POLITECNICO DI TORINO
Repository ISTITUZIONALE

# Application of a Hierarchical Task Planner to a Lunar Lava Tube Analogue Robotic Mission

*Original*
Application of a Hierarchical Task Planner to a Lunar Lava Tube Analogue Robotic Mission / Rimani, J., Viola, N., Lizy-Destrez, S.. - ELETTRONICO. - D1-IAF SPACE SYSTEMS SYMPOSIUM:(2021). (72nd International Astronautical Congress Dubai October 2021).

*Availability:*
This version is available at: 11583/2935111 since: 2026-01-05T17:17:12Z

*Publisher:*
International Astronautical Federation (IAF)

*Published*
DOI:

*Terms of use:*

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

*Publisher copyright*

(Article begins on next page)

27 October 2026

Authors' Version

# APPLICATION OF A HIERARCHICAL TASK PLANNER TO A LUNAR LAVA TUBE ANALOGUE ROBOTIC MISSION

**Jasmine Rimani**[1, *], **Nicole Viola**[2], and **Stéphanie Lizy-Destrez**[3]

[1] *Department of Mechanical and Aerospace Engineering, Politecnico di Torino, Torino, Italy,* Email: jasmine.rimani@polito.it , *Department of Aerospace Vehicles Designs and Control, ISAE-SUPAERO, Toulouse, France,* Email:jasmine.rimani@isae-supaero.fr

[2] *Department of Aerospace Vehicles Designs and Control, ISAE-SUPAERO, Toulouse, France,* Email: stephanie.lizy-destrez@isae-supaero.fr

[3] *Department of Mechanical and Aerospace Engineering, Politecnico di Torino, Torino, Italy,* Email:nicole.viola@polito.it

[*]Corresponding author
Email: jasmine.rimani@polito.it

**Abstract**

As missions grow more complex and daring, the concept of autonomous systems gains more importance. One of the most innovative missions is related to lunar lava tube exploration. Those volcanic architectures should be a natural shelter for the future manned mission in the equatorial sector of the Moon. Nevertheless, direct communication between Earth and the inside of the lava tube is difficult. Therefore, the robotic precursor exploration mission should be autonomous. The systems should move in an unknown environment, plan their tasks autonomously, decide the best course of action given their available resources, and communicate their results back to Earth. The aim of this study, conducted by Politecnico di Torino and ISAE-SUPAERO, is to define a structure that links operations, hierarchical task planning, path planning, localization and mapping to provide autonomous navigation capabilities for the lava tube exploration. The case study is a rover and a drone tested during the IGLUNA ESALab@CH Field Campaign, an activity coordinated by Space Innovation (Previous Swiss Space Center). The rover mission should explore an unknown environment reaching different targets. The rover is equipped with a LiDAR and two cameras, one for obstacle avoidance and one for mapping and localizing. The mission set-up will be similar to the one designed for exploring the lava tubes where a series of rovers and thruster-flying/hopping bots will join forces to provide information on their environment to a control centre. Therefore, during the analogue mission, the rover will be helped by a small drone, equipped with a tracking camera and LiDAR, capable of performing short flights of a few minutes to simulate the lunar bot. The systems will be in contact with a control centre far from the testing field. Moreover, the communication set-up will emulate a lunar one in bandwidth and communication delay. The tasks processed by the rover will be split between the two systems. The drone will provide a global map to the rover. This map will define the goals and provide their position with respect to the rover starting point. A hierarchical task planner would then acquire the list of goals and evaluate the best plan to reach them while monitoring both systems' available resources. It will monitor the mission execution, avoid obstacles, and the number path or plan re-evaluations. In this paper, the hierarchical task planner's performance and its integration with the mapping, path planning, and path execution modules will be presented.

