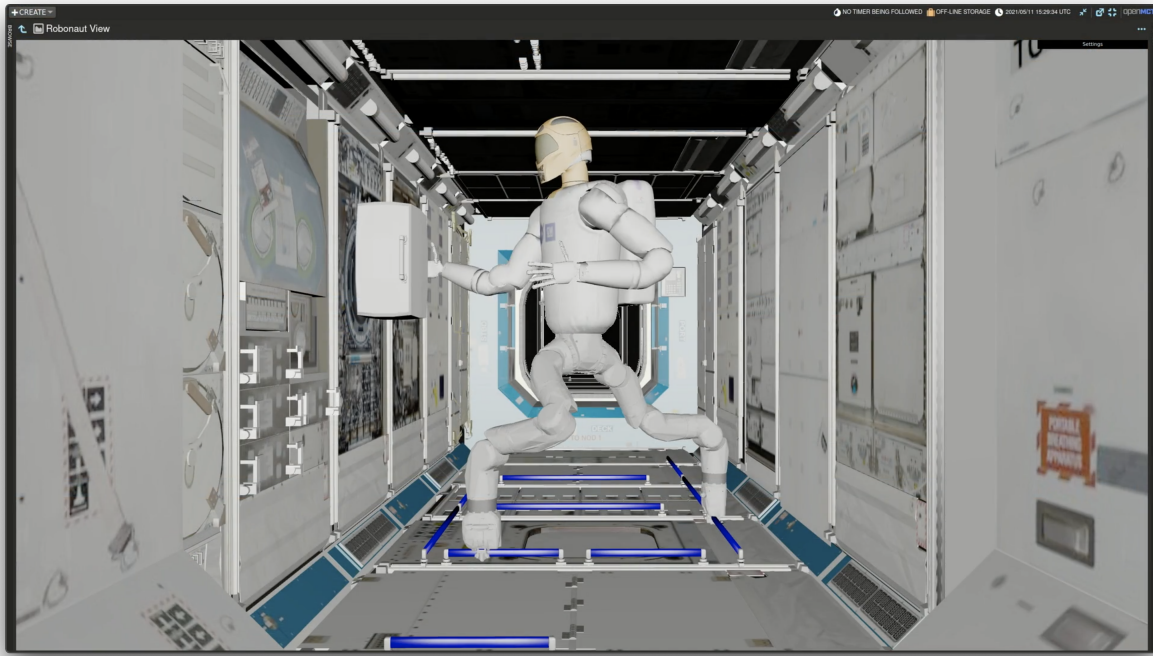


Figure 4: R2 Robot in Gazebo Simulating a Bag Transfer Scenario on the ISS

Each of these combinations potentially involves the need to consider design options for extending plan-based architectures. As an illustration, the authors are in the processing of testing modifications to ROSPlan (Figure 3) to enable dispatching of plans using multiple dispatchers, one for each robot. Distributed decision-making requires an infrastructure for coordination. To that end, we're currently exploring coordinated scheduling in ROSPlan using a centralized STN dispatcher (Smith, Gallagher, and Zimmerman 2007).

## Summary

This position paper has provided an overview of deliberation systems for Intra-Vehicle Robotic systems on future exploration vehicles. We have sketched how we are using AI planning and execution systems to solving IVA planning. Many challenges remain and we encourage the community to investigate these problems.

## Acknowledgements

This work was performed under NASA's Integrated System for Autonomous and Adaptive Care-taking (ISAAC) Project. The rest of the ISAAC team: Trey Smith, Maria Bualat, Oleg Alexandrov, Brian Coltin, Terry Fong, Janette Garcia, Kathryn Hamilton, Lewis Hill, Marina Moreira, Nicole Ortega, Joseph Peal, Jonathan Rogers, Misha Savchenko, Khaled Sharif.

## References

- Azimi, S.; Zemler, E.; and Morris, R. A. 2019. Autonomous Robotics Manipulation for In-space Intra-Vehicle Activity. In *Workshop on Planning and Robotics (PlanRob)*.
- Badger, J. M.; Strawser, P.; and Claunch, C. 2019. A Distributed Hierarchical Framework for Autonomous Spacecraft Control. In *2019 IEEE Aerospace Conference*, 1–8.
- Brenner, M.; and Nebel, B. 2009. Continual Planning and Acting in Dynamic Multiagent Environments. *Journal of Autonomous Agents and Multiagent Systems* 19: 297–331.
- Cashmore, M.; Fox, M.; Long, D.; Magazzeni, D.; Ridder, B.; Carrera, A.; Palomeras, N.; Huros, N.; and Carreras, M. 2015. ROSPlan: Planning in the Robot Operating System. In *Proceedings of the Twenty-Fifth International Conference on Automated Planning and Scheduling*, 333–341. ICAPS.
- Chang, L.; and Mogg, T. 2018. SpaceX delivers CIMON, along with berries and ice cream, at ISS. In *Emerging Tech. Digital Trends*.
- Dvorak, D. D.; Ingham, M. D.; Morris, J. R.; and Gersh, J. 2012. Goal-Based Operations: An Overview. *Journal of Aerospace Computing, Information and Communication*.
- Gateway. 2018. *Q&A: NASA's New Spaceship*. <https://www.nasa.gov/feature/questions-nasas-new-spaceship>.
- Ingrand, F.; and Ghallab, M. 2017. Deliberation for Autonomous Robots. *Artif. Intell.* 247(C): 10–44. ISSN 0004-3702. doi:10.1016/j.artint.2014.11.003. URL <https://doi.org/10.1016/j.artint.2014.11.003>.

Mansouri, M.; Pecora, F.; and Schüller, P. 2021. Combining Task and Motion Planning: Challenges and Guidelines. *frontiers in Robotics and AI* .

Smith, S. F.; Gallagher, A.; and Zimmerman, T. 2007. Distributed Management of Flexible Times Schedules. In *Proceedings of the 6th International Joint Conference on Autonomous Agents and Multiagent Systems*, AAMAS '07. New York, NY, USA: Association for Computing Machinery. ISBN 9788190426275. doi:10.1145/1329125.1329215. URL <https://doi.org/10.1145/1329125.1329215>.

Yan, Z.; Jouandeau, N.; and Cherif, A. A. 2013. A Survey and Analysis of Multi-Robot Coordination. *International Journal of Advanced Robotic Systems* 10(12): 399. doi:10.5772/57313. URL <https://doi.org/10.5772/57313>.



# Compiling Contingent Planning into Temporal Planning for Robust AUV Deployments

Yaniel Carreno<sup>1,2,3</sup>, Yvan Petillot<sup>1,2</sup>, Ronald P. A. Petrick<sup>1,2</sup>

<sup>1</sup>Edinburgh Centre for Robotics, Edinburgh, United Kingdom

<sup>2</sup>Heriot-Watt University, Edinburgh, United Kingdom

<sup>3</sup>University of Edinburgh, Edinburgh, United Kingdom  
{y.carreno, y.r.petillot, r.petrick}@hw.ac.uk

## Abstract

Autonomous Underwater Vehicles (AUVs) are increasingly used to implement complex missions that require robots with a high level of operational autonomy. AI temporal planners are particularly well suited to solve a large number of these real-world problems which often involve temporal and numeric constraints. However, the applicability of such solvers is limited in cases where certain aspects of the domain are incomplete or unknown. In this work, we present a general approach to task planning based on the combination of temporal planning, contingent planning and run-time sensing. We introduce an approach which solves contingent problems using offline temporal planning, which reduces the risks associated with replanning in non-quiescent environments. This translation extends the range of problems that can be solved by temporal planners. We demonstrate the effectiveness of our approach with a new set of temporally-contingent planning problems from a real underwater domain.

## 1 Introduction and Motivation

As automated planning approaches have matured over the past decade, we have seen their introduction in a wide range of real-world applications, including AUV missions (Thompson and Guihen 2019) interested in ocean sampling, maintenance of offshore structures, emergency response, and military intervention. This can be attributed to the general applicability of planning algorithms (Kerschke et al. 2019) and the variety of input languages available for modelling different types of planning problems (e.g., classical (McDermott et al. 1998), temporal (Fox and Long 2003), contingent (Edelkamp and Hoffmann 2004), etc.). Regardless of the language, planning models must represent the domain properties, goals, constraints, and costs in order that the generated *plan*—a structured sequence of actions that guides the initial world state to a goal state—is suitable for execution in the world. As a result, planning models for real-world applications can be complex, representing numeric and temporal constraints and domain uncertainty.

Temporal planning solutions have proved to be effective for implementing AUV missions (Cashmore et al. 2014; Buksz et al. 2018; Carreno et al. 2021) as they provide knowledge about the activity schedule, deal with concurrent

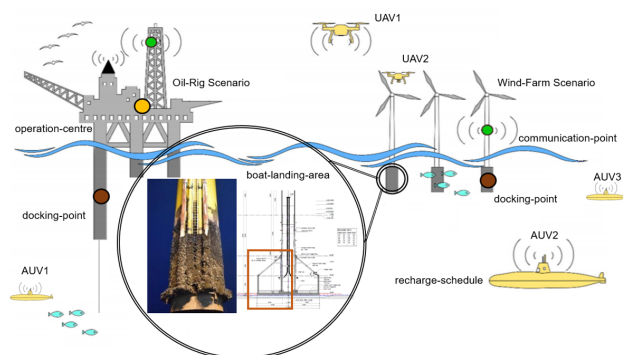


Figure 1: An offshore energy domain including a wind farm scenario. The domain presents multiple AUVs that are used for maintenance and inspection tasks. The AUVs embed a cavitation cleaning technology to remove biofouling from a Boat Landing Area (BLA) at the wind turbine base. The BLA state is unknown at planning time.

actions, and consider numeric constraints. However, such planners are based on deterministic planning models with predictable outcomes and completely known initial states, limiting their applicability in real-world domains that are often incomplete or unknown. One option for mitigating these issues is to introduce robust execution monitoring and on-line replanning, in order to detect and reactively respond to unexpected changes in the world. However, the replanning process requires additional computational effort during plan execution. In non-quiescent domains (i.e., underwater), replanning introduces delays associated with the generation of new plans that can lead to a plan no longer being valid for the AUV’s current state when execution begins.

Another solution to deal with knowledge incompleteness is the application of contingent planning approaches. Contingent planning (Peot and Smith 1992; Weld, Anderson, and Smith 1998) copes with certain types of incomplete information by treating the plan output as a decision tree with a set of different contingent branches that could arise. Unlike the replanning approach, contingent planning has more offline computation and generates a plan solution that considers decision points predicated on sensing outcomes. A contingent plan guides the agent to act conditionally to achieve the goal,



with *sensing actions* in the decision tree enabling the planner to decide which branch to take. However, these planners are not usually suitable for solving problems with numeric and temporal constraints. This paper focuses on temporal planning problems with numeric conditions where the action sequences required to reach the goals give rise to conditional plans and reasoning about incomplete information resulting from sensing actions. Consider the following example:

**Example 1 (Biofouling Cleaning).** An offshore underwater scenario (see Figure 1) includes an energy structure and a wind farm with several turbines that require regular inspection and maintenance. One common problem for wind turbines is biofouling that causes degradation in the base, seriously affecting the boat landing area (BLA) and obstructing the transportation of operators to work in other parts of the structure. The mission goal is to clean the BLA of three turbines ( $t_1$ ,  $t_2$  and  $t_3$ ). An AUV with underwater monitoring equipment and cavitation cleaning equipment is used to inspect the BLA regularly, evaluate its state (obstructed/non-obstructed) and implement cleaning actions in the area if this is required. In the initial state, the robot is at the docking point. From there, it is possible to navigate to the BLA, and from there, the AUV inspects the area. The AUV’s action depends on the area’s state: if the area presents biofouling (obstructed), then the AUV should clean it, which is a highly energy consuming action; if the area is clean (non-obstructed), then the AUV does not need to perform any further action associated to this goal. The state of the BLA is unknown at planning time and can be checked using a sensing action. The robot needs to recharge the battery by sharing a docking point with other AUVs with scheduled recharging times. Finally, the robot has to communicate to the operation centre every time a cleaning action is completed.

We note that no sequence of actions allows the AUV to achieve the goals without first gathering additional knowledge from the BLA, which requires reasoning about incomplete sensing information. Since domain incompleteness is limited to a few possible states for the BLA, the problem is suitable for contingent planning. In addition, the problem involves temporal and numeric conditions. Temporal constraints are essential for scheduling the recharging tasks with the docking point unavailable for specific time slots while other AUVs are recharging. Our robot would also need to recharge at different times depending on the number of cleaning actions it has to implement and the goal locations. Finally, the domain presents numeric constraints associated with reporting the cleaning actions to the operation centre.

This paper proposes an approach for compiling contingent plan construction into a temporal planning framework for AUV missions. We present a new planning solution which can reason about incomplete knowledge while meeting the temporal and numeric constraints of the problem. We test our approach by solving contingent problems offline using temporal planners. We also provide a new AUV planning domain (inspired by Example 1) that we use to evaluate the performance of our solver in simulated and real scenarios. This translation introduces additional flexibility in the plan construction when performing tasks in dynamic environments.

## 2 Related Work

The planning community has had a long-standing interest in the problem of planning with incomplete information and sensing actions. Examples of early offline contingent planners which address these problems are PKS (Petrack and Bacchus 2002), Contingent-FF (Hoffmann and Brafman 2005), CLG (Albore, Palacios, and Geffner 2009), and DNF and CNF (To, Pontelli, and Son 2011). These planners are able to search and deliberate, have the capacity to avoid deadends, and also support the generation of high-quality optimised plans. Palacios, Albore, and Geffner (2014); Brafman and Shani (2012); Bonet and Geffner (2014) present compilation-based approaches that compute full offline solutions using classical planning. However, finding solutions for all contingent branches is conditioned by classical planning heuristic generation methods. Other approaches (Muise, Belle, and McIlraith 2014; Camacho, Muise, and McIlraith 2016) generate smaller and faster conditional plans by treating the problem as Fully Observable and Non-Deterministic (FOND). None of these contingent planners reason explicitly about time. We connect contingency analysis with temporal reasoning in our approach which is an understudied area of research.

Few approaches in the literature consider both incomplete information and explicit notions of time. Temporal planning with solvers based on heuristic forward search have shown the best performance while generating and executing real underwater deployments (Cashmore et al. 2014; Buksz et al. 2018). Examples of planners reasoning about time are POPF (Coles et al. 2010), and OPTIC (Benton, Coles, and Coles 2012). These approaches solve a large number of well-established domains (Long and Fox 2003) that require temporal and numeric analysis, providing good scalability performance. However, their solutions do not consider model’s uncertainty. Tsamardinos, Vidal, and Pollack (2003) proposed the CPT formalisation, which adds observation nodes and attaches labels to all nodes to indicate conditions for a node’s execution. CPT allows reasoning about the construction of conditional plans with temporal constraints. This formalisation is further extended by the TCP framework (Foss and Onder 2005) to cope with parallel plans, non-temporal metrics and multiple planner goals. TCP is a greedy iterative algorithm that inserts branches based on time rather than world conditions. This differs from our approach, where the acquisition of unknown world information leads to a plan solution. Strategies based on Simple Temporal Networks with Uncertainty (STNU) such as (Cimatti et al. 2014) are used as temporal scheduling tools where conditions and decisions can be added to STNUs (Combi, Hunsberger, and Posenato 2013; Zavatteri and Viganò 2019). Combi et al. (2019) present an encoding to Conditional Simple Temporal Networks with Uncertainty and Resources (CSTNURs) with promising results in a set of applications. These solutions do not focus on solving problems where incomplete information is acquired using sensing actions. Finally, some work associated with temporal plan merging (Hashmi and Seghrouchni 2010) and opportunistic planning consider temporal (Cashmore et al. 2017) and resource (Coles 2012) constraints. However, these approaches aim to generate al-

ternative branches to achieve soft goals, which differs from our solution where hard goal completion requires contingent reasoning.

### 3 Temporally-Contingent Planning

In this section we formulate the temporally-contingent planning (TCP) problem and review the policy solving the model. In this work we adopt the Planning Domain Definition Language (PDDL) (McDermott et al. 1998).

**Problem Definition.** Considering PDDL2.1 (Fox and Long 2003), Definition 1 defines a propositional temporal planning problem, with Timed Initial Literals (TILs) (Cresswell and Coddington 2003).

**Definition 1.** A temporal planning problem is a tuple  $\mathcal{P}_T := \langle \mathcal{P}, \mathcal{F}, \mathcal{A}_T, \mathcal{I}_T, \mathcal{G}_T, \mathcal{T} \rangle$ , where  $\mathcal{P}$  is set of atomic propositions describing the state of the world;  $\mathcal{F}$  is a set of task numeric variables called fluents;  $\mathcal{A}_T$  is a set of instantaneous and durative actions, with controllable and known duration;  $\mathcal{I}_T : \mathcal{P} \cup \mathcal{F} \rightarrow \{\top, \perp\} \cup \mathbb{R}$  is a total function describing the initial state of predicates and functions;  $\mathcal{G}_T : \mathcal{P} \cup \mathcal{F} \rightarrow \{\top, \perp\} \cup \mathbb{R}$  is a (possibly) partial function that describes the goal conditions, where each goal  $g \in \mathcal{G}_T$  is a  $p$ , where  $p \subseteq \mathcal{P}$ ; and  $\mathcal{T}$  is a set of time windows. Each time window is defined using timed initial literals (TILs). Let  $\mathcal{Z}$  be the (finite) set of all TILs, where each TIL  $l = \langle t(l), \text{lit}(l) \rangle \in \mathcal{T}$  defines the time  $t(l)$  and the literal  $\text{lit}(l)$ , specifying which proposition  $p$  becomes true (or false) at time  $t(l)$ , where  $p \in \mathcal{P}$ .

**Definition 2.** An instantaneous action  $a_{it}$ , where  $a_{it} \in \mathcal{A}_T$  is a tuple  $\langle a_{it_{pre}}, a_{it_{eff}} \rangle$ ;  $a_{it_{pre}}$  is a set of preconditions that must hold for the action to be applicable; and  $a_{it_{eff}}$  is the set of action effects of type: positive effects ( $a_{it_{eff}}^+$ ), negative effects ( $a_{it_{eff}}^-$ ) and numeric effects ( $a_{it_{eff}}^n$ ).

**Definition 3.** A durative action  $a_{dt}$ , where  $a_{dt} \in \mathcal{A}_T$  is a tuple  $\langle a_{dt_{pre}}, a_{dt_{eff}}, a_{dt_{dur}} \rangle$ ;  $a_{dt_{pre}}$  is a set of conditions that must hold for the action to be applicable of type: at-start ( $a_{dt_{pre}}^+$ ), over-all ( $a_{dt_{pre}}^{\leftrightarrow}$ ), and at-end ( $a_{dt_{pre}}^-$ );  $a_{dt_{eff}}$  is the set of action effects of type: positive starting effects ( $a_{dt_{eff}}^+$ ), negative starting effects ( $a_{dt_{eff}}^-$ ), numeric starting effects ( $a_{dt_{eff}}^n$ ), continuous numeric effects ( $a_{dt_{eff}}^{\leftrightarrow}$ ), positive ending effects ( $a_{dt_{eff}}^+$ ), negative ending effects ( $a_{dt_{eff}}^-$ ), numeric ending effects ( $a_{dt_{eff}}^n$ ); and  $a_{dt_{dur}}$  is a set of duration constraints.

For the scope of this paper, temporal plans consider sequences of durative actions that are bounded by a starting and ending time. We call these plan solutions *time-aware plans* ( $\Pi_T$ ). Figure 2a shows a fragment of a sequence of actions that describe the solution of Example 1 where the problem does not present incomplete knowledge in the initial state  $\mathcal{I}_T$ . Therefore, the planning problem  $\mathcal{P}_T$  considers  $\mathcal{P}$  contains a proposition  $p$  which defines the state of the BLA at turbine  $t_1$  as (`bla_obstructed t1`) (require bio-fouling cleaning) and  $p \in \mathcal{I}_T$ . The plan structure in this case is a single sequence of durative actions (defined by the arrows) which transform the initial state  $S_0$  to a goal state  $S_g$ .

In this case no contingent branches are required to reach the goal state as the problem is completely known and the sensing action is not required to achieve incomplete information.

We now define the contingent planning problem. To do so, we consider partial observability and the existence of non-deterministic (sensing) actions. Contingent planning approaches do not consider temporal notions. Therefore, the reasoning around action duration is relegated to the run-time plan implementation. Additionally, we make a distinction between sensing actions and physical actions to simplify the notation. For instance, in Figure 2b, action `inspect-area` is a sensing action that provides the type of information required to generate contingent branches. On the other hand, physical actions such as `navigation` are ordinary planning actions that support goal implementation but do not contribute to contingent branch construction. Physical actions are instantaneous actions (see Definition 2).

**Definition 4.** A contingent planning problem is a tuple  $\mathcal{P}_C := \langle \mathcal{P}, \mathcal{A}_C, \Phi, \mathcal{I}_C, \mathcal{G}_C \rangle$ , where  $\mathcal{P}$  is set of atomic propositions describing the state of the world;  $\mathcal{A}_C$  is a set of instantaneous physical actions providing the means of change in the domain;  $\Phi$  is a set of sensing actions (observations), separate from  $\mathcal{A}_C$  such that  $\Phi \cap \mathcal{A}_C = \emptyset$ ;  $\mathcal{I}_C$  is the set of clauses over  $\mathcal{P}$  that denotes the initial state; and  $\mathcal{G}_C$  is a set of literals over  $\mathcal{P}$  representing the goals.

**Definition 5.** A sensing action  $\phi_i$ , where  $\phi_i \in \Phi$  is a tuple  $\langle \phi_{pre}, \phi_{eff} \rangle$ ;  $\phi_{pre}$  is a set of conditions that must hold for the action to be applicable; and  $\phi_{eff}$  reveal the truth value of a proposition  $p$ ,  $p \in \mathcal{P}$ .

The contingent planning problem  $\mathcal{P}_C$  presents two action types that have as preconditions a conjunction of literals. These two action types reveal differences in the action effects. A physical instantaneous action  $a_{ic} \in \mathcal{A}_C$ , is modelled by  $a_{ic} : a_{ic_{pre}} \rightarrow a_{ic_{eff}}$ , where  $a_{ic_{pre}}$  (a set of literals) characterises the preconditions and  $a_{ic_{eff}}$  (a set of literals) is the set of conditional effects. The effect of a physical action is defined by  $\phi_{eff}$ . A literal  $l$  is a proposition  $p \in \mathcal{P}$  or its negation  $\neg p$ . A set of literals  $\mathcal{L}$  is consistent if the condition  $\{p, \neg p\} \not\subseteq \mathcal{L}$  holds; and complete if  $\{p, \neg p\} \cap \mathcal{L} \neq \emptyset$  holds for every  $p \in \mathcal{P}$ . A state  $s$  is defined as a consistent and complete set of literals. A belief state  $b$  represents the set of world states that are possible.

A contingent plan solution  $\Pi_C$  for  $\mathcal{P}_C$  is a branching structure called a transition tree constructed from the actions and observations of  $\mathcal{P}$ . In practice, observations are treated as sensing actions. Figure 2b shows fragment of a contingent plan solution for Example 1. Note  $\Pi_C$  does not contemplate the action duration and therefore the solution is limited for problems with temporal requirements. The AUV cannot reason about the recharging time limitations imposed for sharing the docking point. If the robot needs to recharge and the point is occupied the AUV will need to wait considering it is not possible to establish previous reasoning to optimise the position of the action in the plan. The recognition of the domain incompleteness support the generation of branches that respond to all possible outcomes of the unknown proposition. A temporally-contingent planning problem  $\mathcal{P}_{TC}$  is  $\mathcal{P}_{TC} = \mathcal{P}_T \cup \mathcal{P}_C$ .

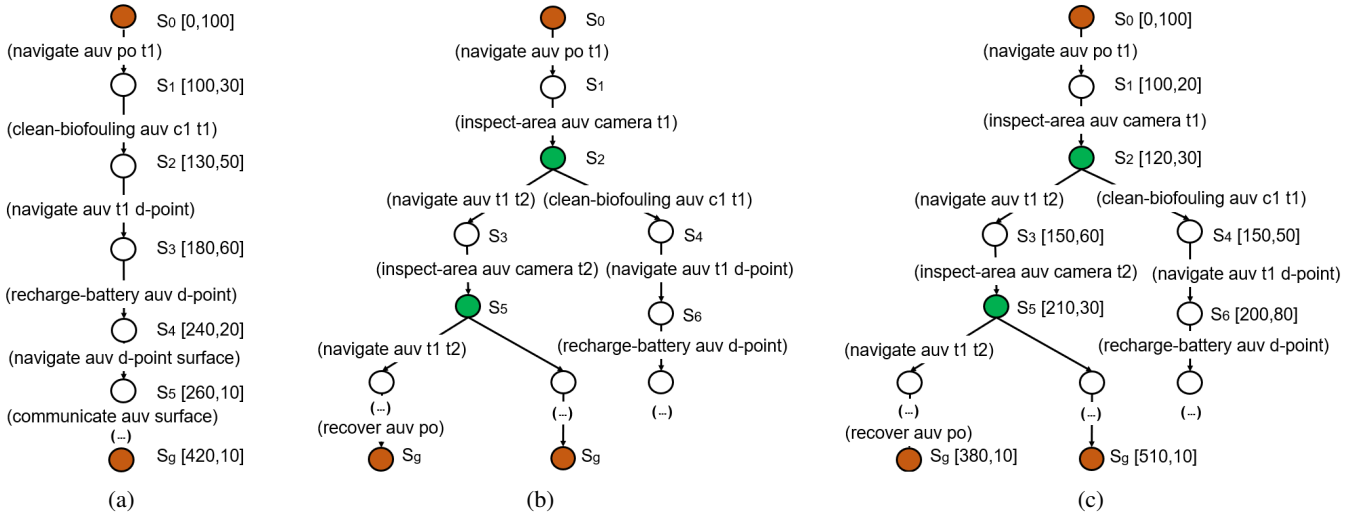


Figure 2: In (a), fragment of a temporal plan  $\Pi_T$ . In (b) fragment of a contingent plan. In (c) fragment of a contingent plan with temporal constraints  $\Pi_{TC}$  for Example 1.  $\Pi_T$  and  $\Pi_{TC}$  contains nodes representing durative actions and edges  $[t, d]$  representing the action starting time and duration.  $\Pi_C$  and  $\Pi_{TC}$  contain nodes (green) sensing actions (*inspect-area*), which lead to multiple sub-plans, and edges showing action's connections.

**Definition 6.** A temporally-contingent planning problem is a tuple  $\mathcal{P}_{TC} := \langle \mathcal{P}, \mathcal{F}, \mathcal{A}, \Delta, \mathcal{I}_T, \mathcal{G}_T, \mathcal{T} \rangle$ , where  $\mathcal{P}$  is set of atomic propositions describing the state of the world;  $\mathcal{F}$  is a set of task numeric variables called fluents;  $\mathcal{A}$  is a set of instantaneous ( $a_{it}$ ) and durative ( $a_{id}$ ) physical actions with  $a_{it} = a_{ic}$ ;  $\Delta$  is a set of durative sensing actions, where  $\delta \in \Delta$  is a sensing action defined as  $\langle \delta_{pre}, \delta_{eff}, \delta_{dur} \rangle$  with  $\delta_{pre}$  the preconditions required for sensing action  $\delta$  being executable,  $\delta_{eff}$  defines the sensing action effects, where a literal  $l$  (a proposition  $p$ ) in the set  $\delta_{eff} : \mathcal{L}$  reveals the truth value of the unknown atomic proposition  $p \in \mathcal{P}$  at the end of the action, and  $\delta_{dur}$  parameter representing a set of duration constraints (controllable and known),  $\delta$  holds the same action precondition and effect types than a durative action  $a_{dt}$ ;  $\mathcal{I}_T$  is the set of clauses over  $\mathcal{P} \cup \mathcal{F}$  that denotes the initial state;  $\mathcal{G}_T$  is a set of literals over  $\mathcal{P} \cup \mathcal{F}$  representing the goals; and  $\mathcal{T}$  is a set of time windows. Each time window is defined using timed initial literals (TILs).

**State Model for  $\mathcal{P}_{TC}$ .** The  $\mathcal{P}_{TC}$  defines the state model  $\mathbf{S}(\mathcal{P}_{TC}) = \langle \mathcal{S}, \mathcal{S}_I, \mathcal{S}_G, \mathcal{M}, \mathcal{O} \rangle$ ; where  $\mathcal{S}$  represents a finite set of states over the propositions and fluents  $\mathcal{P} \cup \mathcal{F}$ ;  $\mathcal{S}_I$  is the set of possible initial states,  $\mathcal{S}_I \subseteq \mathcal{S}$ ;  $\mathcal{S}_G$  is the set of goal states,  $\mathcal{S}_G \subseteq \mathcal{S}$ ;  $\mathcal{M}$  is a set of actions with  $\mathcal{M}(s)$  denoting the actions in  $\mathcal{M}$  ( $\mathcal{A}$  and  $\Delta$ ) that are applicable in the state  $s$  (actions with preconditions true in  $s$ );  $\mathcal{O}$  is a set of observation tokens. An action  $a$  or  $\delta$  applicable in a state  $s$  changes the state to  $s' = f(a, t, s)$  at time  $t$  and results in an observation token  $o(s', t, a) \in \mathcal{O}$ .  $f(a, t, s)$  is the state-transition function determined by the conditional effects associated with physical and sensing actions in  $\mathcal{M}$  and the time  $t$  the conditional effects occur; and  $o(s, t, a)$  is  $\top$  for  $a \in \mathcal{A}$ , and  $o(s, t, \delta)$  is  $\top$  or  $\perp$  according to the truth value of proposition  $p$  in state  $s$  for  $\delta \in \Delta$ . Executions are sequences of action-time-observation sets  $(m_0, t_{m_0}, o_0) \dots (m_n, t_{m_n}, o_n)$ ,

where  $m \in \mathcal{M}$ . An execution is possible in the model  $\mathbf{S}$  if, starting from the initial belief  $b_0$ , each action  $m_n$  is applicable in the belief  $b_n$  resulting from the execution up to  $m_n$ ; and the minimum time  $t_{m_{n-1}}$  at which an applicable action  $m_{n-1}$  can start is  $t_{m_{n-1}} = t_{m_{n-2}} + d_{m_{n-2}}$ , where  $m_{n-2}$  is a parent action,  $m_{n-1}$  is a child action,  $t_{m_{n-2}}$  is  $m_{n-2}$  starting time, and  $d_{m_{n-2}}$  its duration. A tree policy  $\pi$  is a function mapping executions into actions. The executions induced by a tree policy  $\pi$  are the possible executions in which  $a_i$  is the action dictated by the policy given the the execution up to  $a_i$ . A policy solves the model if all such executions reach a goal belief state, i.e., a belief state  $b \subseteq \mathcal{S}_G$ . The solution to this planning problem requires a solver capable of solving a time-knowledge aware plan  $\Pi_{TC}$ . A plan solution for the temporally-contingent planning problem is shown in Figure 2c. The branched plan shows the solution deals with the domain incompleteness. In addition,  $\Pi_{TC}$  reasons about the temporal requirements allowing the scheduling of actions in the plan with numeric and temporal constraints. The state model  $\mathbf{S}(\mathcal{P}_{TC})$  handles the deterministic temporally-contingent planning.

## 4 Problem Translation and Planning

This section describes the methodology for translating temporally contingent planning problems  $\mathcal{P}_{TC}$  to temporal problems  $\mathcal{P}_T$  that can be solved for a temporal planner. This strategy takes as initial reference the work implemented in (Palacios, Albore, and Geffner 2014) where authors propose two alternative translations of contingent into classical problems. Our approach introduces a set of changes regarding the second translation presented in mentioned work. First, we insert the temporal notions into the contingent problem. Second, we remove the axioms from the translation, which increases the number of fluents and actions. Finally, the

sensing action is the only source of uncertainty in the original temporally-contingent planning problem, similarly to the problems analysed by Muise, Belle, and McIlraith (2014). The sensing action reveals the truth value of a proposition while introducing a set of other conditional effects. For instance, the effect of executing action `(inspect-area auv camera t1)` can be  $\neg(\text{bla.obstructed } t1)$  (the BLA is clean) and `(available auv)` (the AUV is ready to execute the next action). The approach accounts for the actions in one specific traversal of the policy tree, building a temporal solution for each conditional branch. An example of a specific traversal is the temporal plan in Figure 2a where the BLA is assumed to require biofouling cleaning. We use the *stack* element defined by Palacios, Albore, and Geffner (2014), which pushes the states that predict  $p$  to be false to be analysed after finishing with all true  $p$ , where  $p \in \mathcal{P}$ . A stack size of  $\mathcal{H}$  will suffice to obtain temporally-contingent plans with branches that accommodate up to  $\mathcal{H}$  observations. The  $\mathcal{H}$  value should be small considering the size of contingent plans represented by trees is exponential in the maximum number of observations. However, the source of uncertainty in our problem is limited to sensing action which substantially reduce the translation complexity.

**Definition 7.** The translation  $T(\mathcal{P}_{TC})$  of  $\mathcal{P}_{TC}$ , where  $\mathcal{P}_{TC} := \langle \mathcal{P}, \mathcal{FA}, \Delta, \mathcal{I}_T, \mathcal{G}_T, \mathcal{T} \rangle$  and the stack parameter  $\mathcal{H} > 0$  is the temporal planning problem with subproblems (or instances)  $T(\mathcal{P}_{TC}) = \langle \mathcal{P}', \mathcal{F}', \mathcal{A}', \Delta', \mathcal{I}'_T, \mathcal{G}'_T, \mathcal{T}' \rangle$  supporting the translation, where

- $\mathcal{P}' = \{ \mathcal{L}/s, m(s), lev(l), stack(s, l) : \mathcal{L} \in \mathcal{P}, s \in b_{\mathcal{I}_T}, l \in [0, \mathcal{H}] \}$ ,
- $\mathcal{F}' = \mathcal{F}$ ,
- $\mathcal{I}'_T = \{ \mathcal{L}/s, m(s), lev(0) : \mathcal{L} \in \mathcal{P}, s \in b_{\mathcal{I}_T}, s \models \mathcal{L} \}$ ,
- $\mathcal{G}'_T = \mathcal{G}_T$ ,
- $\mathcal{T}' = \mathcal{T}$ ,
- **Actions:**
  - $\mathcal{A}' = \mathcal{A} \cup \{ a(p, l, t) : a(p, t) \in \delta, l \in [0, \mathcal{H}] \}$ , *preconditions*  $\mathcal{L}$  of  $a \in \mathcal{A}$  replaced by  $\mathcal{XL}$  and  $\neg \mathcal{XG}$ ; *effects*  $a : \mathcal{C} \rightarrow \mathcal{E}$  replaced by  $a : \mathcal{C}/s, m(s) \rightarrow \mathcal{E}/s$  for each  $s \in b_{\mathcal{I}_T}$ , where  $\mathcal{C}$  and  $\mathcal{E}$  are sets (conjunction) of literals,  $a_{it_{pre}}$  and  $a_{dt_{pre}}$  are enclosed in  $\mathcal{C}$ , and  $a_{it_{eff}}$  and  $a_{dt_{eff}}$  are enclosed in  $\mathcal{E}$ ,
  - $\Delta' = \Delta \cup \{ \delta(p, l, t) : \delta(p, t) \in \Delta, l \in [0, \mathcal{H}] \}$ , *preconditions*  $\mathcal{L}$  of  $\delta(p) \in \Delta$  become *preconditions*  $\mathcal{XL}$  for  $\delta(p, l, t)$  in addition to  $lev(l)$ ,  $\neg \mathcal{XG}$ ,  $\neg \mathcal{Xp}$ ,  $\neg \mathcal{X}\neg p$ ; *effects* of  $\delta(p, l, t)$  are  $\neg lev(l)$ ,  $lev(l+1)$ , and *conditional effect*  $m(s)$ ,  $\neg p/s \rightarrow stack(s, l+1)$ ,  $\neg m(s)$ ,
  - **Pop actions** *preconditions* of  $pop(l)$  are  $lev(l)$  and  $\mathcal{XG}$ ; *effects* are  $\neg lev(l)$ ,  $lev(l-1)$  and *conditional effects*  $m(s) \rightarrow \neg m(s)$  and  $stack(s, l) \rightarrow \neg stack(s, l)$  for each  $s \in b_{\mathcal{I}_T}$

The action following a sensing actions  $\delta(p)$  where  $p$  is true is considered first and the ordering is decided using  $\mathcal{H}$ . The translation  $T(\mathcal{P}_{TC})$  contains propositions  $\mathcal{L}/s$  for the literals  $\mathcal{L}$  in  $\mathcal{P}_{TC}$  and the possible initial states  $s \in b_{\mathcal{I}_T}$  where  $b_{\mathcal{I}_T} = \mathcal{S}_0$ . Literals in  $\mathcal{P}_{TC}$  have the form  $p$  and  $\neg p$  for  $p \in \mathcal{P}$ . The literals  $\mathcal{L}/s$  represent that  $\mathcal{L}$  is true under

```
(:unknown-prop
  (state_on v1) (state_on v2)
  (bla.obstructed t1) (bla.obstructed t2) (...))
)
```

Figure 3: Construct `unknown-prop` defining the unknown propositions in the  $\mathcal{P}_{TC}$ .

```
(:knowledge-updates
  (oneof (state_on v1)
    (and (not (state_on v1)) (valve_closed wp32)))
  (oneof ... ))
)
```

Figure 4: Construct `knowledge-updates` defining the possible outcomes associated to the unknown proposition.

the assumption that  $s$  is the true hidden initial state. Propositions  $m(s)$  are responsible for tracking of the set of possible initial states  $s$  at any time.  $\mathcal{XL}$  defines preconditions where the context given by the hidden state  $s$  is implicit. The stack is represented by propositions  $lev(l)$  to indicate the top of the stack,  $stack(s, l)$  to indicate that the hidden state  $s$  has been pushed onto the stack at level  $l$ , and static propositions  $next(l, l+1)$  that represent that one level follows the other. We assume, static propositions are true for each  $l \in [0, \mathcal{H}]$  where  $\mathcal{H}+1$  is the max stack level and  $\mathcal{G}$  is a single proposition.

The number of actions in the translation is  $\mathcal{O}(|\mathcal{A}| + \mathcal{H} \cdot |\Delta|)$  where  $\mathcal{H}+1$  is the stack size, the number of propositions is  $\mathcal{O}((|\mathcal{P}| + \mathcal{H}) \cdot |b_{\mathcal{I}_T}|)$ , while the maximum number of conditional effects per action is  $\mathcal{O}(|\mathcal{S}| \cdot |\mathcal{P}|)$ . The translation encodes the accessibility relations among worlds by the  $m(s)$  literals that represent the set of hidden initial states that are possible given the current execution. The approach uses a stack where the hidden states  $s$  that predict  $\neg p$  after the execution of the sensing action  $a(p, l, t)$  are stored. The translation makes also sure that  $p$  is not known either true or false by adding the literals  $\neg \mathcal{Xp}$  and  $\neg \mathcal{X}\neg p$  in the action precondition. Finally, goals can be reached in the form of the  $\mathcal{XG}$  literal, so that the executions associated with hidden states that have been pushed onto the stack become current and can be extended to reach the goal as well.