**Keywords:** .

## 1. Introduction

During Earth-orbiting missions, periodic high-quality communications with ground stations are guaranteed for both scientific and housekeeping purposes. This is not true for deep-space or underground planetary missions, where the quality of the communications links can be extremely poor. Nevertheless, missions are growing more complex, and the capability of planning, controlling, and predicting all potential mission events decreases. Autonomous decision-making capabilities onboard the future exploration system are, therefore, considered critical technologies for this new era of space exploration. That aim can be achieved thanks to the technological advances of the last decade in artificial intelligence [1]. In this context, space agencies, companies, and universities are engaged in the definition of a broad spectrum of technological maturation studies toward autonomous operations and navigation [2] [1] [3] [4]

One of the most interesting and challenging missions is exploring planetary lava tubes. In particular, this paper focuses on lunar lava tubes. Lunar lava tubes are volcanic structures that unfold beneath the surface of our natural satellite. They can be accessed from the skylights, portions of their ceiling that have collapsed. Three main lava tubes have been detected on the lunar surface [5]: one in Marius Hills, one in Mare Tranquillitatis, and one in Mare Ingenii.

On the one hand, they are particularly interesting for studying the geological and mineralogical history of the Moon. On the other hand, they may serve as a perfect shelter for future human missions in the equatorial region of the Moon. Indeed, scientists have hypothesized that those volcanic tunnels may offer an additional layer of protection for astronauts against radiation and micrometeoroids.

However, before any human mission can be allowed in the lava tubes, a robotic exploration mission is envisioned. The exploration systems should be able to autonomously map, navigate, and operate within the tubes, as well as relay sensitive data to Earth.

This study starts from the need for autonomy to define an integrated architecture that combines mapping, localization, path planning, and on-board decision-making for robotic exploration systems.

The decision-making algorithm leverages state-of-the-art symbolic AI methods to generate a sequence of actions for the systems to execute. Those actions are then interfaced with a state machine that connects to the system's sensors and actuators to execute the plan.

The integrated architecture has been tested and validated during the IGLUNA Analog mission with the CoRoDro team [1], an ESALab@CH initiative coordinated by the Space Innovation [2].

## 2. Case Study

The task planning algorithm presented in this paper has been tested during the IGLUNA analog mission field campaign. During the mission, two systems, a rover and a drone, collaborated to explore an unknown environment. The testing has been conducted in France at the ONERA test site near Toulouse. However, the entire operational chain was tested remotely, relying on a network that simulates Moon-Earth communication. The mission was divided into two main phases:

- Recognition phase: during the recognition phase, the drone was flying a pre-planned path, mapping the environment, and recognizing points of interest.
- Mission phase: during the mission phase, the drone and the rover began exploring the unknown environment simultaneously, visiting different points of interest, taking pictures, and reading the artags.

[1] https://youtu.be/HWYhED8Z17Y
[2] https://space-innovation.ch/

During both phases, the systems were autonomous: the objective was to test the algorithms and the strategies that would permit reaching the level of autonomy required to explore the lava tubes. More in detail, the objectives of the study were to (i) demonstrate the feasibility of autonomous navigation in an unknown field, (ii) study the collaboration between systems with different mobility, and (iii) optimize the planning of tasks during remote operations. Moreover, during testing, we used an "Earth drone" to simulate the operations of a lunar bot. During the lunar mission, the small propelled bot will fly to map and explore the environment before the rover. The same bot will use propelled flight to slow its descent through the lava tubes while mapping the skylights. In the lava tube, the bot may decide to perform short flights to map the environment better. Therefore, the drone's flights were kept very short, around 3 minutes. The lunar bot should have an autonomy of around 45 minutes [6]. The concepts of operations of the analog mission are illustrated in Figure 1. The drone is controlled in position: the equipped tracking camera provides information on coordinates (x, y) with respect to the "zero point". The "zero point" can be the origin of the coordinate system during the mapping phase or a new position set with the ARTag linker during the mission phase. To be more precise, during the mission phase, the system can reset its notion of position relative to the "point of interest" it sees. Neither the drone nor the rover uses GPS technology; the only position information comes from the odometry and SLAM algorithms in the T265 camera. Before the field test campaign, we conducted precision tests on the tracking camera. The maximum percentage difference between the optitrack and the camera data is 2% in height position. That difference is primarily due to the position of the nearest optitrack beacon to the camera. However, we are mostly interested in the system's coordinates (x, y) during its flight, and in that case, the difference drops to less than 1%. The rover mounts the same bundle of cameras, a D435i and T265. In this case, the movements are less sudden; therefore, the (x,y) positions are very precise. Again, for the z-axis, there are some differences: the camera is 30 cm above the rover's body. Therefore, the T265 is indeed at the zero position, as indicated by a 30 cm displacement from the optitrack.