**Encoding the Unknown Propositions.** Hidden literals defining the possible initial states are externally defined and used by the framework to generate the translation and the plan solution. Example of representing unknown information for the Simulation Scenario is presented in Figure 3. The unknown propositions indicate that the `(state_on v1)` and `(state_on v2)` are unknown in the initial state. The generation of the contingent sub-plans (branches) that model the real value of an unknown literal  $l$  at the planning time takes the information from the possible knowledge updates. Figure 4 shows a constructed example that defines the updates associated with the incomplete information that will be true in each branch. The updates are a complex nesting of **and** and **oneof** clauses. Some of the nondeterminism is independent, and others contain dependencies defined by the **ands**.

**Temporal Reasoning.** When translating the temporally-contingent planning problem  $\mathcal{P}_{TC}$  it is required to find a policy that guarantees the validity of the solution for all instances of  $T(\mathcal{P}_{TC})$  that provide the full solution. We assume the durative sensing and physical actions in the tree require the maximum duration to complete. First, we identify a seed plan by considering there are not states that predict  $p$  to be false modelling the plan's actions. Resulting from the translation we know the points in the seed plan where  $p$  becomes false and we use this to insert contingency branches associated to particular  $T(\mathcal{P}_{TC})$  instances. The system adds ordering constraints. An action  $a_q$  is applicable in a plan if its effects do not delete the propositional invariants of any previous actions in the course to be completed at the time  $a_q$  ended. Following this dynamic, each action included in the partial plan adds a new set of propositions that lead to a new state. This analysis consider building a sequential temporal plan for each possible  $T(\mathcal{P}_{TC})$  instance. We ensure a temporal instance  $\Gamma$  is sequential if there exists a sequential plan  $\pi_n$  solving  $\Gamma$ . Our approach solves all instances translated from  $\mathcal{P}_{TC}$  as sequential temporal instances considering the computation of the plan solution and its scheduling. The scheduling considers  $t_{b1} = t_{seed_{SA}}$ , where  $t_{seed_{SA}}$  is the time in the seed branch where the hidden  $l$  is revealed, and  $t_{b1}$  is the branch starting time. The action order depends on the time propositions hold in the seed plan and in the sub-plans (branching plans) section before the current state. We do not allow an action to move (or being scheduled) before a branch starts (assuming it is needed in that branch and it is not part of the seed plan) to avoid the movement interfere with other branches. Our system can just move actions earlier in a plan if there is correspondence in the sequence.

**Definition 8.** *There is correspondence between plan steps if step PS-1 is possible after plan step PS-0 in at least one topological sort of the plan graph. A topological sort of a plan's graph is a linear ordering of the graph's steps that respect the temporal constraints denoted by the graph's causal links, conditioning links, and the ordering constraints.*

**Plan Generation.** Algorithm 1 generalises our approach which focuses in five steps: (i) take the problem and encode it into a temporal planning problem, (ii) solve the resulting temporal planning problem, (iii) decode the plan into a temporal contingent plan, and (iv) if an uncovered discrepancy is found, refine model/initial state. The unknown knowledge  $\mathcal{UK}$  associated with individual propositions are processed (line 2) to extract all true values following the idea mentioned when the stack parameter was introduced in this section. This information is used alongside the initial state  $\mathcal{S}_T$  to generate a plan (line 3). This plan contains then all sensing actions that potentially lead to the creating of additional branches. Then the structure of the plan is represented as a tree  $(\mathcal{Q}, \mathcal{U})$  (line 4). Each  $q \in \mathcal{Q}$  is an action in the plan with  $q^0$  the root of the tree that corresponds to the first plan step and  $(e, b) \in \mathcal{U}$ . The algorithm evaluates the sensing action in the plan in an ordered manner (line 6) to build all additional branches. The state is the query for a different outcome of the sensing action to construct the extensions or branches (line 7-8). The Build-

---

**Algorithm 1: Planning Compiler**


---

**Input:**  $\mathcal{S}_T$ : Problem Initial State.  
**Input:**  $\mathcal{UK}$ : Unknown Knowledge  
**Output:**  $\Pi_{tc}$ : Branched Plan.

```

1 begin
2    $\mathcal{Y} \leftarrow \text{ExtractTrueUnknownKnowledge}(\mathcal{UK})$ 
3    $\Pi_{tc} \leftarrow \text{GeneratePlan}(\mathcal{Y}, \mathcal{S}_T)$ 
4    $(\mathcal{Q}, \mathcal{U}) \leftarrow \text{tree for } \Pi_{tc}$ 
5    $\mathcal{S}' \leftarrow \mathcal{S}$ 
6   for each ordered  $\delta \in \Pi_{tc}$  do
7      $\mathcal{S}' \leftarrow \delta . \text{apply}(\mathcal{S}')$ 
8      $(\mathcal{Q}', \mathcal{U}') \leftarrow \text{BuildBranch}(\mathcal{S}')$ 
9     if  $\mathcal{Q}' \neq \emptyset$  then
10       $\mathcal{F} \leftarrow \text{CheckUnknownKnowledge}(\mathcal{UK})$ 
11       $e \leftarrow \text{Root}(\mathcal{Q}', \mathcal{F})$ 
12       $b' \leftarrow b + 1$ 
13      while  $e' = b'$  do
14         $b' \leftarrow \text{Increment}(b', 1)$ 
15         $e \leftarrow \text{Increment}(e, 1)$ 
16       $e \leftarrow \text{Root}(\mathcal{Q}, \mathcal{U}, \mathcal{F})$ 
17       $e \leftarrow \text{AddSubtree}(\mathcal{Q}', \mathcal{U}')$ 
18       $\mathcal{Q} \leftarrow \text{Add}(b' - 1, e)$ 
19 return  $\Pi_{tc}$ 

```

---

Branch function in our method is associated with evaluating the current state for a different possible outcome of the unknown proposition. If this return is non-empty (line 9), the algorithm keeps evaluating the branch until reaching the goal state (line 13-14). During branch evaluation, the tree is rooted (line 15). Considering it might be the case, we found additional branches in the actual subbranch we are expanding (line 16). The algorithm keeps iteratively working on expanding all subtrees until no further sensing actions require expansion. Then Planning Compiler returns the temporally-contingent plan solution  $\Pi_{tc}$  (line 18).

**Lemma 1. Correctness.** *We adopt a notion of correctness considering: at run-time, the planner must have all necessary knowledge at every step for plan's execution, and if the solver returns a plan  $\Pi_{TC}$ , the plan solve the temporally-contingent planning problem. Following this the approach returns a plan just when all sensing actions have been totally expanded. The introduction of the check on  $\mathcal{Q}$  ensures we explore all possible branches and the solution only exists if the approach successfully evaluates all nodes in the policy tree. If in the policy tree no nodes precedes its parents and children nodes associated with positive observations are come first. Let  $\pi(n_N)$  represent the action performed by the policy at time  $t$  in node  $n_N$ ,  $\mathcal{V}(n_N)$  represents the set of hidden states in the initial state, compatible with the execution up to  $n_N$ , and  $\text{lev}(n_N)$  the level of the node in the tree. If the levels are not greater than  $\mathcal{H} + 1$ , and the time  $t_{n_{N+1}}$  at which node  $n_{N+1}$  starts is always  $t_{n_N} + d_{n_N}$ , where  $d_{n_N}$  represents the duration of executing node  $n_N$  then the sequence of actions  $\pi'(n_0), \dots, \pi'(n_N), \pi'(n_{N+1})$  is a temporal plan for  $T(\mathcal{P}_{TC})$ .*

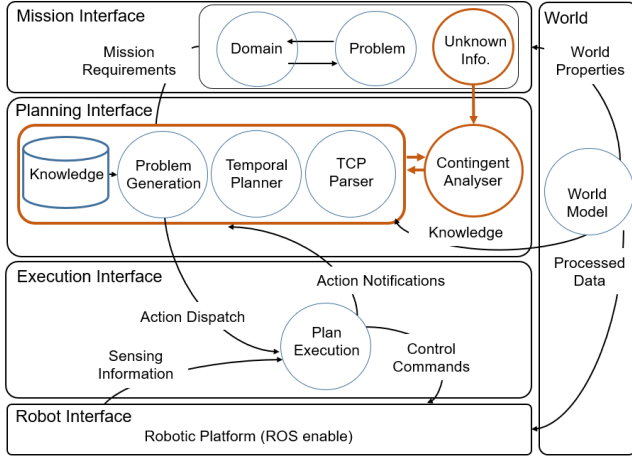


Figure 5: System Architecture for high-level task planning in the AUV. The Plan Dispatcher takes information from the Sensor Interface to determine the dispatch by considering sub-plans obtained in the (offline) planning phase. The Planning Interface connects the Contingent Analyser with a Goal-Based Temporal Planner to generate a branching solution that respond to all possible outcomes of the unknown properties which are listed as inputs to the system.

**Lemma 2. Soundness.** Let  $r_0, \dots, r_k$  be a temporal plan for  $\mathcal{P}_{TC}$  such that  $m_i(s)$  represents the status of the  $m(s)$  fluents when the action  $r_N$  is applied at time  $t_N$ . Let the sequence of nodes  $n_0, \dots, n_{k-1}, n_k$ , where parent node  $n_N$  at time  $t_N$  always come first than child node  $n_{N+1}$  at time  $t_{N+1}$  such that  $t_N < t_N + d_N < t_{N+1}$  and  $d_N$  represents the duration of executing the node  $n_N$ . There is a state  $s$  such that both  $m_{n_N}(s)$  and  $m_{n_{N+1}}(s)$  are true and there is no  $k$ ,  $N < k < N + 1$ , such that  $m_k(s)$  is true as well. In such a case, the edge from  $n_N$  to  $n_{N+1}$  is  $\top$  for  $\delta \in \Delta$  and  $a \in \mathcal{A}$  and  $N + 1 > N$ . Else, the edge from  $n_N$  to  $n_{N+1}$  is  $\perp$ . Finally, the policy  $\pi(n_N)$  over the nodes of the tree solves  $\mathcal{P}_{TC}$ .

## 5 System Architecture

In this section, we describe the main elements of the AUV system architecture that supports the execution of temporally-contingent plans. The set of possible sub-plans (branches) is generated offline using the strategy defined in Section 4. The execution of a particular branch is decided online based on the information gathered. Figure 5 shows the system architecture. The system contains four modules that are interconnected at different levels with the World: (i) Mission Interface, (ii) Planning Interface, (iii) Execution Interface, and (iv) Robot Interface.

**Mission Interface.** Includes the PDDL domain and problem that describes the initial state  $\mathcal{S}_I$  requirements. In addition, this component embeds the Unknown Info., which encapsulates all the unknown domain information required to build the contingent branches. For instance, Unknown Info. contains the possible states of the valves and the BLA in

the Biofouling Cleaning domain (e.g., `(state_on v1)` and `¬(state_on v1)`, `(bla_obstructed t1)`, etc.)

**Planning Interface.** Includes all the available knowledge at the planning time to generate the problem and plan solution. For this work, a Contingent Analyser takes the information available in the Unknown Info. component to implement the compilation of temporal planners into a contingent planning structure that allows the obtain a temporally-contingent plan solution. The Contingent Analyser interacts with the Problem Generator and a temporal planner (we use OPTIC (Benton, Coles, and Coles 2012)) to generate the branches that respond to all possible values of the unknown propositions. We built a parser to evaluate the plan solution before starting the action dispatch that connects the Planning Interface directly with the Execution Interface.

**Execution Interface.** Takes the dispatched action from the Planning Interface and translates it to action commands understandable for the AUV. The Execution Interface acts as a bridge providing the Sensing Information used to determine the actual value of the incomplete information at the execution time. This component checks the sensing action's implementation closely to provide feedback to the Action Dispatcher and decide the following action to execute.

**Robot Interface.** Includes the robotic platforms we can use in the mission. This interface provides all necessary data to evaluate the mission's execution. This architecture extends the actual ROSPlan (Cashmore et al. 2015) Action Interface introducing a set of features to support the implementation of conditional plans with durative actions. The planner produces plans using a domain model and a problem which are inputs to the Knowledge Base (KB). In case of failures, the system is able to react and replan considering the unfinished goals. This architecture allows to integrate our system in multiple robotic platforms. We implement the approach in simulation using a RexROV2 (see Figure 6a), and in a real environment using BlueROV2 (see Figure 6b).

## 6 Domain and Problem Definition

This section describes the main properties of two underwater domains<sup>1</sup> where the AUVs RexROV2 and BlueROV2 have to complete missions in the presence of incomplete information, numeric and temporal constraints. As a running example, we consider an AUV which is used to implement multiple tasks in the underwater domain such as: (i) seabed mapping, (ii) structure reconstruction, (iii) inspection of valves, (iv) manipulation of their handles, and (v) cleaning of the biofouling at the BLA. Figure 6a shows the simulation scenario and Figure 6b shows a real environment.

**Domain Description (Simulation Scenario).** An offshore scenario called Biofouling Cleaning (Figure 6a) includes a set of blowout preventers (BOPs), structures with a valve attached that can be open or closed (the valve state is unknown at planning time). In addition, the environment presents multiple wind turbines which require regular inspection of their

<sup>1</sup>In <https://github.com/YanielCarreno/tcp-domains.git> you can find the domains and problems we analyse in this paper.



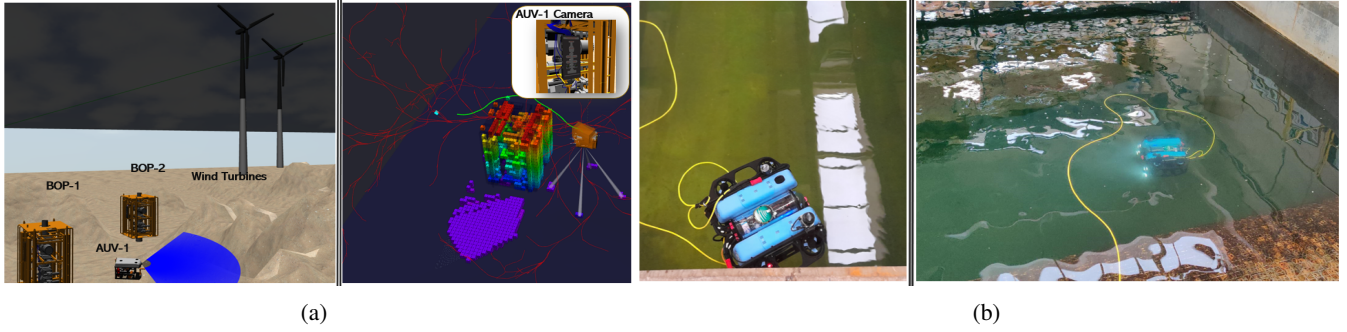


Figure 6: In (a) the Biofouling Cleaning domain with multiple turbines that require inspection. The AUV-1 generates a plan solution to solve mission goals while keeping battery levels at adequate levels. In (b) the BlueROV2 operates as an AUV implementing inspection and manipulation missions with incomplete information of the initial state.

bases and the execution of maintenance around the BLA that is usually affected by the biofouling effects. The structure's coordinates are known, and the AUV (RexROV2) does not have any initial knowledge about the seabed characteristics. This scenario allows the implementation of missions that cover all types of tasks described in this section. The sequence of actions the AUV should implement depends on the sensing action outcomes. This domain was partially described in Example 1. Therefore, all mission requirements defined in Example 1 holds in this domain. The only addition to this domain concerning the example is the valves (v1 and v2) inspection, which must be closed at the mission end.

In this example, the action `sense-valve` adds knowledge related to possible valve states. The action `close-valve` presents the precondition (`valve.state ?v state_on`). Therefore, the run-time plan execution determines if the valves requires to be manipulated if during plan execution the AUV identifies `state_on` the branch to choose should have the `close-valve` action. This domain includes temporal constraints to support recharging. In addition, the domain introduces numeric constraints associated with data communication. Action `sense-valve` has an effect that increases (`data_acquired ?r - robot`) if the valve is open. In addition, the action `sense-valve` is conditioned by the robot's data capacity. This constraint makes the robot navigate to the surface and communicate data, using action `broadcast-data`, before executing a new sensing action if data was previously acquired. Therefore, the  $\neg$ `state_on` plan solution presents a completely different sequence of actions.

Temporal constraints are essential for scheduling recharging activities due to the docking point availability. There is no sequence of actions that allows the AUV to achieve the goal without knowledge of the valve states: choosing the correct action to execute after sensing the valve state depends on the (run-time) result of whether the valve is open or closed. The characteristics of this problem where the solution requires temporal and numeric constraints and incomplete sensing information make it a temporally-contingent planning problem. Figure 7 shows a fragment of the plan for the Biofouling Cleaning problem of closing two valves. The plan presents a set of branches that lead to different ac-

Time:	(Action Name)	[Duration]
0.00:	(navigation auv base v1)	[100.00]
100.01:	(sense-valve auv cameral v1)	[30.00]
	<BRANCH, 1, true, (state_on v1)>	
130.02:	(close-valve auv v1)	[50.00]
180.03:	(navigation auv v1 surfc.3)	[67.00]
247.04:	(recharge-battery auv surfc.3)	[43.80]
290.85:	(broadcast-data auv surfc.3)	[10.00]
300.86:	(navigation auv surfc.3 v2)	[70.10]
370.97:	(sense-valve auv cameral v2)	[30.00]
	<BRANCH, 2, true, (state_on v2)>	
400.98:	(close-valve auv v2)	[50.00]
450.01:	(navigation auv v2 base)	[160.00]
610.02:	(broadcast-data auv base)	[10.00]
620.03:	(recover-robot auv base)	[1.00]
	<BRANCH, 2, false, (state_on v2)>	
400.98:	(navigation auv v2 base)	[260.00]
606.99:	(recover-robot auv base)	[1.00]
	<BRANCH, 1, false, (state_on v1)>	
130.02:	(navigation auv v1 v2)	[280.00]
410.03:	(sense-valve auv cameral v2)	[30.00]
	<BRANCH, 2, true, (state_on v2)>	
440.04:	(close-valve auv v2)	[50.00]
490.05:	(navigation auv v2 surfc.5)	[140.00]
630.06:	(recharge-battery auv surfc.5)	[42.00]
672.07:	(broadcast-data auv surf.5)	[10.00]
682.08:	(navigation auv surf.5 base)	[320.00]
1002.09:	(recover-robot auv base)	[1.00]

Figure 7: A fragment of a temporally-contingent plan for the Biofouling Cleaning domain where the AUV needs to close two valves. The plan considers action start times and durations. The sensing action `sense-valve` leads to conditional plan branches that consider the set of possible outcomes.

tion sequences depending on the output of the sensing action. In addition, the recharging action is implemented at other times. Another mission goal is to implement the maintenance of the BLA at one of the wind farm turbines. The actions to implement this goal are considered in Example 1. The AUV should use its sensory system to evaluate the BLA's state. If the condition is optimal (non-obstructed), the robot already completes the goal. On the other hand, if the

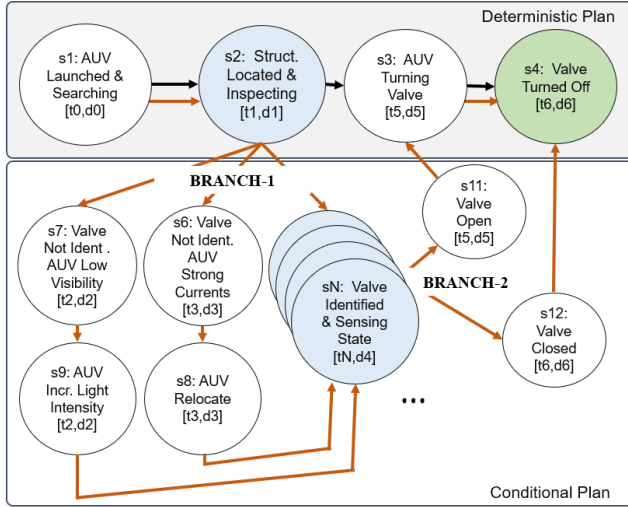


Figure 8: A layered temporally-contingent plan modelling an AUV in a valve turning mission, with simplified states.

area is dirty, the AUV should implement cleaning labours to leave the area ready for the boats. The implementation of the cleaning action consumes a substantial amount of energy which might force the AUV to go for recharging with high frequency if more goals are allocated.

**Domain Description (Real Scenario):** A real underwater scenario called Underwater Structure (Figure 6b) was built in the Ocean Systems Laboratory at Heriot-Watt University. The domain presents two structures, the AUV (BlueROV2) knows the structures’ positions but lacks knowledge about the remaining features that define the environment (e.g., structure’s shapes, floor irregularities, obstacles, etc.). This scenario is restricted to missions associated with structure mapping and exploration. Our AUV is equipped with electrical manipulators, stereo cameras and sonar.

Figure 8 shows a branch representation of a possible plan solution where the Underwater Structure domain formalises a single valve operation and introduces possible disturbances in the model to be consider at the planning time. The normal operation of the system (Initial-Approach) using a temporal planner does not review the effect of the disturbances (while attempting to identify and underwater valve embedded in one of the structures) and consider the state of the valve known ( $S_1 - S_2 - S_3 - S_4$ ). The introduction of the contingency elements into the system supports a more realistic model of the environment which lead to multiple branches. Focusing on  $S_2$  the plan can introduce a set of possible unsafe modes related to the non identification of the valve. Most importantly, the plan can provide alternative solution to these problems in advance. Therefore, if the system faces these problems at the execution time recovery mechanism are already enclosed in the original plan.

## 7 Experiments and Results

Our domain and problems are encoded in PDDL. All experiments in this section are run on Ubuntu 16.04, with an

Intel Core i7-8700, limiting the planner to 30 min (minutes) of CPU@3.2GHz, 16GB of RAM, stack size  $\mathcal{H} = 10$ . The experiments focus on the domains described in Section 6 and evaluate the Temporally-Contingent Planning (TCP) approach. The TCP is defined as the combination of temporal planner and the contingent analyser that allows the generation of a temporally-contingent plan solution.

**Experiment 1 (Offline Planning).** This experiment evaluates the quality of the planning algorithm while generating plans for 10 problem instances for the domains in Section 6. Table 1 shows the results of the evaluation. In the first domain, the planner obtains solvable solutions for most problems that reflect different number of assets (A) of types turbines or valves and battery levels (B-L). The approach does not generate a solution for the last two problems considering the planning time limitations. A similar situation occurs in the real scenario when the maximum number of mapping failures associated with valve identifications is 10. The main reason for *time out*—(TO) time limitation for planning—is a large number of branches the system needs to create online to cope with all possible states of the unknown properties (e.g., valve’s location and valve’s state).

Results show the maximum makespan assuming the robot takes the most extended branch (seed branch). We have noticed the planning time is considerably affected by the introduction of time slots for recharging. For the second domain, the plan solvability rate is also successful. We notice the makespan is significantly smaller for this domain. Failures associated with the non-identification of the valve do not increase mission makespan. This is a consequence of the fact that the actions related to fixing the problems are primarily static and require a short period. Planning time is short, which is relevant since we aim to increase the complexity and the size of the experiments as future work. This experiment attempts to evaluate the robustness of the system. Therefore, we extend the planning times for more extended periods than we should have when implementing missions considering the dynamic of the underwater domain.

**Experiment 2 (Biofouling Cleaning Plan Execution).** This experiment evaluates the mission failure rate (FR), planning execution times (PET), and replanning times (RT) when executing missions over long-term periods. Experiment 2 considers the execution of the Biofouling Cleaning domain (Simulation Scenario) problems in Experiment 1 and presents the number of actions executed overall plans generated to solve each problem for the benchmark solver ( $A^*$ ) and the TCP ( $A^{**}$ ) approach. Table 2 shows our approach outperforms the benchmark planner in most of the problems considering our system generates a plan that deals with uncertainty levels not considered by the baseline system. In addition, our method can reduce the risks of plan unsolvability that appears due to the time constraints or limitations in the resources the domain introduces. For instance, the robot needs to communicate data every time the BLA is cleaned. This can bring problems when the system takes the current state for replanning as the data communication became a hard goal. Its execution receives a high priority, affecting the initial plan the AUV was implementing. In this

	Biofouling Cleaning				Underwater Structure			
Prob.	A	B-L	PT	M	F	B-L	PT	M
1	2	60	1.3	701.2	1	50	1.2	43.4
2	4	60	1.2	834.7	2	60	1.6	51.9
3	5	90	1.3	870.2	2	70	1.1	54.2
4	7	80	2.0	1650.1	3	80	3.5	68.4
5	8	50	5.6	2050.3	4	90	2.6	67.5
6	8	90	4.2	1927.7	4	60	3.1	102.1
7	10	100	2.3	2872.8	6	90	1.4	177.0
8	10	70	4.7	3543.6	8	90	3.1	182.0
9	10	50	TO	TO	10	60	TO	TO
10	15	70	TO	TO	10	90	TO	TO

Table 1: Experiment 1: Results for planning time (PT) in min and makespan (M) in min for multiple problem instances of the (i) Biofouling Cleaning domain and the (ii) Underwater Structure domain for different sets of assets (A) valves or turbines battery levels (B-L) in %, and number of failures (F). Time out (TO) while finding a solution.

		Benchmark Planner			TCP Approach		
Prob.	A*/A**	FR	PET	RT	FR	PET	RT
1	52/35	0.1	988.2	8.5	0.0	691.2	–
2	84/53	0.2	993.6	15.9	0.0	802.5	–
3	72/51	0.2	780.2	–	0.0	772.1	–
4	92/73	0.3	1756.8	139.4	0.0	1212.3	–
5	103/84	0.3	<b>2073.3</b>	37.1	0.1	2168.3	28.7
6	145/102	0.3	1909.8	12.4	0.0	1819.5	–
7	118/87	0.3	2956.1	92.3	0.0	2637.4	–
8	268/113	0.3	3598.2	127.3	0.0	3200.3	–
9	312/–	0.4	4109.2	161.9	TO	TO	TO
10	532/–	0.5	6230.1	235.1	TO	TO	TO

Table 2: Experiment 2: Failure Rate (FR) in %, Planning Execution Times (PT) and Replanning Times (RT) in min when evaluating a benchmark temporal planner and our Temporarily Contingent Planning (TCP) approach is a set of 10 problems with different levels of complexity for the Biofouling Cleaning domain. Time out (TO) while finding a solution.

type of settings, missions dealing with dynamic changes using uncertainty have a high risk of failure. Finally, the benchmark planner needs to replan multiple time in all problems which increases the number of actions (A\*) required to solve the whole mission respect to our approach (A\*\*).

### Experiment 3 (Underwater Structure Plan Execution).

This experiment focuses on execution, and we evaluate the system’s performance in a laboratory environment using BlueROV2. Figure 9 analyses the mission execution time for 10 different problems with the introduction of 5 forced failures during the mission, associated with the valve state and valve’s localisation. The generation of contingent branches during the planning stage significantly improves mission implementation times. This is mainly due to the time required by the temporal planner (OPTIC) to replan to achieve a plan that responds to the valve’s actual state or identification. Our

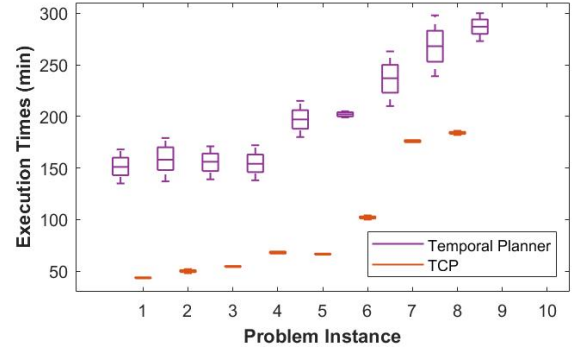


Figure 9: Experiment 3: Plan execution time in 10 problem instances of the Underwater Structure domain over 10 runs. Results show our approach solves problems with uncertainty in the  $\mathcal{S}_T$  and temporal constraints.

approach reduces replanning in non-quiet environments since the algorithm can consider contingent effects at planning time. In addition, it deals with temporal and numeric constraints. However, contingent plans suffer from small-time variations associated with the delays resulting from the system choosing the contingent branch to follow. We typically have conditional plans to handle problems that have a high probability of occurring. This is the case of the domains we presented in this work. Replanning shows promising result to maintain the mission’s survivability. However, it can introduce unnecessary delays in the mission, mainly when we can use alternative algorithms that deal with certain levels of uncertainty in the domain.

## 8 Conclusions

We present a general approach to task planning based on the combination of temporal planning, contingent planning and run-time sensing. Our approach focuses on the translation of temporally-contingent planning problems into temporal problems. This idea enlarges the range of problems that temporal planners can solve. We introduce an alternative solution that solves temporally-contingent planning problems using offline temporal planning, which reduces the risks associated with replanning in non-quiet environments. We demonstrate the effectiveness of our method of dealing with AUV missions in simulated and real scenarios. We propose a new type of domain environment that require solvers to consider temporal reasoning, such as (i) timed initial literals and deadlines and (ii) manage resources using numerical fluents. Future work aims to extend the approach to more challenging scenarios when the contingent planning reasoning requires considering other robots. In addition, we plan to extend the system to deal with more significant size problems where the source of uncertainty can be associated with exogenous events.

## Acknowledgements

This work was funded and supported by the ORCA Hub (orcahub.org), under EPSRC grant EP/R026173/1.

## References

- Albore, A.; Palacios, H.; and Geffner, H. 2009. A translation-based approach to contingent planning. In *IJCAI*.
- Benton, J.; Coles, A. J.; and Coles, A. 2012. Temporal Planning with Preferences and Time-Dependent Continuous Costs. In *ICAPS*.
- Bonet, B.; and Geffner, H. 2014. Belief tracking for planning with sensing: Width, complexity and approximations. *JAIR* 50: 923–970.
- Brafman, R. I.; and Shani, G. 2012. A Multi-Path Compilation Approach to Contingent Planning. In *AAAI*. Citeseer.
- Bukasz, D.; Cashmore, M.; Krarup, B.; Magazzeni, D.; and Ridder, B. 2018. Strategic-tactical planning for autonomous underwater vehicles over long horizons. In *2018 IEEE/RSJ IROS*, 3565–3572. IEEE.
- Camacho, A.; Muise, C. J.; and McIlraith, S. A. 2016. From FOND to Robust Probabilistic Planning: Computing Compact Policies that Bypass Avoidable Deadends. In *ICAPS*, 65–69.
- Carreno, Y.; Scharff Willners, J.; Petillot, Y. R.; and Petrick, R. P. A. 2021. Situation-Aware Task Planning for Robust AUV Exploration in Extreme Environments. In *Proceedings of the IJCAI Workshop on Robust and Reliable Autonomy in the Wild (R2AW)*.
- Cashmore, M.; Fox, M.; Larkworthy, T.; Long, D.; and Magazzeni, D. 2014. AUV mission control via temporal planning. In *ICRA*, 6535–6541.
- Cashmore, M.; Fox, M.; Long, D.; Magazzeni, D.; and Ridder, B. 2017. Opportunistic planning in autonomous underwater missions. *IEEE Transactions on Automation Science and Engineering* 15(2): 519–530.
- Cashmore, M.; Fox, M.; Long, D.; Magazzeni, D.; Ridder, B.; Carrera, A.; Palomeras, N.; Hurtos, N.; and Carreras, M. 2015. ROSPlan: Planning in the Robot Operating System. In *ICAPS*, volume 25.
- Cimatti, A.; Hunsberger, L.; Micheli, A.; and Roveri, M. 2014. Using timed game automata to synthesize execution strategies for simple temporal networks with uncertainty. In *AAAI*, volume 28.
- Coles, A. J. 2012. Opportunistic Branched Plans to Maximise Utility in the Presence of Resource Uncertainty. In *ECAI*, volume 2012, 252.
- Coles, A. J.; Coles, A.; Fox, M.; and Long, D. 2010. Forward-Chaining Partial-Order Planning. In *ICAPS*.
- Combi, C.; Hunsberger, L.; and Posenato, R. 2013. An algorithm for checking the dynamic controllability of a conditional simple temporal network with uncertainty. *Evaluation* 1(1).
- Combi, C.; Posenato, R.; Viganò, L.; and Zaverri, M. 2019. Conditional simple temporal networks with uncertainty and resources. *JAIR* 64: 931–985.
- Cresswell, S.; and Coddington, A. 2003. Planning with timed literals and deadlines. In *UK PlanSIG*, 23–35.
- Edelkamp, S.; and Hoffmann, J. 2004. PDDL2. 2: The language for the classical part of the 4th international planning competition. Technical report, Technical Report 195, University of Freiburg.
- Foss, J. N.; and Onder, N. 2005. Generating temporally contingent plans. In *IJCAI Workshop on Planning and Learning in A Priori Unknown or Dynamic Domains*.
- Fox, M.; and Long, D. 2003. PDDL2.1: An extension to PDDL for expressing temporal planning domains. *JAIR* 20: 61–124.
- Hashmi, M. A.; and Seghrouchni, A. E. F. 2010. Merging of temporal plans supported by plan repairing. In *2010 22nd IEEE International Conference on Tools with Artificial Intelligence*, volume 2, 87–94. IEEE.
- Hoffmann, J.; and Brafman, R. 2005. Contingent planning via heuristic forward search with implicit belief states. In *ICAPS*.
- Kerschke, P.; Hoos, H. H.; Neumann, F.; and Trautmann, H. 2019. Automated algorithm selection: Survey and perspectives. *Evolutionary computation* 27(1): 3–45.
- Long, D.; and Fox, M. 2003. The 3rd international planning competition: Results and analysis. *JAIR* 20: 1–59.
- McDermott, D.; Ghallab, M.; Howe, A.; Knoblock, C.; Ram, A.; Veloso, M.; Weld, D.; and Wilkins, D. 1998. PDDL – The Planning Domain Definition Language (Version 1.2). Technical Report CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control.
- Muise, C.; Belle, V.; and McIlraith, S. A. 2014. Computing contingent plans via fully observable non-deterministic planning. In *AAAI*.
- Palacios, H.; Albore, A.; and Geffner, H. 2014. Compiling contingent planning into classical planning: New translations and results. In *ICAPS Workshop on Models and Paradigms for Planning under Uncertainty*.
- Peot, M. A.; and Smith, D. E. 1992. Conditional nonlinear planning. In *AIPS*, 189–197. Elsevier.
- Petrick, R. P.; and Bacchus, F. 2002. A Knowledge-Based Approach to Planning with Incomplete Information and Sensing. In *AIPS*, 212–222.
- Thompson, F.; and Guihen, D. 2019. Review of mission planning for autonomous marine vehicle fleets. *Journal of Field Robotics* 36(2): 333–354.
- To, S. T.; Pontelli, E.; and Son, T. C. 2011. On the effectiveness of CNF and DNF representations in contingent planning. In *IJCAI*.
- Tsamardinos, I.; Vidal, T.; and Pollack, M. E. 2003. CTP: A new constraint-based formalism for conditional, temporal planning. *Constraints* 8(4): 365–388.
- Weld, D. S.; Anderson, C. R.; and Smith, D. E. 1998. Extending graphplan to handle uncertainty & sensing actions. In *AAAI*, 897–904.
- Zaverri, M.; and Viganò, L. 2019. Conditional simple temporal networks with uncertainty and decisions. *Theoretical Computer Science* 797: 77–101.

# Combining Task and Motion Planning through Rapidly-exploring Random Trees

\*\*\*

## Abstract

Combined task and motion planning is a relevant issue in robotics<sup>1</sup>. In path and motion planning, Rapidly-exploring Random Trees (RRTs) have been proposed as effective methods to efficiently search high-dimensional spaces. On the other hand, the deployment of these techniques to symbolic task planning problems has been partially investigated. In this paper, we explore this issue proposing a method to combine task and motion planning based on RRTs. Our approach relies on a metric space where both symbolic (task) and sub-symbolic (motion) spaces are represented. The associated notion of distance is then exploited by a RRT-based planner to generate a plan that includes both symbolic actions and obstacle-free trajectories. The proposed method is assessed in several case studies provided by a real-world hospital logistic scenario, where an omni-directional mobile robot is involved in pick-carry-and-place tasks.

## INTRODUCTION

Combining task and motion planning is a crucial issue in robotics. These two planning problems are usually tackled in a separated manner in order to exploit the complementary features of the two planners. While task planners typically work with high-level specifications of the problem, searching for abstract actions, motion planners generate obstacle-free motions in the configuration space taking into account motion constraints. Therefore, the typical approach is to first solve the task planning problem and then deploy the motion planner to find the actual implementation of the abstract plan in the configuration space. However, when the mission is complex and/or the environment is highly constrained (navigation in cluttered environments, manipulation problems, etc.), this decoupling is no longer effective. In these cases, since symbolic and motions constraints can be strictly intertwined, the high-level plan can be easily unfeasible from a motion perspective. In these circumstances, a unified approach to task and motion planning is needed (Ingrand and Ghallab 2017).

Different approaches have been proposed in the literature to address combined task and motion planning (Dantam et al. 2016; Erdem et al. 2011; Lagriffoul et al. 2014; Choi and Amir 2009; Cambon, Alami, and Gravot 2009; Kaelbling and Lozano-Pérez 2011; Barry, Kaelbling, and Lozano-Pérez 2013; Bidot et al. 2017; Thomason and Knepper 2019). Following (Bidot et al. 2017), we can distinguish approaches where: motion planning is primary but guided by task planning (Cambon, Alami, and Gravot 2009; Plaku and Hager 2010; Barry, Kaelbling, and Lozano-Pérez 2013; Thomason and Knepper 2019); task planning is primary and motion planning is selectively invoked (Kaelbling and Lozano-Pérez 2011); task planning and motion planning are interacting processes and a generated task plan is iteratively expanded by the motion planner, which also checks for action executability in the configuration space (Dantam et al. 2016; de Silva, Pandey, and Alami 2013; Erdem et al. 2011).

In this work, we explore a novel approach in which task and motion planning are handled in a uniform manner by a sampling-based planning algorithm. Specifically, we propose and investigate the effectiveness of a method which fully relies on an extension of Rapidly-exploring Random Trees (RRTs).

RRTs are widely exploited algorithms for motion planning in robotics (Lavalle and Kuffner 1999), since they enable an efficient search in non convex, high-dimensional spaces. On the other hand, the deployment of these methods at the task level has been only partially exploited and investigated. RRTs for discrete spaces have been explored by (Morgan and Branicky 2004), but symbolic task planning was not considered. A RRT-based approach to symbolic task planning in STRIPS has been proposed by (Burfoot, Pineau, and Dudek 2006), but the combination of task and motion planning has not been addressed. An embedding of the symbolic state into the continuous space is proposed in (Thomason and Knepper 2019) to enable sampling-based motion planning methods that simultaneously address symbolic, geometric, and kinematic constraints; on the other hand, a task planning heuristic is exploited as a long-horizon symbolic guidance through the search space. In (Plaku and Hager 2010), RRTs are deployed for motion planning in the configuration space, while symbolic task planning is exploited to support motion planning by providing actions and regions of

<sup>1</sup>Notice that this work is currently under review for a robotics conference, therefore, as suggested by the workshop policy, the submission is anonymous to avoid possible conflicts.



the continuous space that the sampling-based motion planner can further explore to advance its search. In this case, symbolic and motion planning are therefore associated with different search strategies and an external black-box symbolic planner is invoked to compute an action plan.