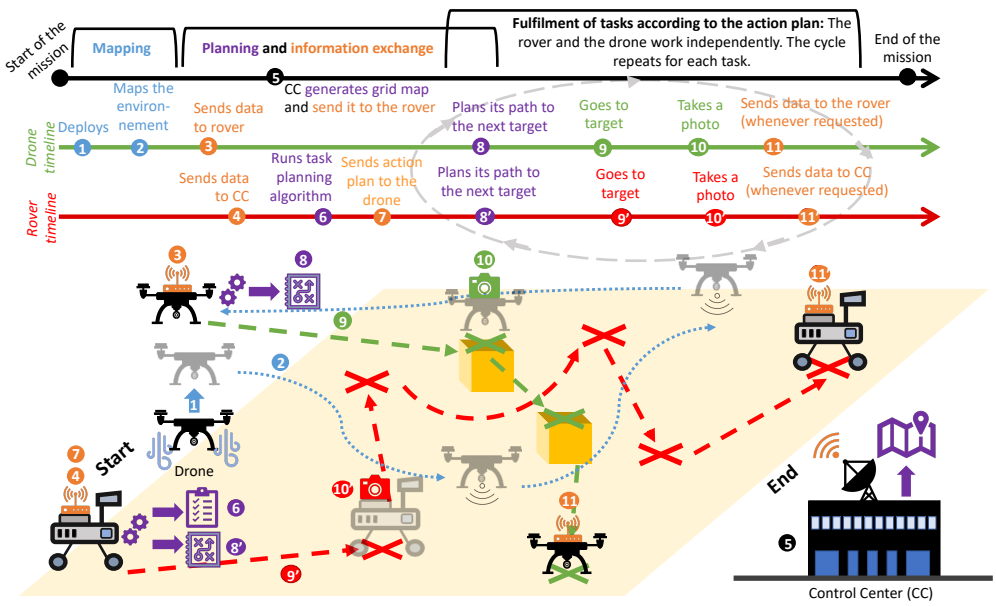


*Figure 1: Concept of Operations of the CORODRO team during the IGLUNA mission*

The used systems are shown in Figure 2 and their components are described in Tables 1 and 2.

The general dimensions of the systems were respectively:

- 0.067 $m^3$ (0.42 x0.40 x0.40) $m^3$ for the rover,
- and 0.088 $m^3$ (0.57x0.57x0.27) $m^3$ for the drone.