In this paper, we propose and investigate a different approach that directly deploys the basic RRT-based search mechanism into an extended search space that combines the configuration space and the symbolic state space. The aim is to explore the extent to which a direct deployment of a basic RRT sampling-based search is feasible and effective in this extended space. Since the RRT needs a metric space to be sampled, we firstly introduce a distance measure suitable for an extended state space that combines configurations and symbolic states. We then deploy this distance to guide the expansion of the RRT in the extended metric space. Notice that we deliberately deployed a vanilla version of the RRT search (Lavalle and Kuffner 1999) to assess the sampling-based search mechanism, without the support of other optimizations (Kuffner and Lavalle 2000).

In this work, we detail the approach and discuss its feasibility and scalability in real-world robotic case studies provided by a hospital logistic scenario. In this context, we consider incrementally complex pick, carry, and place tasks in cluttered environments. The results collected with the proposed pilot study suggest that the approach is feasible in realistic scenarios.

## RRT-based Task and Motion Planning

In this section, we introduce our Task and Motion RRT planner (TM-RRT) starting from an extended version of a simple RRT planner. Firstly, our planning problem can be defined by the tuple  $(C, S, M, A, q_{init}, s_{init}, s_{goal}, G)$ , where  $C \subseteq \mathbb{R}^n$  is the  $n$ -dimensional configuration space of the robot,  $S$  is the symbolic state space,  $M$  and  $A$  are sets of motions and actions respectively,  $q_{init} \in C$  and  $s_{init} \in S$  are the initial configuration and initial symbolic state, and  $s_{goal} \in S$  is the symbolic goal state. Each state in the symbolic space  $S$  is implicitly represented by means of a finite set of ground predicates in a set  $P$  representing all the possible properties that can hold in a state (with closed-world assumption), i.e.,  $S = 2^P$ .

In this framework, we represent motions and actions. A motion  $m$  is a  $n$ -dimensional vector of movements that linearly drives the robot from a configuration to another. On the other hand, an action  $a \in A$  is a STRIPS-like operator, which is associated with preconditions  $pre(a)$ , add-list and delete-list  $eff(a)^+$ ,  $eff(a)^-$ , specifying the transition between two symbolic states (preconditions and effects are represented by sets of ground predicates in  $P$ ).

Moreover, in order to assess action accomplishment in the configuration space, we introduce a function  $G : A \times S \rightarrow C$  such that, given an action  $a$  applied in the symbolic state  $s$ , it provides a target configuration  $q$  representing the configuration to be reached by the robot in order to finalize  $a$ . This function works as an interface between the symbolic and the configuration space.

In this context, we want to find a sequence of motions and actions that can be executed by the robot in order to reach the

desired symbolic state  $s_{goal}$ , starting from the initial configuration  $q_{init}$  and the initial symbolic state  $s_{init}$ . To this end, we aim at producing a list of action and motion pairs  $\pi = \langle (a_1, m_1), \dots, (a_k, m_k) \rangle$ , where each motion specifies a movement to be performed by the robot in order to accomplish the associated action, while each action labels motions to be executed in order to reach the final goal state. For instance, consider a person that wants to open a bottle of water. At the task level, the action can be decomposed in two actions: *grasp-bottle* ( $a_1$ ) and *open-bottle* ( $a_2$ ). The first one is needed to hold the bottle in position, while the second one permits to remove the cap from it. On the other hand, at the motion level, the first action can be executed by moving the left arm toward the bottle ( $m_1$ ) and closing the left hand to hold it ( $m_2$ ), while the second action execution is composed of a first right hand motion toward the cap ( $m_3$ ) followed by another one performed to remove it ( $m_4$ ). In this case, the first two motions are performed in order to accomplish the *grasp-bottle* action, while the others are needed to accomplish the *open-bottle* action. Therefore, in the final plan  $\pi$  we will have four pairs, one for each motion, and two for each action, that is  $\pi = \langle (a_1, m_1), (a_1, m_2), (a_2, m_3), (a_2, m_4) \rangle$ . Notice that in this plan each pair represents a motion labeled by a symbolic action (e.g.,  $m_1$  and  $m_2$  are labeled by  $a_1$ ).

## An Extended RRT

The RRT algorithm (Lavalle and Kuffner 1999; 2000) is a sampling-based method that provides a motion plan by rapidly generating a tree rooted at the starting configuration until the goal configuration is reached. In this work, for the sake of clarity, we introduce our approach considering a simple version of the RRT algorithm, which will be suitably adapted to handle both task and motion planning. Notice that more sophisticated variants of RRTs can be similarly deployed (Kuffner and Lavalle 2000; Karaman and Frazzoli 2011; Gammell, Srinivasa, and Barfoot 2014; Klemm et al. 2015; Chen et al. 2018).

Following (Lavalle and Kuffner 2000), the class of problems considered by RRTs is typically defined by the following elements: a state space  $X$ ; boundaries  $x_{init} \in X$  and  $X_{goal} \subset X$  representing initial and goal configurations; a collision detection function  $D : X \rightarrow \{true, false\}$  checking whether or not a state is accessible; a set of control inputs  $U$  (motions); a simulator which predicts the state transition; a metric  $\rho : X \times X \rightarrow [0, \inf)$  specifying the distance between states.

We now show how combined task and motion planning problems can be included in this class and hence solved by RRTs. For this purpose, we consider a combined state space  $X = C \times S$  and a combined set of inputs  $U = M \times A$ . Starting from these sets, we define as boundary values  $x_{init} = (q_{init}, s_{init})$  and  $X_{goal} = (\cdot, s_{goal}) \subset C \times S$ , and a collision detection function  $D : C \times S \rightarrow \{true, false\}$  checking for the constraints of the combined state. Notice that, in this formulation, inconsistent symbolic states are treated analogously to obstacles to be avoided along the path.

As for the state transition, we define a simulator such that, given a combined state  $x(t) = (q(t), s(t))$  and an input  $\{u(t') = (m(t'), a(t')) | t \leq t' \leq t + \Delta t\}$ , the new state



is computed as follows:

$$x(t + \Delta t) = \begin{cases} (q(t + \Delta t), s(t)) & \text{if } q(t) \neq G(a(t'), s(t)) \\ (q(t + \Delta t), s(t + \Delta t)) & \text{otherwise} \end{cases} \quad (1)$$

That is, if  $q(t)$  is not the target configuration for  $a$ , only the configuration is updated, otherwise, once the target has been reached the symbolic state is updated as well.

We then define a distance  $d_u$  on the combined space  $X$  obtained as a weighted sum of two distance functions: the distance  $d_c$  in the configuration space  $C$  and a distance  $d_s$  on the symbolic space  $S$ . The latter, is based on the notion of symmetric difference of two sets. More specifically, the distance  $d_u$  is defined as follows:

$$d_u((q', s'), (q'', s'')) = w_c i + w_s j \quad (2)$$

where  $w_c, w_s \in \mathbb{R}^+$  are two positive and non-zero constant values introduced to weight the configuration distance  $i$  and the symbolic distance  $j$ , which are defined as follows:

$$i = d_c(q', q'') \quad (3)$$

$$j = d_s(s', s'') = |s' \triangle s''| = |(s' \cup s'') \setminus (s' \cap s'')| \quad (4)$$

Here, the distance  $d_c$  is for any distance between two configurations  $q'$  and  $q''$  (i.e. the one used to expand the RRT in the configuration space), while the symbolic distance is defined as the cardinality of the symmetric difference of the two symbolic states  $s'$  and  $s''$ . Since symbolic states are represented by sets of ground predicates, this cardinality provides a notion of proximity among the states.

We now have to show that  $d_u$  is a distance in the metric space  $(X, d_u)$ . We recall that a function  $d : X \times X \rightarrow \mathbb{R}$  is a distance in  $X$  iff (i)  $d(x, y) \geq 0$ ; (ii)  $d(x, y) = 0 \iff x = y$ ; (iii)  $d(x, y) = d(y, x)$ ; (iv)  $d(x, y) \leq d_u(x, z) + d(y, z)$ . Notice that if both  $d_c$  and  $d_s$  are distances in  $C$  and  $S$  respectively, then (i), (ii), (iii), (iv) hold also for  $d_u$  in  $C \times S$ . On the other hand,  $d_c$  is a distance on the configuration space by assumption, while it can be easily shown that the cardinality of the symmetric difference satisfies all the distance properties listed above.

## RRT-based Task and Motion Planner

Given the distance measure introduced in the previous section, we now aim at introducing a RRT algorithm that spans toward the unified metric space (configurations and symbolic) in search for a plan that is both collision-free and symbolically consistent.

---

### Algorithm 1 TM-RRT algorithm

---

```

1: function BUILD_TMRRT( $x_{init}, s_{goal}$ )
2:    $\mathcal{T} \leftarrow \text{addRRT}(\mathcal{T}, x_{init})$ 
3:   while  $\neg \text{reached}(\mathcal{T}, s_{goal})$  do
4:      $x_{rand} \leftarrow \text{random\_state}(\mathcal{T}, s_{goal})$ 
5:      $\mathcal{T} \leftarrow \text{extend}(\mathcal{T}, x_{rand})$ 
6:   end while
7:   return  $\mathcal{T}$ 
8: end function

```

---

The TM-RRT algorithm that combines task and motion planning is illustrated in Algorithm 1. It receives in input

the initial state in the combined space  $x_{init} = (q_{init}, s_{init})$  and a symbolic goal state  $s_{goal}$ , generating a RRT whose branches are associated with task-level actions and free-obstacle motions. The output of Algorithm 1 is the tree structure itself, since the plan can be suitably retrieved by going backward from the solution vertex (leaf) to the initial vertex (root). The tree is firstly initialized to the initial configuration (line 2) then, until the goal state  $s_{goal}$  is reached (line 3), the algorithm randomly selects a combined state (line 4) extending the tree in that direction (line 5). Finally, the tree containing the goal state is given as output (line 7). Notice that, differently from the basic version of the RRT in (Lavalle and Kuffner 1999), this algorithm stops when a feasible path connecting the initial and the goal states is found. Moreover a suitable *random\_state* function is defined in which symbolic actions are exploited to guide the sampling process toward the goal state.

---

### Algorithm 2 TM-RRT random sampling

---

```

1: function RANDOM_STATE( $\mathcal{T}, s_{goal}$ )
2:    $s_{rand} \leftarrow \text{random\_sym}(p_s, s_{goal})$ 
3:    $s_{near} \leftarrow \text{nearest\_sym}(\mathcal{T}, s_{rand})$ 
4:    $a \leftarrow \text{select\_action}(s_{near}, s_{rand})$ 
5:    $q_{sub} \leftarrow G(a, s_{near})$ 
6:    $q_{rand} \leftarrow \text{random\_conf}(p_c, q_{sub})$ 
7:   return  $(q_{rand}, s_{rand})$ 
8: end function

```

---

The description of the *random\_state* function is illustrated in Algorithm 2. In a first phase, a symbolic state  $s_{rand}$  is drawn from the space  $S$  (line 2). To bias this sampling towards the goal state, we adopt a randomized choice by selecting the goal state  $s_{goal}$  with probability  $p_s$  along with a randomly extracted symbolic state (i.e., a subset of  $P$  as in (Burfoot, Pineau, and Dudek 2006)) with probability  $1 - p_s$ . The selected state  $s_{rand}$  should induce a tree expansion in its direction. At the symbolic level, this expansion is performed by a symbolic action. In this respect, given  $s_{rand}$ , the algorithm selects the nearest state  $s_{near}$  of the tree (line 3) and selects an action  $a$  from the state  $s_{near}$  towards  $s_{rand}$  (line 4).

In a second phase, we exploit the  $G$  function (line 5) to retrieve the target-configuration  $q_{sub}$  associated with the action  $a$  executed in  $s_{near}$ . The  $q_{sub}$  is here exploited as a guidance that drives motion-level planning towards a trajectory that accomplishes  $a$  (sub-goal accomplishment). Also in this case, a randomized choice is introduced to obtain this drive: with probability  $p_c$  the drawn configuration  $q_{rand}$  equals the sub-goal  $q_{sub}$ , otherwise, with probability  $1 - p_c$  a new random configuration is sampled, with a uniform distribution over  $C$  (line 6). Once we have both  $q_{rand}$  and  $s_{rand}$ , a new sample  $(q_{rand}, s_{rand})$  in the unified space is provided in output. After the random selection of the state, the tree is expanded in that direction. This process is described in Algorithm 3. Analogously to the basic version of the RRT (Lavalle and Kuffner 2000), the *extend* function takes as input the tree  $\mathcal{T}$  and the random state  $x$ . The function selects the nearest node of the tree from the random state  $x$  accord-

---

**Algorithm 3** TM-RRT extension

---

```
1: function EXTEND( $\mathcal{T}, x$ )
2:    $x_{near} \leftarrow \text{nearest\_neighbor}(\mathcal{T}, x)$ 
3:    $u_{new} \leftarrow \text{select\_input}(x_{near}, x, len)$ 
4:    $x_{new} \leftarrow \text{new\_state}(x_{near}, u_{new})$ 
5:   if  $D(x_{new})$  then
6:      $\mathcal{T} \leftarrow \text{add\_vertex}(\mathcal{T}, x_{new})$ 
7:      $\mathcal{T} \leftarrow \text{add\_edge}(x_{near}, x_{new}, u_{new})$ 
8:   end if
9:   return  $\mathcal{T}$ 
10: end function
```

---

ing to the combined distance  $d_u$  (line 2); then, a combined input  $u_{new} = (m, a)$ , with  $m$  of maximum length  $len$  is selected (line 3) and applied to the nearest state  $x_{near}$  (line 4). If the new state is consistent - both obstacle free and symbolically consistent - (line 5) the tree is updated with the new node (line 6) and new edge (line 7).

### Case studies

In this section, we present and discuss case studies provided by a hospital-logistic scenario (see Figure 1 (right)) where a mobile robot is involved in multiple pick, carry, and place tasks. The experiments have been carried out in a realistic *CoppeliaSim* simulated environment of  $10 \times 10$  meters, deploying a laptop *Intel i5-5200U 2.20GHz, 8Gb ram*, with a single threaded implementation of the algorithm. An example of the simulated environment is provided in Figure 1, where we can find a mobile robot (gray, in simulation), two carts (cyan) and four target poses (green squares). The robotic system is an omni-directional mobile platform endowed with four mecanum wheels and an elevator that allows it to go under the carts and lift them from below. The configuration of the environment (rooms, passages, number of carts and poses) will be changed in the simulated experiments in order to generate incrementally complex settings.

In this scenario, we assume the robot configuration space  $C \subseteq \mathbb{R}^2 \times SO(1)$ , hence each  $q \in C$  is a triple  $q = [x, y, \theta]$ , where  $x$  and  $y$  are coordinates and  $\theta$  is the rotation. The set  $A$  includes two types of actions,  $pick(Cart, Pose)$  and  $place(Cart, Pose)$ , while the symbolic state is described by the predicates:  $free(Pose)$  that holds if the  $Pose$  is free,  $carry(Cart)$  which holds if the robot is carrying  $Cart$ ,  $on(Cart, Pose)$  stating that  $Cart$  is in  $Pose$ , and  $carrying$ , that holds if the robot is moving a cart. Both predicates and actions can be instantiated with carts and target-poses, hence the number of ground actions and predicates can be defined by the number of carts/target-poses.

#### Case 1: Parameters setting

The TM-RRT set-up depends on 5 main parameters: the probabilities  $p_c$  and  $p_s$  (see lines 2 and 6 in Algorithm 2) that regulate the goal-oriented sampling of the RRT, the length of the motion segment toward the random sample  $len$  (see line 3 in Algorithm 3), and the weights  $w_c$  and  $w_s$  introduced to define the unified distance  $d_u$  (see Equation 2). As a preliminary step, we assess the performance of the planner by

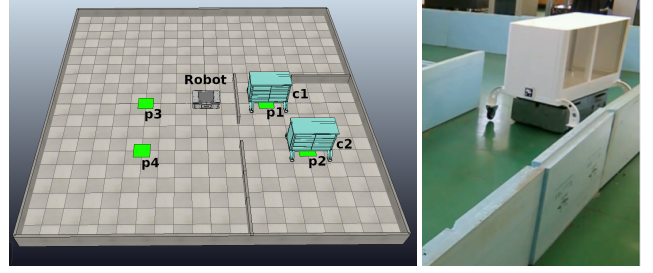


Figure 1: Example of simulated (left) and real (right) hospital environment. In the simulation 2 carts are deployed ( $c_1$  and  $c_2$ ) and one of them ( $c_1$ ) is blocking the entrance.

changing the parameters that structurally affect the underlying metric space and its exploration. Namely, we consider the ratio between the two weights  $w_c$  and  $w_s$  used to define the unified distance along with the  $len$  expansion step of the tree. In contrast, the probabilities  $p_c$  and  $p_s$  are both fixed to 0.3 in order to compare the results with the same search strategy.

In the following tests, we assume  $w_c = 1$  and  $w_s = kw_c$  and the ratio  $k \in \{0.1, 0.5, 1, 2, 5, 10\}$ , and  $len$  ranging in  $[0.1 - 1]$  with an increasing step of 0.1. The testing environment is illustrated in Figure 1 (left) and includes a small room with 2 entrances and a bigger outer area. In this setting, we deploy 2 carts (cyan objects  $c_1$  and  $c_2$ ) and introduce 4 target poses (green squares  $p_1$  to  $p_4$ ). The goal for the robot is to place  $c_1$  in  $p_3$  and  $c_2$  in  $p_4$ . Notice that one of the carts ( $c_1$ ) is intentionally positioned close to the bigger entrance in order to obstruct the passage. This way, the mobile robot can still access the room through the small entrance, but a free passage is needed anyway to carry out the cart  $c_2$ . This environment represents a simple “trap” since all the plans starting with the  $pick(c_2)$  action cannot be completed with  $carry$  and  $place$  because the only exit from the small room, available for  $c_2$ , is blocked. Instead, in order to solve the task, the robot firstly has to pick carry and place  $c_1$  from pose  $p_1$  to pose  $p_3$ , then cart  $c_2$  can be moved from pose  $p_2$  to pose  $p_4$ .

In this scenario, we tested the algorithm by performing 30 execution for each combination of the parameters. We also introduced a time-limit of 300 seconds considering failures all the executions exceeding this limit. For each couple of parameters  $(lim, k)$  we measured the time needed to generate the plan along with its size (average and standard deviation) and the failure rate. The results are illustrated in Figure 2. It is possible to notice that the failure rate decrease rapidly when the weight  $w_s$  of the symbolic distance becomes two or more times greater than  $w_c$ . Analogously, planning time also decreases with the increment of both  $k$  and  $len$ . In contrast, the size of the plan increases with the increment of  $len$ . This was expected since longer paths are generally associated to fast, but inefficient routes, while shorter paths are more detailed but computationally expensive. On the other hand, the plan size is still slightly decreasing with the increment of  $k$ . These collected results suggest that settings where  $w_c > w_s$  are self-defeating (higher failure rate), while

Table 1: Results of Case 2 (60 runs for each setting).

number of carts		1	2	3	4
time (s)	avg	0.22	1.31	11.80	39.27
	std	0.32	1.46	10.61	48.85
length (m)	avg	12.66	32.77	55.26	80.61
	std	6.64	8.51	12.07	11.88
plan-size	avg	142	364	625	906
	std	77	97	143	151
success		100%	100%	100%	93%

the  $len$  parameter can be regulated to trade-off between efficiency and quality of the solution. For the other case studies, we selected the set-up associated with no failures and minimal execution time (see green line in Figure 2), namely  $len = 0.9$  and  $k = 5$ .

### Case 2: Scalability

We now consider the scalability of the proposed approach with respect to the number of actions and ground predicates (i.e., the size of the sets  $A$  and  $P$ ). We defined a simulated environment with 2 rooms and 3 passages between rooms that are large enough to allow cart transitions (see Figure 3). In this scenario, the robot has to move carts from their positions in the right-room to the fixed position in the left-room (e.g., from  $p_1$  to  $p_2$  in Figure 3, first on the left). We considered 4 configurations of this scenario with an increasing number of carts (from 1 to 4) and target-poses (from 2 to 8). The increment of tasks/poses induces an increment of the available ground actions and predicates as illustrated in Figure 4.

For each setting, we performed 60 runs (240 tests in total), measuring planning-times (in seconds), size of the plans (number of steps), paths length (meters) and success rate (percentage of plans generated in less than 300 seconds).

The empirical results are summarized in Table 1. Tests with 1 up to 3 carts have been always successfully executed, while in the 4-carts case, success rate is 93% (planning problem could be solved within the deadline). Notice that for the 4 considered settings the sizes and lengths of the plans increase almost linearly with the size of the problem, while planning times grow more rapidly. This was expected and related to the increasing complexity of the symbolic search space (number of ground predicates and actions).

### Case 3: Anomalies

In order to assess our approach in case of anomalies, we propose a scenario where the carts configuration is intentionally designed to hinder task execution. More specifically, we assume that the simulated environment includes 2 small rooms connected by an outer area and 2 carts (see Figure 5). The goal is to move the cart  $c_2$  (lower-left cart) from the lower to the upper room and to leave cart  $c_1$  in pose  $p_1$ . In order to accomplish this task, the robot has to move the cart  $c_1$  outside of the room (to free the passage), carry the second cart  $c_2$  to the upper room and, finally, take back  $c_1$  at the initial pose. Notice that in this setting the robot is mostly guided

by the symbolic distance to firstly take cart  $c_2$  (hence to perform  $pick(c_2, p_2)$  and  $place(c_2, p_4)$ ) instead of moving  $c_1$ . Here the RRT-based sampling allows the robot to overcome this local minima by rejecting the colliding solutions, while considering also sub-optimal ones.

The collected empirical results are illustrated in Table 2. Also in this case 60 tests have been performed with executions always successful (plan generated in less than 300 seconds). The task and motion planning process takes in average 14.82 seconds and produces a plan of 673.71 average steps. Since in this setting 3 pick-and-place tasks are executed, we can compare these results with respect to the ones of the previous case study (Table 1, 3-carts) observing that the anomaly induces a slight increment of the planning time (around 3 seconds ( $\sim 25\%$ )), while the plan size is analogous (around +8% plan-size and +9% path length).

Table 2: Results of Case 3 (60 runs).

local-minima	time	length	plan-size	success
avg	14.84	59.93	673.71	100%
std	10.15	12.53	137.67	

## Conclusions

We presented a RRT-based approach to combined task and motion planning. While RRTs are widely exploited for solving motion planning problem in high-dimensional spaces, their deployment for symbolic task planning problems is more limited. In this respect, we propose a RRT-based algorithm suitable for finding plans in an extended search space that combines configuration states and symbolic states. This extended state space is associated with a distance measure that combines a distance in the configuration space and distance on the symbolic state. Given this unified distance, we deployed a basic version of the RRT-based search on the extended metric state. We detailed the approach and discussed its feasibility and scalability in a pilot study provided by a real-world hospital scenario that involves a mobile robot executing pick-carry-place tasks. Despite the simplicity of the algorithm, the collected empirical results show that the approach is feasible and effective in increasingly complex realistic test cases, encouraging us towards the deployment of more sophisticated RRT methods and a more refined implementation of the algorithm (the one tested in this work is single threaded). As a future work, we plan to compare the performance of different variants of RRT algorithms and notions of symbolic distance in both navigation and manipulation tasks.

## References

- Barry, J. L.; Kaelbling, L. P.; and Lozano-Pérez, T. 2013. A hierarchical approach to manipulation with diverse actions. In *2013 IEEE International Conference on Robotics and Automation, Karlsruhe, Germany, May 6-10, 2013*, 1799–1806.
- Bidot, J.; Karlsson, L.; Lagriffoul, F.; and Saffiotti, A. 2017.

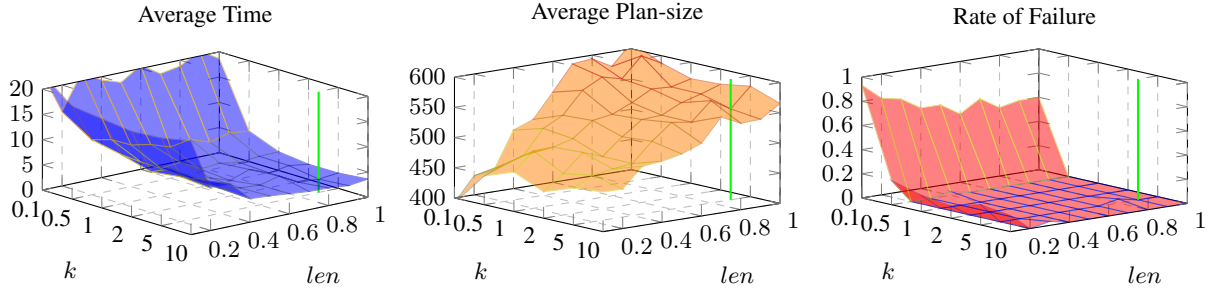


Figure 2: Representation of the average times (seconds), plan-size (steps) and the rate of failures for each couple of parameters over 30 runs. For the next cases we select the parameters producing no failures and minor average time (green line).

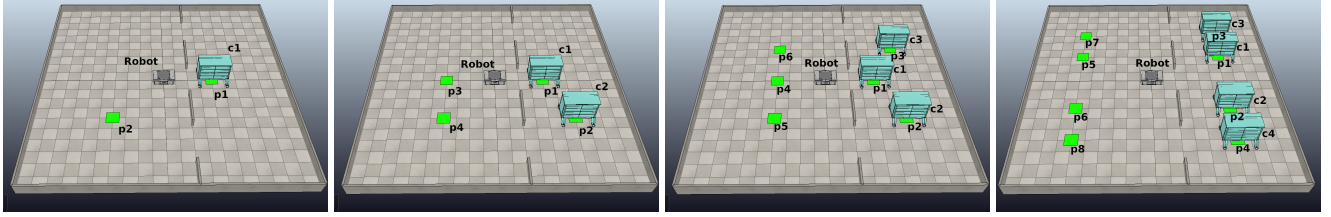


Figure 3: Simulated environments for each setting from 1 (left) to 4 (right) carts. In all these settings the robot carries carts from the right room to the target-poses in the left room. There are three passages between rooms.

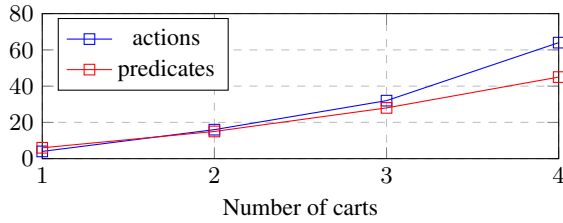


Figure 4: Number of actions and ground predicates for each domain.

Geometric backtracking for combined task and motion planning in robotic systems. *Artif. Intell.* 247:229–265.

Burfoot, D.; Pineau, J.; and Dudek, G. 2006. Rrt-plan: A randomized algorithm for STRIPS planning. In *Proceedings of the Sixteenth International Conference on Automated Planning and Scheduling, ICAPS 2006, Cumbria, UK, June 6-10, 2006*, 362–365.

Cambon, S.; Alami, R.; and Gravot, F. 2009. A hybrid approach to intricate motion, manipulation and task planning. *I. J. Robotics Res.* 28(1):104–126.

Chen, L.; Shan, Y.; Tian, W.; Li, B.; and Cao, D. 2018. A fast and efficient double-tree rrt\*-like sampling-based planner applying on mobile robotic systems. *IEEE/ASME Transactions on Mechatronics* 23(6):2568–2578.

Choi, J., and Amir, E. 2009. Combining planning and motion planning. In *2009 IEEE International Conference on Robotics and Automation, ICRA 2009, Kobe, Japan, May 12-17, 2009*, 238–244.

Dantam, N. T.; Kingston, Z. K.; Chaudhuri, S.; and Kavraki,

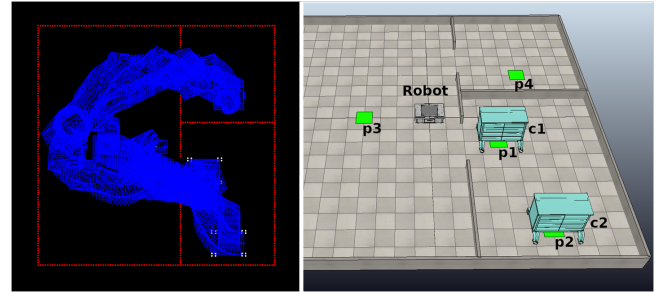


Figure 5: Generated plan (left) and environment (right). The robot moves the central cart outside of the rooms in order to free the passage.

L. E. 2016. Incremental task and motion planning: A constraint-based approach. In *Robotics: Science and systems*, volume 12, 00052. Ann Arbor, MI, USA.

de Silva, L.; Pandey, A. K.; and Alami, R. 2013. An interface for interleaved symbolic-geometric planning and backtracking. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems, Tokyo, Japan, November 3-7, 2013*, 232–239.

Erdem, E.; Haspalamutgil, K.; Palaz, C.; Patoglu, V.; and Uras, T. 2011. Combining high-level causal reasoning with low-level geometric reasoning and motion planning for robotic manipulation. In *2011 IEEE International Conference on Robotics and Automation*, 4575–4581. IEEE.

Gammell, J. D.; Srinivasa, S. S.; and Barfoot, T. D. 2014. Informed rrt\*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic.

In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2997–3004. IEEE.

Ingrand, F., and Ghallab, M. 2017. Deliberation for autonomous robots: A survey. *Artificial Intelligence* 247:pp.10–44.

Kaelbling, L. P., and Lozano-Pérez, T. 2011. Hierarchical task and motion planning in the now. In *IEEE International Conference on Robotics and Automation, ICRA 2011, Shanghai, China, 9-13 May 2011*, 1470–1477.

Karaman, S., and Frazzoli, E. 2011. Sampling-based algorithms for optimal motion planning. *The international journal of robotics research* 30(7):846–894.

Klemm, S.; Oberländer, J.; Hermann, A.; Roennau, A.; Schamm, T.; Zollner, J. M.; and Dillmann, R. 2015. Rrt-connect: Faster, asymptotically optimal motion planning. In *2015 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, 1670–1677. IEEE.

Kuffner, J. J., and Lavalley, S. M. 2000. Rrt-connect: An efficient approach to single-query path planning. In *Proceedings IEEE International Conference on Robotics and Automation*, 995–1001.

Lagriffoul, F.; Dimitrov, D.; Bidot, J.; Saffiotti, A.; and Karlsson, L. 2014. Efficiently combining task and motion planning using geometric constraints. *The International Journal of Robotics Research* 33(14):1726–1747.

Lavalley, S. M., and Kuffner, J. J. 1999. Randomized kinodynamic planning. In *Proceedings IEEE International Conference on Robotics and Automation*, 473–489.

Lavalley, S. M., and Kuffner, J. J. 2000. Rapidly-exploring random trees: Progress and prospects. In *Algorithmic and Computational Robotics: New Directions*, 293–308.

Morgan, S., and Branicky, M. 2004. Sampling-based planning for discrete spaces. In *Proceedings of 2004 IEEE International Conference on Intelligent Robots and Systems, 2004*, 1938–1945.

Plaku, E., and Hager, G. D. 2010. Sampling-based motion and symbolic action planning with geometric and differential constraints. In *IEEE International Conference on Robotics and Automation, ICRA 2010, Anchorage, Alaska, USA, 3-7 May 2010*, 5002–5008.

Thomason, W., and Knepper, R. A. 2019. A unified sampling-based approach to integrated task and motion planning. In *International Symposium on Robotics Research (ISRR)*.

# Two-layered Architecture for Telepresence Robots: Combining Personalization and Reactivity

**Gloria Beraldo** and **Riccardo De Benedictis** and **Amedeo Cesta** and **Gabriella Cortellesa**  
Institute of Cognitive Sciences and Technologies, National Research Council of Italy, Rome, Italy

## Abstract

This paper proposes a system design for integrating the planning of courses of actions with their execution on a telepresence robots to face some outstanding challenges, namely, the need to provide personalised services for effective human-robot interaction, and to create the robot's ability to perform reactive behaviors in autonomous or semi-autonomous way. With this purpose in mind, we put together the strengths of AI and robotics into a two-layered architecture: the first layer is in charge of planning and scheduling the daily activities by respecting the personalised constraints from human needs, the second implements and adapts the robot's actions by evaluating contextual information. Besides promoting the continuous and mutual interaction between the two layers, the proposed system favors the distribution of services on the cloud and on board the robot by considering the trade-off between personalisation and reactivity. The preliminary tests highlight the feasibility of the system and lay the basis for developing a new assistive solution.

## Introduction

Recent studies foresee an increment of the old-age dependency ratio (number of people age 65 or above compared to those age 15-64) from the current 28% to 50% by 2060, highlighting the necessity of developing new healthcare services for assisting elderly people in the daily life activities (Group and others 2018; Mois and Beer 2020). Ageing causes a physiological decrease of motor, sensory and cognitive abilities in people. Seniors with cognitive disorders could have problems in being self-sufficient and living alone at home. Furthermore, elderly people need to reduce the risk of accidents at home, the progressive cognitive decline, dementia and the isolation from the family members and friends. For these reasons, over the past few years, requests for personalised care services have increased. Given these premises, it is emerging the demand to combine artificial intelligence and robotics to develop new care solutions based on the design of social robots able, to establish empathic bonds with people as well as to monitor and coaching them in order to anticipate their needs and predict the occurrence of impairments (Pollack 2005). Mobile Telepresence Robots, in particular, represent a class of robots that allow mitigating the problem of isolation of elderly people

living alone (Cesta et al. 2018; Sheridan 1992). Although the technology used on these robotic platforms has evolved considerably in recent years (Isabet et al. 2021), these tools are basically relegated to providing telepresence services on different mobile robotic platforms that can be controlled remotely (Melendez-Fernandez, Galindo, and Gonzalez-Jimenez 2017; Kristoffersson, Coradeschi, and Loutfi 2013; Tsui et al. 2011), having no or very limited autonomy (Orlandini et al. 2016). Usability and acceptability of these platforms, however, require that they have a certain level of autonomy, supporting their teleoperation by people typically not very familiar with technology (e.g., family members of the assisted persons, as well as doctors and nurses), as well as endowing the robots with proactive services and capabilities, exploitable when they are not remotely teleoperated (Isabet et al. 2021; Riano, Burbridge, and McGinnity 2011; Laniel et al. 2021).

We are currently facing this research topic inside the project called SI-ROBOTICS (Social ROBOTics for active and healthy ageing), which aims designing and developing advanced and customized human-robot interaction for supporting elderly people at home. The main goal consists of creating an holistic system that includes multivariate services from the monitoring of the physiological measurements to the motivation of performing cognitive and physical exercises through the robot, from telepresence services to advance remote teleoperation to keep the contact with secondary users (e.g., doctors, medical staff, family, friends). In this context, one of the challenges is the definition of the appropriate role of the robot. On the one side, robots are expected to implement in autonomy reactive and intelligent behaviors according to the contextualization of the specific situations to assist the users. On the other, robots have to operate inside predefined boundaries to be acceptable and to adapt to the necessities of people. In other words, it emerges the necessity of planning the appropriate tasks involving the robot during the day, to schedule them by respecting a set of constraints (e.g., temporal, related to people's preference), and finally, to implement them exploiting the robot's capabilities. However, although the planning and scheduling is essential, at the same time there are some aspects of the interaction that cannot be planned a priori. For instance, you can imagine if the robot is expected to remember the person the monitoring of his/her physiological parameters, but the



person is not at home.

To limit the failures, in this paper, we propose to fuse the strengths of a global planning and scheduler in combination with the implementation of reactive behaviors that locally access and interpret the contextual information. With these purpose, we also focus on designing strategies to modify the original plan after triggers from the robot. In particular, during the execution of the reactive routines, the presented architecture incorporates both the capabilities of rearranging the expected time of the scheduled activity (e.g., delay it) and/or to completely changing the plan when it is impossible to be committed (as in the previous example).

## Proposing a two-layered architecture

It is worth highlighting that the integration between artificial intelligence and robotics services is not trivial as merely solving a problem of task planning and execution. The integration, indeed, should rely on two different layers of abstraction that actively operate and continuously exchange information among them. The pursued idea is based on the integration of different AI technologies that support the implementation of capabilities necessary to support humans in daily living scenarios fostering personalization and adaptation of the interaction (Rossi, Ferland, and Tapus 2017; Moro, Nejat, and Mihailidis 2018). Similarly to other works (Strannegård et al. 2013; Sun 2015; Augello et al. 2017), the proposed architecture is inspired by the dual process theory, a psychological model according to which the human mind would follow two distinct and parallel reasoning processes: one faster and more intuitive (e.g., associated with robotics), the other slower but more logical and reflective (e.g., related to semantic reasoning, automated planning and scheduling) (James 1890; Wason and Evans 1974; Petty and Cacioppo 1986; Kahneman 2011). In accordance with this theory, our goal is to combine the strengths of the two forms of reasoning and acting into one synergic system, whose architecture is organized as follows:

- The higher *slower* layer, associated with the artificial intelligence, deals with the problem of abstracting the context, reasoning and planning of a personalised set of actions performed by the robot. This layer distributes the services *on cloud* that are independent from the robotic platform and require burdensome computation resources.
- The lower *faster* layer, related to robotics, is in charge of trying to execute the scheduled actions by rearranging them proactively according to the specific context and by defining the aspects that are not established a priori. For this reason, the services are directly implemented *on-board* of the robot.

An illustrative representation of the proposed two-layered architecture is shown in Figure 1.

Given the structure of the architecture, it is fundamental to provide a good communication between the two layers that also takes into account their different “speed”. Indeed, the lower is more responsive to possible changes because it acts locally in a limited window of time. The second is more sophisticated because it evaluates the entire picture of possible activities but with the trade-off of requiring time to for-

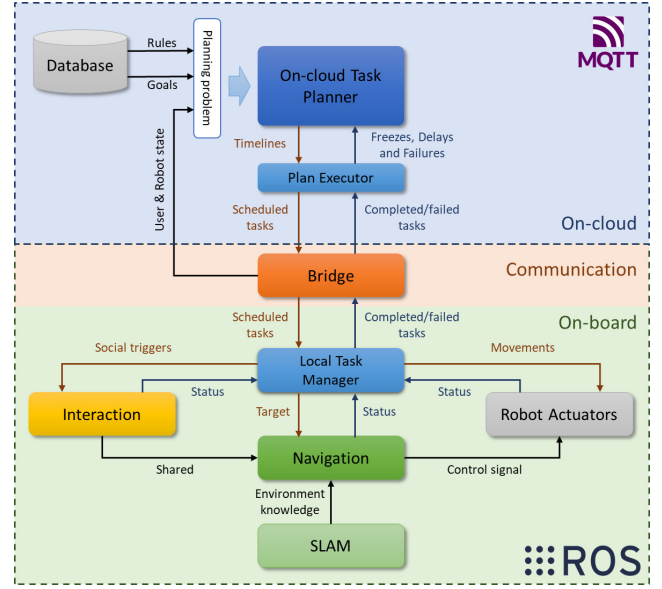


Figure 1: A sketch of the two-layered architecture. The first is based on the Message Queuing Telemetry Transport (MQTT), the second on Robot Operating System (ROS).

mulate a feasible solution respecting all the constraints and adapting it at execution time. The communication function is managed by a bridge that plays a dual role: put in communication the previous two layers and translate the information in the expected format.