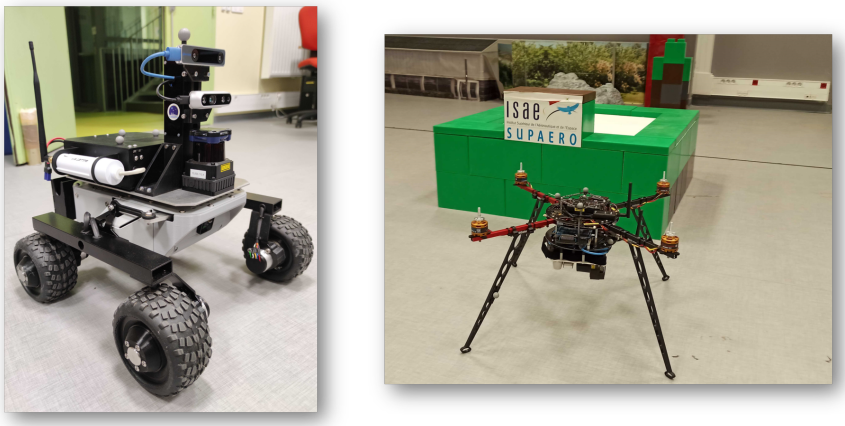


*Figure 2: Rover and drone that will be used during the IGLUNA Mission*

*Table 1: Drone equipment list*

| **Component** | **Subsystem** | **Reference** | **Key specs** |
|---|---|---|---|
| Cameras | Depth Camera<br>Tracking Camera | Intel D435i<br>Intel T265 | Max frame rate: 90 fps<br>Max frame rate: 120 fps |
| Communication | Radio Receiver<br>WiFi System | Futaba R6208 SB<br>Odroid WiFi Module | Frequency: 2.4 GHz<br>Transfer speed: around 200 Mbps |
| Computer | Onboard Computer | ODROID XU4 | – |
| Avionics | Flight Controller<br>ESCs | 3DR Pixhawk 3<br>BL-Ctrl V3.0 | –<br>– |
| Motors<br>Power System | 4 Motors<br>Battery | –<br>LiPo 4S | –<br>Capacity: 3700 mAh |

## 3. Task Planning Algorithm

Task planning is the discipline that considers how the tasks, the actions a robotic system must accomplish, are distributed in time. It usually outputs a sequence of actions that can be interfaced directly to the system through an execution layer, e.g., a state machine. In the specific case of the IGLUNA analog mission, HDDL (Hierarchical Definition Domain Language) [7] was used. HDDL is heavily based on PDDL (Problem Definition Domain Language); however, it introduces a hierarchical structure into the game. Both HDDL and PDDL are in the domain of *symbolic AI* and are based on ontologies, which are knowledge systems in AI [8]. Hierarchical planners perform a set of actions at different levels of abstraction [9]. The main objective is not directly achieving a goal state. More in detail, the goal objectives are derived by refining the initial task network [10], yielding an executable sequence of actions. Many tasks in real life are already organized in hierarchical structures [10], therefore, the use of HDDL naturally arises from the functional analysis of

*Table 2: Rover components list*

| Component | Subsystem | Reference | Key specs |
|---|---|---|---|
| Sensors | Leo Rover Camera | HOKUYO UST-20LX | Max frame rate: 40 fps |
| | Depth Camera | Intel D435i | Max frame rate: 90 fps |
| | Tracking Camera | Intel T265 | Max frame rate: 120 fps |
| | Lidar | HOKUYO UTM-30 LX-EW | Frame rate: 50 fps |
| Communication | WiFi Internal (RPi Antenna) | – | 2.4/5 GHz |
| | WiFi Access Point | – | 2.4 GHz |
| | WiFi Upper Rover System | – | – |
| Computer | Onboard (Leo Rover) | Raspberry Pi 3B+ | 0.4 GHz CPU |
| | Upper Rover | ODROID XU4 | – |
| Controller | – | Husarion CORE2-ROS | – |
| Mobility System | Wheels | – | Diameter: 130 mm |
| | Tires | – | Rubber with foam insert |
| | Wheel Actuators | Buehler DC Motors | – |
| | Wheel Encoders | Pololu Romi 12 CPR | Magnetic encoders |
| IMU | – | SBG IG 500A-G5A2P1-P | – |
| Power System | Rover Battery | Li-Ion with PCM | 5000 mAh |
| | Upper Battery | Kunzer MPB 150 | 15000 mAh |

what the system is expected to do during the mission. Moreover, the IGLUNA mission has been modeled using MBSE (Model-Based System Engineering). Therefore, the task's natural structure is hierarchical. Hence, it is possible to directly translate the MBSE model into the HDDL model of IGLUNA [11] using Python code. In our case study, the task-planning software defines the overall behavior of the two collaborative systems. It defines: (i) when to execute the different tasks, (ii) the order of the different waypoints to visit during the mission, and (iii) the overall behaviors of the rover and the drone in the different situations. The task planner software has six main modules that span from the decision-making layer to the executive one. A general description of the modules of the HDDL planning architecture can be found in Table 3.