In the following sections, we will clarify the roles and the characteristics of both layers.

## The on-cloud layer

As mentioned in the previous section, the overall system must be able to manage the communication between, and the interaction with, the different users of the system, whether they are assisted elderly, medical staff, relatives or just friends who can interact, through the system, with clients. These users, in particular, are typically connected to the system remotely, from their respective workstations or, sometimes, from their homes. Therefore, it is crucial to establish a cloud infrastructure that allows the exchange of the information among the different components of the overall system. Such infrastructure, in particular, will cover the deliberative role of all the autonomous components that characterize the system, planning the activities for the achievement of objectives related to the well-being of the person and coordinating their execution, dynamically adapting it to the information that continuously emerges from the environment.

Specifically, a database is responsible for keeping information regarding the different users who have access to the system, their roles (e.g., assisted users, medical staff, etc.) and, for the assisted users, those characteristics which are relevant for the generation of personalized plans (e.g., any physical and/or cognitive problems). The database also contains information regarding the associated assistive structures (the main objective is to minimize hospitalizations,

hence these structures are, mostly, the assisted users' own homes), the users who have access to the structure, the environmental and physiological sensors within the structure and the available robotic platforms.

As the overall system consists of several building blocks, as briefly mentioned, coordination among their performed activities results mandatory. For this reason, the on-cloud tier also deals with the planning, scheduling and execution, with possible dynamic adaptation over time, of the highest level tasks that the various components connected to the system must carry out over time. This coordinating role, in particular, is carried out by different instances (one for each assistive structure) of a timeline-based planner (Muscettola et al. 1992), which deals with managing, in an integrated and homogeneous way, the different forms of semantic and causal reasoning required by the system's high-level tasks. Such activities will then be carried out by the reactive tier, which in turn will provide feedback for the dynamic adaptation of the personalised generated plans.

### Timeline-based planning

In order to better understand what we are talking about, it is worth introducing some technical background about timeline-based planning. The building block for the logical and reflexive reasoning tier, in particular, is the *token* which, in its most general form, is described by an expression having the form  $n(x_0, \dots, x_i)_\chi$ . In timeline-based planning tokens are used to represent the single unit of information. In particular,  $n$  is a *predicate* symbol and  $x_0, \dots, x_i$  are its *parameters* (i.e., constants, numeric variables or object variables). Such parameters are constituted, in general, by the variables of a constraint network (Dechter 2003; Lecoutre 2009) and can hence be constrained to reduce their allowed values and bring the system to a desired behavior. Finally,  $\chi \in \{f, g\}$  is a constant representing the class of the token (i.e., either a *fact* or a *goal*).

The semantics, in this case, is borrowed from Constraint Logic Programming (Apt and Wallace 2007). Specifically, while the facts are considered inherently true, the goals must be achieved. The achievement of a goal can take place either through a *unification* with either a fact or another already achieved goal with the same predicate (i.e., the parameters of the current goal and the token with which is unifying are constrained to be pairwise equal), or through the application of a rule.

Rules are expressions of the form  $n(x_0, \dots, x_k) \leftarrow \mathbf{r}$  where  $n(x_0, \dots, x_k)$  is the *head* of the rule and  $\mathbf{r}$  is the *body* of the rule, i.e., either another token, a *constraint* among tokens (possibly including the  $x_0 \dots x_k$  variables), a *conjunction* of requirements or a *disjunction* of requirements. Specifically, rules define the causal relations that must be complied to in order for a given goal to be achieved. Roughly speaking, for *each* goal having the "form" of the head of a rule, the body of the rule (i.e., further tokens, constraints, conjunctions and disjunctions) must also be asserted. Through the application of the rules it is hence possible to establish and generate relationships between the information units represented by the tokens.

Finally, a *timeline-based planning problem* is a triple  $\mathcal{P} = (\mathbf{O}, \mathcal{R}, \mathbf{r})$ , where  $\mathbf{O}$  is a set of typed objects, needed for instantiating the domains of the constraint network variables,  $\mathcal{R}$  is a set of rules and  $\mathbf{r}$  is the body of a rule, i.e., either a token (whether a fact or goal), a constraint among tokens, a conjunction of requirements or a disjunction of requirements. In simple terms, the reasoning procedure deals with applying the rules in order to guarantee the achievement of goals (note that this process can introduce other (sub)goals) or with demonstrating their semantic equivalence with other facts or with other already achieved goals (refer to (De Benedictis and Cesta 2020) for further details).

Note that some of the numeric parameters of the tokens can be used to represent temporal information such as the start or the end of some tasks. Additionally, constraints can be used to impose orderings on such tasks, placing them at proper times. Some tokens, moreover, may be equipped with a special object variable  $\tau$  that identifies the timeline affected by the token. Different tokens with the same value of the  $\tau$  variable interact with each other on the same timeline. Such interaction, in particular, depends on the nature of the timeline, prohibiting the temporal overlap in case of *state-variables*, or allowing it, as long as the concurrent uses remain below the resource's capacity, in case of a *reusable-resource*. By doing so, it is possible to homogeneously represent different types of information that can interact with each other such as the users' profile, the tasks that the robot must perform, and also the tasks that the robot proposes to users such as physical and cognitive rehabilitation exercises, hence resulting in the generation of a high-level personalised interaction with the user.

Comparing with the human mind, a token can be seen as an "idea", like the idea of having to perform a certain task (e.g., a physical rehabilitation exercise) in a given future temporal interval, or the idea that the person with whom the system is interacting has certain characteristics which can be exploited for customizing the interaction. Through the application of the rules, the resolution process introduces new "ideas" and new relationships between them.

### Personalization of the daily activities according to user's needs and preferences

Given its general modeling flexibility, we have chosen to use the above form of reasoning at a deliberative tier, to generate personalised high-level plans and to adapt them according to the new information that dynamically emerges from the environment. Within the proposed architecture, specifically, personalization concerns the generation of activities over time (e.g., physical and cognitive rehabilitation exercises) personalised on the basis of the user's characteristics (i.e., any physical problems, the degree of training, the level of perceived fatigue after a training session, etc.) and on her/his preferences (e.g., the preferred times to carry out such activities). In particular, by exploiting a combination of rules and constraints, and by using constants, whose value is established starting from the information stored in the database, to represent the user's status and her/his preferences, it is possible to generate a long-term plan containing several high-level interactions customized according to the

user's needs.

Figure 2 sketches how it is possible to exploit the application of the rules, during the resolution process, to generate personalised plans that aim to keep the user healthy. Suppose, for example, that from the interaction with an elderly user  $u$  emerges that the person suffers from memory problems while is free of major physical problems. The deliberative tier could exploit this knowledge to plan some cognitive rehabilitation exercises, aiming to limit the person's further cognitive decline, and some physical exercises, aiming to keep the person physically active. A hypothetical high-level goal could be, therefore, KEEPHEALTHY. The rule that describes the requirements for the achievement of such a goal could contain, among other things, a first disjunction like  $(u.memory\_issues \wedge CognitiveExercise_g(\tau, s, e)) \vee \neg u.memory\_issues$  and a second disjunction like  $(\neg u.physical\_issues \wedge PhysicalExercise_g(\tau, s, e)) \vee u.physical\_issues$ . In order to reach the KEEPHEALTHY goal, specifically, the application of the rule, during the resolution process, will introduce a COGNITIVEEXERCISE sub-goal within the partial plan if and only if the involved user has memory problems and a PHYSICALEXERCISE sub-goal if and only if the involved user has no physical problems.

It is worth noting that the introduced sub-goals are endowed with the  $s$  and  $e$  numerical variables, representing the starting and the ending of the activity. Furthermore, the sub-goals affect the timeline represented by the object variable  $\tau$ . The value of such variables is, in general, decided by the planner, provided that all the constraints defined both in the problem and in the applied rules are satisfied. State-variable timelines, specifically, prevent the temporal overlapping of their tokens introducing, if needed, ordering constraints between them avoiding, in our case, the overlapping of a cognitive rehabilitation exercise and a physical one. Finally, it is worth noticing that the introduced sub-goal are, actually, goals and, like other goals, require to be achieved, either by unifying with another already achieved COGNITIVEEXERCISE and PHYSICALEXERCISE goals, or by applying the COGNITIVEEXERCISE's and PHYSICALEXERCISE's associated rule, possibly introducing additional constraints and further sub-goals, such as a WARM-UP exercise preceding the PHYSICALEXERCISE.

## Executing a timeline-based plan

Generating a personalised plan that manages the coordination among the different activities that might take place within an assistive structure constitutes only part of the faced difficulties. The generated plan, in particular, must deal with the evolving reality of all the (often, unpredictable) events that can happen within an assistive structure. In other words, the plan must be executed. To manage the execution of the plan, two associative containers are filled with all the tasks of the plan that will eventually be executed, indexed, respectively, by the value of their start variable and by the value of their end variable. For convenience, the keys of these containers are sorted in ascending order, so as to quickly identify the next tasks which will start/end. An internal *current-time*

variable is incremented at each execution step (in our case, every second) and whenever its value exceeds the beginning (or the end) of a task, these are started (or ended). During the execution of a plan, however, various things can happen that require its adaptation (or, in some cases, its cancellation) depending on the fresh information that dynamically emerges from the environment. In its most general form, in particular, the types of adaptations we consider fall into three categories: 1) *freezing* the start or the end of a task; 2) *delaying* the start or the end of a task and 3) *failing* the execution of one (or more) tasks.

Each of the above adaptations, in particular, can lead to potentially important consequences for the plan. In the event of a task failure, for example, it is necessary to take into account the involved causal relations. In the case shown in Figure 2, for example, the WARM-UP activity is introduced so as to allow the achievement of the physical exercise goal. The failure of the WARM-UP exercise, consequently, implies the failure of the physical exercise goal, possibly propagating the failure to the KEEPHEALTHY high-level goal. Similarly, in case of a delaying of the start of a task, tightening the lower-bound of the corresponding temporal variable, by introducing a new constraint, might imply the propagation to other variables. In the case shown in Figure 2, for example, the physical exercise precedes the cognitive one. Delaying the start of the physical exercise involves updating the value assigned to the temporal variable that indicates its end, updating the value assigned to the temporal variable indicating the start of cognitive exercise, and so on. Moreover, whenever a task is started (or, similarly, ended), the corresponding temporal variable must be frozen. Specifically, the value of the variable is constrained to be equal to its current value. In so doing, the number of allowed values of other variables is possibly reduced, for example, forbidding the violation of max duration constraints. Finally, it is worth noting that the tightening of a lower-bound on a temporal variable could make the problem inconsistent, decreeing the failure of the task associated with the temporal variable.

Considering the positioning of the plan executor in the cloud, however, it is advisable to limit as much as possible the exchange of information between the robotic platforms and the planner, so as to reduce the amount of information traveling through the network. In this regard, it is worth highlighting that, depending on the characteristics of the robot, four (possibly intersecting) sets of tasks can be identified: 1) *notify-start*, during the execution of the plan, all the tokens with these predicates are dispatched when the start of the tasks is reached; 2) *notify-end*, during the execution of the plan, all the tokens with these predicates are dispatched when the end of the task is reached; 3) *auto-start*, during the execution of the plan, all the tokens with these predicates are considered started when the start of the task is reached; 4) *auto-end*, during the execution of the plan, all the tokens with these predicates are considered finished when the end of the task is reached.

This characterization of the predicates allows the plan executor to have a good flexibility, covering most of the scenarios that can arise in the case of interaction by a companion robot with the user. In short, the start of the tasks is

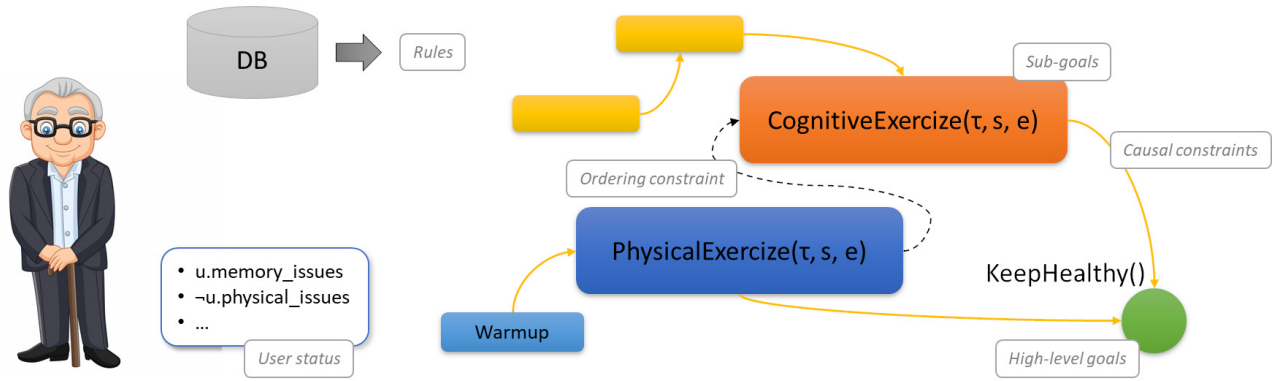


Figure 2: Personalization of daily activities through integrated semantic and causal reasoning.

always decided by the planner, which starts them as soon as the value of the current-time reaches the value of the start variable. When this happens, the planner freezes the start variables and stores the starting tasks in an internal *executing* set. Auto-end tasks are terminated whenever the value of the end variable is reached, removing them from the *executing* set and freezing their ending variables. In all other cases the end of the task is delayed by one time unit. Whenever a task is declared as terminated by the reactive layer, however, it is removed from the *executing* set and its ending variable is frozen. Finally, only the start and the end of tasks classified as notify-start and notify-end are notified at the reactive level. In so doing, it is possible to limit the communication to the sole tasks which are strictly relevant to the reactive tier which, on the other hand, must only notify their successful (or ruinous) termination.

As an example, navigation activities, for which the duration is not known a priori, fall within the sole notify-start set. Whenever the current-time value reaches the value of the start variable, the latter is frozen and the start of the action is notified to the reactive tier. The end of the task, on the contrary, will be determined by a notification coming from the reactive tier. Since the task is not classified as auto-end, whenever the current-time value reaches the value of the end variable, the latter will be increased by a time unit at each execution step, propagating all the plan constraints with possible consequences on other future tasks, until the executing task is notified as successful (or ruinous).

### The communication layer

As we previously anticipated, it is essential to introduce a communication layer which aims at triggering the starting of the activities scheduled by the cloud to the on-board layers and notifying their conclusions. That service is perfectly integrated into the architecture and it is managed by a bridge as shown in Figure 1. Moreover, the bridge is crucial as intermediate glue for connecting the two different middlewares on which the *on-cloud* and the *on-board* layers rely. On the one side, we exploit the Message Queue Telemetry Transport (MQTT) as the standard protocol for

the Internet of Things (IoT)<sup>1</sup> and for connecting devices on the cloud, on the other Robot Operating System (ROS)<sup>2</sup> the standard “de facto” for robotics applications. The main pro of MQTT is the possibility of connecting different nature of devices (e.g., sensors, phones, tablets, personal computers, televisions) over the cloud, while counting on queuing messages and on guarantees for their delivery (QoS), necessary in all those situations in which disconnections may occur (e.g., in the case of connections with mobile devices). The advantage of introducing ROS, conversely, consists of adding advanced services (e.g., autonomous and shared autonomy navigation, people-aware navigation, Simultaneous Localization and Mapping (SLAM), skeleton detection) on board of commercial telepresence robots. These services exploit the standard provided by ROS and extend some state-of-the-art algorithms in robotic and computer vision shared over the wide community.

To allow the exchange of the information in a flexible way, the bridge<sup>3</sup> exploits the common *publisher/subscriber* protocol of communication. Specifically, the bridge is in charge of converting the data in the expected format (e.g. ROS message, JavaScript Object Notation (JSON), general chunk of data) and republishing them in the corresponding topic.

It is worth highlighting the importance of using the bridge namely the advantage of managing data of different nature by leading them back to standard messages distributed in the architecture. Moreover, it allows treating data in the traditional way from both sides namely independently from the presence of the other middleware by favoring reusability and integrity.

### The interaction between the on-cloud and on-board levels

The interaction between the *on-cloud* and the *on-board* levels is dynamic and intermittent. However, the two layers are strictly interconnected: the *on-cloud* determines the start of the activities and might estimate their end, the *on-board* es-

<sup>1</sup><https://mqtt.org/>

<sup>2</sup><http://wiki.ros.org/>

<sup>3</sup>The bridge relies on the ROS package: [https://github.com/groove-x/mqtt\\_bridge\\_package](https://github.com/groove-x/mqtt_bridge_package)

establishes when they are definitely concluded. Since different reactive components may be interested in different high-level commands produced by the deliberative tier during execution, the current implementation publishes the commands on different topics identified by the token predicate corresponding to the command. In this way, each member can subscribe only to the topics of their interest, limiting the exchange of less interesting information. Furthermore, since the predicate is identified by the topic, the information emitted by the cloud is limited to the value of the token parameters, two boolean parameters indicate whether the task is starting or ending. Finally, an additional parameter is used to identify the command sent from the cloud, allowing the reactive module to communicate the conclusion or failure of the tasks.

Given the heterogeneity of the reactive services, we also arrange an intermediary module, placed on the *on-board* layer, that is in charge to directly communicate with the cloud through the bridge and orchestrates the different modules. Specifically, this module, that we call *local task manager*, waits for receiving messages from the bridge. Each message includes the scheduled activity identified with a number and the high-level arguments (e.g., the expected robot's position at the beginning of the activity). Thus, the *local task manager* is in charge of awakening the specific reactive module (actuators, interaction and navigation). Indeed, with respect to the executor of the *on-cloud* layer, it works at a different level of abstraction. For example, from the current activity *CognitiveExercise* communicated by the *on-board* layer, the *local task manager* activates the *robot's camera*, the *skeleton tracker* to verify the present of a person and the *fts* functionalities for managing the interaction.

At the end of the activity, the involved module communicates the conclusion to the *local task manager* by specifying the identifier of the finished activity and its binary outcome (success or failed). Finally, the *local task manager* collects all the identifiers of the activities ended with success and failed that are published respectively in the topics */done* and */failed* to be transmitted to the *cloud* through the bridge. The deliberative tier, at this point, removes the completed tasks from the set of the executing ones and freezes their final variable. In the case of a failure, on the contrary, the cloud removes the failed tasks from the plan, properly managing the causal consequences (i.e., eliminating any other activities that cannot be achieved due to the failure of some activities) and generating, if allowed by the disjunctions defined within the rules, an alternative plan.

### The on-board layer

The *on-board* layer manages the reactive behaviors implemented by the robot from the formulated plan.

The preliminary version of the architecture provides three main kinds of services: the ones associated with the direct movements of robots actuators, the ones related to the robot navigation and, finally the ones handling the interaction between humans and the robot. As shown in Figure 1, these services are managed by three different modules, but not completely stand-alone.

The first class of services are introduced to implement basic actions on the robot's actuators oriented to improve the interaction with humans (i.e., move hand up to greet, regulate the robot's height according to the person's position) and/or to better acquire data from the environment (i.e., move the robot's head to also shift the field of view of the camera). The low-level control signals are established by the *on-cloud* planner to set the default configuration of the joints and the actuators of the robot according to the user's needs, that might be modified by the interaction and the navigation modules afterwards according to the specific situation. Indeed, both modules, interaction and navigation ones, process the context information at higher level than actuators, that are finally associated with robot's movements (e.g., from the robot's trajectory to velocity commands, from the greeting sentence to hello gesture).

To manage the robot's motion in the space, the navigation module<sup>4</sup> abstracts the knowledge of the environment and outputs navigation goal for the robot. The navigation goals for the robot are set in multiple ways. The navigation goals can be statically chose from the *on-cloud* planner according to the preference of the users. For instance, the robot is expected to move to the utility room during the lunch time. Then, the navigation goals are directly set by the navigation module from the robot's environmental perception resulting into an autonomous navigation. For instance, this modality is activated when the robot moves to the charging basis (i.e., the goal corresponds to the charging basis position) or when the robot is required to look for the person to monitor and interact with him/her (i.e., the goal corresponds to the target person). Finally, in the proposed architecture, the navigation goals for the robot can be achieved via the communication between the navigation module with the one handling the interaction (see Figure 1). Specifically, the navigation goals are resulted from the fusion of the context information with the input received from the user (e.g., turning commands for the robot). This modality, called shared navigation, aims facilitating the robot's control during the telepresence task where a secondary user remotely teleoperates the robot. However, it is worth noticing that this modality differentiates from the traditional manual teleoperation in which the robot exactly reproduces the user's commands. Indeed, also in the shared navigation, the robot is in charge to autonomously compute a navigation goal that is continuously updated while it is moving and in particular when the robot receives a new user's command. Please refer to (Beraldo et al. 2021; Beraldo, Tonin, and Menegatti 2021) for further details.

In all the modalities, the navigation module relies on a module of Simultaneous Localization and Mapping (SLAM)<sup>5</sup> that creates/exploits a representation of the environment (i.e., map) in real-time from the robot's sensors and localizes the robot inside. The SLAM modules is included not only to improve the reliability of the navigation

<sup>4</sup>The navigation module is based on the ROS package <http://wiki.ros.org/navigation>

<sup>5</sup>SLAM modules relies on the ROS packages: [http://wiki.ros.org/slam\\_gmapping](http://wiki.ros.org/slam_gmapping) and <http://wiki.ros.org/amcl>



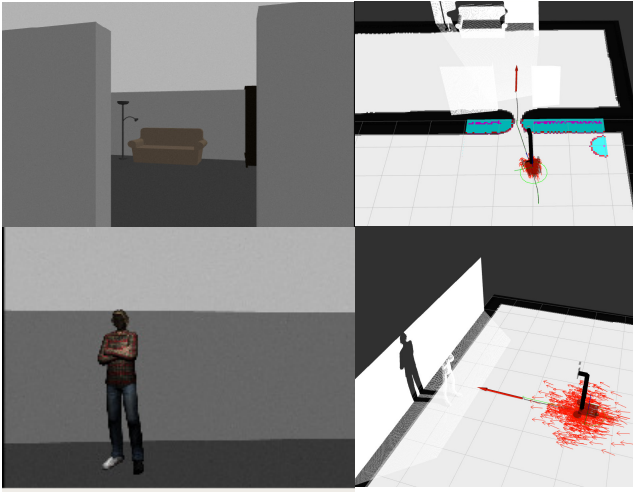


Figure 3: An illustrative example of the robot’s navigation service (tested in simulation). In the first case, the robot is autonomously moving towards the goal position chosen by the *on-cloud* planner. In the second, the robot sets the navigation goal in the proximity of the detected person. In both situations, the robot exploits a map of the environment on which it estimates the position thanks to the SLAM functionality.

system, but also to enable the robot to reach navigation goals far from its position (e.g., from a room to another), as commonly required in telepresence applications. An example of the navigation task is shown in Figure 3.

Finally, the current architecture includes multimodal services oriented to the two main activities for the elderly monitoring: the cognitive and the physical exercises and the measure of the physiological parameters. Relying on *text to speech* and *speech to text* functionalities for human-robot dialogues, these services enhance the concepts described in (De Benedictis et al. 2020) by introducing a vision-based module to detect people. As described previously, specifically, the topic of the robot’s dialogue is established by the *cloud-layer*, while the reactive services deal with managing the dialogue with the user<sup>6</sup>, reproducing the personalised sentences from the robot and interfacing with the microphones to acquire and process human voices. The vocal interaction is triggered from both the *cloud* and the *reactive*<sup>7</sup> layers. The first happens during the execution of a cognitive exercise. An example of cognitive exercise consists of requiring the user to count the occurrences of a specific word in a list uttered by the robot. The second is triggered when a person is detected in the neighborhood of the robot. In the second case, we assume the robot has to be ready to interact with person. Therefore, the robot requires the permission of activating the microphone and waits for user’s confirma-

<sup>6</sup>The reactive layer manages natural language understanding and generation by interfacing with a RASA server (Bocklisch et al. 2017).

<sup>7</sup>The reactive layer wrappers the Google Cloud Speech API in a ROS package: <https://cloud.google.com/?hl=it>

tion. The detection of the person is managed by a skeleton tracker based on a RGB-D camera mounted on the robot: specifically the keypoints are computed on the RGB image, while the depth is used to estimate the distance between the person and the robot. The keypoints returned by the skeleton detection module<sup>8</sup> are also used to compute the center of gravity of the person (from camera to robot frame) for setting navigation goal when robot needs reaching the person (for instance in Figure 3). Moreover, the skeleton detection

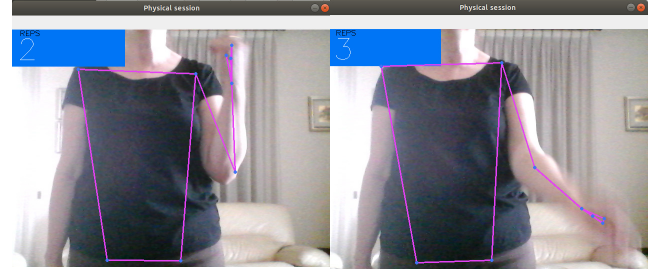


Figure 4: An explanatory demonstration of a session of physical exercises. The person is required to raise and stretch the left arm. In the while, the robot monitors the involved joints and counts the number of repetitions.

becomes essential to monitor physical exercises. An example is shown in Figure 4. During the execution of this activity, the robot processes the estimated joints positions by computing the relative angles according to the exercise and compares them with the expected range in accordance with a reference template. By evaluating the change in the position and in rotation of the joints, the robot is also exploited as a companion to count the number of repetitions as well as to encourage the person during the execution.

Finally, since the importance of prevention and elderly care, the reactive services involve the robot to motivate the user to measure physiological parameters (e.g., pressure, heart rate, the temperature) at specific time planned by the *on-cloud* layer or during the breaks between one repetition and another during the physical exercises. The robot is also in charge of transmitting the sensor’s readings to the *cloud*.

### The adaptive implementation of the reactive behaviors

It is worth explicitly clarifying that the reactive layer does not only merely execute the activities proposed by the *on-cloud* planner but it is in charge of defining the detailed low-level robot’s behaviors. Let discuss this aspect by taking one simple example among the kind of interaction included in our architecture. A more detailed overview is shown through the pseudo-code in Algorithm 1.

You can imagine that at 4 pm the scheduler provides the start of physical exercises. The robot is notified with a message of the starting of the corresponding activity. Thus, the robot is required to move towards a specific position in the environment  $(x_{pe}, y_{pe})$  established in accordance with the

<sup>8</sup>The skeleton detection module encapsulates the MediaPipe library <https://mediapipe.dev/> in a ROS package



user's preference. The navigation module is triggered by the *local task manager* and the best trajectory from the current position to the goal (i.e.,  $(x_{pe}, y_{pe})$  the coordinates provided by the cloud) is computed by a motion planner inside the navigation module. Once reached the goal, before starting the physical exercise, it is necessary the robot ensures of the presence of the person around (see Algorithm 1). Therefore, the *local task manager* requires the activation of the *skeleton tracker*. Only if the skeleton tracker returns the presence of the keypoints with a quite high confidence (e.g.,  $>0.7$ ), the *local task manager* starts the execution of the physical exercise. Otherwise, more interestingly, an unexpected situation for the plan occurs that the robot reactively has to face. With this purpose, the robot autonomously navigates in the environment to find the person. The autonomous navigation modality is waked again and the skeleton tracker remains active. Once the person is detected, this time is the *skeleton tracker* that stops the navigation service and activates the interaction module to greet the person and reminder the physical exercises session.

Although this example is very simple, it clearly suggests how a successful human-robot interaction can be achieved only by the mutual and continuous interchange of information between the two layers.

## Discussion

Nothing forces us to follow the footsteps of psychology, however forms of causal semantic reasoning mimic very well our system 2, more logical and reflective but slower, while, on the contrary, policy-based approaches create behaviors more similar to our system 1, more impulsive and intuitive but faster in making its decisions. In creating a robotic system that interacts with people in a constantly evolving environment, it seemed intuitive to follow a comparison with the two systems. This separation of responsibilities, however, paves the way to the problem of which system has to perform the different tasks.

The benefits of system 2 described in this work are the correctness guaranteed by the rules, its simpler explainability, the possibility of carrying out mathematical reasoning (for example, ordering the different activities over time) and the ability to pursue multiple goals at the same time. Furthermore, the different tasks are organized by the reasoner who chooses and schedules them, pursuing objectives and respecting constraints that may change from time to time. The reasoning process, however, in its most general form, has undecidable complexity and, even in its computationally simpler forms, takes time to generate solutions. Furthermore, the result of the deliberative module is a plan, i.e. a relatively rigid structure which, with a few exceptions (mostly temporal adjustments), hardly adapts to the reality that emerges dynamically.

The system 1 described in this work has almost diametrically opposite characteristics. For this reason, in fact, the integration of the two approaches seems profitable to us. In organizing tasks, in particular, system 1 requires the manual definition of a state machine or, if possible, the need to interact with the environment to learn which actions to perform in which states. This process becomes easily complex and error

---

### Algorithm 1: Reactive Human-Robot Interaction

---

```

while Physical Exercise do
  Go to position  $(x_{pe}, y_{pe})$ 
  Activate skeleton tracker
  if Person detected then
    Reminder of the physical session
    Activate microphone
    Start the physical session
    Processing keypoints
    Physiological Monitoring
  else
    Activate navigation module
    while Robot navigation do
      if Person detected then
        | break
      end
    end
  end
end

end
while Cognitive Exercise do
  Go to position  $(x_{ce}, y_{ce})$ 
  Activate skeleton tracker
  if Person detected then
    Reminder of the cognitive session
    Activate microphone
    Start the cognitive session
    Processing voice
  else
    Activate navigation module
    while Robot navigation do
      if Person detected then
        | break
      end
    end
  end
end

end
while Physiological Monitoring do
  Go to position  $(x_{pm}, y_{pm})$ 
  Activate skeleton tracker
  if Person detected then
    Request to measure specific parameter
    Activate microphone
    Processing voice
    Register the value
  else
    Activate navigation module
    while Robot navigation do
      if Person detected then
        | break
      end
    end
  end
end
end

```

---

prone, in the first case, as the number of tasks and variables that describe the environment increase, while, in the second case, it requires safe environments for the robot and for the people who live within them, as well as times potentially very huge for training (only in very simple cases, indeed, it is possible to exploit reliable simulators that solve, at least in part, the problems of safety and learning time). Except in some very specific cases (e.g., navigation tasks), moreover, system 1 does not behave on the basis of goals that may come from the outside (e.g., from an operator). Such goals, furthermore, are strictly related to the specific kind of task (e.g., navigation tasks can reach only poses (i.e., position and orientation). Furthermore, the system 1 is not very flexible in carrying out tasks of a logical/mathematical type. Once the policy has been defined (or learned), however, it easily adapts to the environment surrounding the robot, with its unpredictable dynamic evolution.

In general, it is not easy to distribute responsibilities between the two systems. There is, as far as we know, a precise rule to follow. In building our system, specifically, we used the following heuristic. As suggested by Kahneman's countless experiments, in particular, system 2 believes itself to be the protagonist of the story but is, on the contrary, a rather secondary actor who rarely intervenes in decisions. For this reason, the approach we followed in assigning responsibilities was to assign as many responsibilities to system 1 as possible, so as to ensure a quick adaptation to dynamic changes. In all those cases in which long sequences of activities depend on parameters linked to the profile of the users and, above all, in all those cases in which forms of temporal reasoning were required, the responsibility falls on the system 2.

## Conclusion

This paper aims presenting a combination of adaptive planning with reactive services on board the robot. In particular, we want to stress that the motivation is to design a system able to capture and manage the uncertainty of environments, such as the domestic one, that dynamically and continuously evolve over time. In this context, on the one side the planning technologies typically suffer on limited capability of easily and flexibility introducing modifications to the generated plans, on the other, pure reactive behaviors are able to implement only basic forms of semantic reasoning, that hardly considers the causal and temporal relations. For these reasons, we strongly promote merging the two approaches to make the robotic platform more reliable and functional for elderly people's needs in the perspective of facilitating a longer and more safe life at home.

## Acknowledgments

Authors are partially supported by the Italian "Ministero dell'Università e della Ricerca" under the project "SI-ROBOTICS: SocIal ROBOTICS for active and healthy ageing" (PON 676 – Ricerca e Innovazione 2014-2020—G.A. ARS01\_01120) and by CNR under FOE\_2020 Strategic Project "Technologies to support the most vulnerable groups: young and old – CLEVERNESS".

## References

- Apt, K. R., and Wallace, M. G. 2007. *Constraint Logic Programming Using ECL<sup>i</sup>PS<sup>e</sup>*. New York, NY, USA: Cambridge University Press.
- Augello, A.; Infantino, I.; Lieto, A.; Maniscalco, U.; Pilato, G.; and Vella, F. 2017. Towards A Dual Process Approach to Computational Explanation in Human-Robot Social Interaction. In *Proceedings of the 1st CAID workshop at IJCAI*.
- Beraldo, G.; Tonin, L.; Cesta, A.; and Menegatti, E. 2021. Brain-driven telepresence robots: A fusion of user's commands with robot's intelligence. In Baldoni, M., and Bandini, S., eds., *AIxIA 2020 – Advances in Artificial Intelligence*, 235–248. Cham: Springer International Publishing.
- Beraldo, G.; Tonin, Luca 2021. Shared intelligence for user-supervised robots: From user's commands to robot's actions. In Baldoni, M., and Bandini, S., eds., *AIxIA 2020 – Advances in Artificial Intelligence*, 457–465. Cham: Springer International Publishing.
- Bocklisch, T.; Faulkner, J.; Pawlowski, N.; and Nichol, A. 2017. Rasa: Open Source Language Understanding and Dialogue Management. *arXiv preprint arXiv:1712.05181*.
- Cesta, A.; Cortellessa, G.; Fracasso, F.; Orlandini, A.; and Turno, M. 2018. User needs and preferences on AAL systems that support older adults and their carers. *J. Ambient Intell. Smart Environ.* 10(1):49–70.
- De Benedictis, R., and Cesta, A. 2020. Lifted Heuristics for Timeline-based Planning. In *ECAI-2020, 24th European Conference on Artificial Intelligence*.
- De Benedictis, R.; Umbrico, A.; Fracasso, F.; Cortellessa, G.; Orlandini, A.; and Cesta, A. 2020. A two-layered approach to adaptive dialogues for robotic assistance. In *2020 29th IEEE International Conference on Robot and Human Interactive Communication (RO-MAN)*, 82–89.
- Dechter, R. 2003. *Constraint Processing*. Elsevier Morgan Kaufmann.
- Group, A. W., et al. 2018. The 2018 ageing report underlying assumptions and projection methodologies. *Economy, finance and the euro publications*.
- Isabet, B.; Pino, M.; Lewis, M.; Benveniste, S.; and Rigaud, A.-S. 2021. Social Telepresence Robots: A Narrative Review of Experiments Involving Older Adults before and during the COVID-19 Pandemic. *International Journal of Environmental Research and Public Health* 18(7).
- James, W. 1890. *The Principles of Psychology, in two volumes*. New York: Henry Holt and Company.
- Kahneman, D. 2011. *Thinking, Fast and Slow*. New York: Farrar, Straus and Giroux.
- Kristoffersson, A.; Coradeschi, S.; and Loutfi, A. 2013. A review of mobile robotic telepresence. *Advances in human-computer interaction* 2013:1–17.
- Laniel, S.; Létourneau, D.; Grondin, F.; Labbé, M.; Ferland, F.; and Michaud, F. 2021. Toward enhancing the autonomy of a telepresence mobile robot for remote home care assistance. *Paladyn, Journal of Behavioral Robotics* 12(1):214–237.

- Lecoutre, C. 2009. *Constraint Networks: Techniques and Algorithms*. Wiley-IEEE Press.
- Melendez-Fernandez, F.; Galindo, C.; and Gonzalez-Jimenez, J. 2017. A web-based solution for robotic telepresence. *International Journal of Advanced Robotic Systems* 14(6):1729881417743738.
- Mois, G., and Beer, J. M. 2020. The role of health-care robotics in providing support to older adults: a socio-ecological perspective. *Current Geriatrics Reports*.
- Moro, C.; Nejat, G.; and Mihailidis, A. 2018. Learning and personalizing socially assistive robot behaviors to aid with activities of daily living. *ACM Trans. Hum.-Robot Interact.* 7(2):15:1–15:25.
- Muscettola, N.; Smith, S.; Cesta, A.; and D'Aloisi, D. 1992. Coordinating Space Telescope Operations in an Integrated Planning and Scheduling Architecture. *IEEE Control Systems* 12.
- Orlandini, A.; Kristoffersson, A.; Almquist, L.; Björkman, P.; Cesta, A.; Cortellessa, G.; Galindo, C.; Gonzalez-Jimenez, J.; Gustafsson, K.; Kiselev, A.; Loutfi, A.; Melendez, F.; Nilsson, M.; Hedman, L. O.; Odontidou, E.; Ruiz-Sarmiento, J.-R.; Scherlund, M.; Tiberio, L.; von Rump, S.; and Coradeschi, S. 2016. ExCITE Project: A Review of Forty-Two Months of Robotic Telepresence Technology Evolution. *Presence* 25(3):204–221.
- Petty, R. E., and Cacioppo, J. T. 1986. The Elaboration Likelihood Model of Persuasion. In *Advances in Experimental Social Psychology*. Elsevier. 123–205.
- Pollack, M. E. 2005. Intelligent technology for an aging population: The use of AI to assist elders with cognitive impairment. *AI Magazine* 26(2):9.
- Riano, L.; Burbridge, C.; and Mc Ginnity, M. 2011. A study of enhanced robot autonomy in telepresence. In *Proceedings of Artificial Intelligence and Cognitive Systems, AICS ; Conference date: 31-08-2011*, 271–283. Ireland: AICS.
- Rossi, S.; Ferland, F.; and Tapus, A. 2017. User profiling and behavioral adaptation for HRI: A survey. *Pattern Recognition Letters* 99:3 – 12.
- Sheridan, T. B. 1992. Musings on Telepresence and Virtual Presence. *Presence: Teleoperators and Virtual Environments* 1(1):120–126.
- Strannegård, C.; von Haugwitz, R.; Wessberg, J.; and Balke-nius, C. 2013. A Cognitive Architecture Based on Dual Process Theory. In *Artificial General Intelligence*. Springer Berlin Heidelberg. 140–149.
- Sun, R. 2015. *The CLARION Cognitive Architecture*. Oxford University Press.
- Tsui, K. M.; Desai, M.; Yanco, H. A.; and Uhlik, C. 2011. Exploring use cases for telepresence robots. In *2011 6th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, 11–18.
- Wason, P., and Evans, J. 1974. Dual processes in reasoning? *Cognition* 3(2):141–154.