- Problem File Module, "problem.hddl" - This module groups all the available information on the environment where the system is moving and its goals.
- Domain File Module, "domain.hddl" - This module outlines all the possible actions that can be executed by the system. It considers even a cluster of possible actions triggered by particular situations.
- HDDL Plan Generation Module - The module will generate the plan to be executed by the systems: (i) when to move, (ii) when to communicate, (iii) with whom to communicate, (iv) in which order to reach each of the waypoints, etc.
- Plan Dispatch Module - The module will parse the just-generated plan and interface it with a State Machine.
- Plan Execution Module - The module will execute the plan generated by the Plan Dispatch Module using the Action Library of ROS.
- Actions Feedback Module - The module will store and manage sensor inputs. It will supply

the readings to the Problem File if new obstacles are detected, if a waypoint cannot be reached, etc., to recompute the overall plan.

| Module | Description | Coding Language | Internal Interfaces | External Interfaces |
|---|---|---|---|---|
| Problem File Module | This module groups all the available information on the environment where the system is moving | HDDL | INPUTS:<br>- Sensor readings supplied the Action Feedback Module<br>- Initial generated grid map and position of the systems in the map<br>OUTPUTS:<br>- HDDL Plan Generation Module | None |
| Domain File Module | This module outlines all the possible actions that can be executed by the system<br>It considers even a cluster of possible actions that can be triggered by particular situations | HDDL | OUTPUTS:<br>- HDDL Plan Generation Module | INPUTS:<br>- Expected behavior of the system<br>- Behavior of the system from testing |
| HDDK Plan Generation Module | The module will generate the plan to be executed by the systems: (i) when to move, (ii) when to communicate, (iii) with whom to communicate, (iv) in which order to reach each of the waypoints, etc. | ".txt" File | INPUTS:<br>- Domain file<br>- Problem file<br>OUTPUTS:<br>- ".txt" file to be parsed | None |
| Plan Dispatch Module | The module will parse the ".txt" file and interface it with a State Machine | Python | INPUTS:<br>- ".txt" file from the HDDL Plan Generation Module | INPUTS:<br>- None for the rover<br>- WiFi-like network for the drone to receive the plan generated by the rover |
| Plan Execution Module | The module will execute the plan generated by the Plan Dispatch Module using the Action Library of ROS | Python | INPUTS:<br>- State machine generated by the Plan Dispatch Module | None |
| Actions Feedback Module | The module will store and manage the inputs from the sensors<br>It will supply the readings to the Problem File if new obstacles are detected or if a way-point cannot be reached | Python | INPUTS:<br>- Sensor readings from ROS topics and services | INPUTS:<br>- None for the rover<br>- WiFi-like network for the drone to send its sensors readings to the rover |

*Table 3: Summary of the HDDL software modules, description, coding language, and interfaces*

Figure 4 shows the block diagram of the envisioned task planner. The software is interfaced with path planning, mission analysis, and localization and mapping as follows:

- The ConOps and design of the mission from Mission Analysis is coded in the domain file, starting from the MSBE model. The starting point for the code is the Enhanced Functional Flow block Diagram (EFFBD) for the drone and rover mission, shown in Fig. 3.
- The occupancy grid from Localization and Real-Time Mapping is translated into the "initial conditions" of the "problem.hddl" file. We coded an interface that, from the grid map, can output the required predicates for the HDDL problem file.
- The Action Library of ROS is configured for both the rover and the drone by the Path Planning team.
- The SMACH Monitor State of ROS [12] to monitor the sensors topic and stop, change, or reconfigure the plan of the rover. If there is an anomaly on the drone, we are just going to stop it —hard failure —or make it go to the recharging station —manageable failure.
- The interface for the control center is implemented as a "visual" state machine that is easy to follow for the control center operator.

The HDDL *plan solver* is implemented in Python and provided by our sponsor, ONERA. It is called HiPOP [13] [14]. This code outputs a plan saved as a ".txt" file. This plan is then parsed by a Python code module implemented as a ROS node. Then, this ROS node is interfaced with ROS Smach [12], the ROS library for implementing and visualizing state machines. The last step is the robotic platform's effective action: the robotic system's actions are followed by logging information displayed in the terminal. The logging information will be readily visible to the control center operators during the field campaign. However, a state machine-based visualization is preferred because it is easier to understand and follow.

As explained, the AI planner exploits the HDDL modeling language. The HDDL language is heavily based on PDDL, only introducing the hierarchical component. The possibility of defining

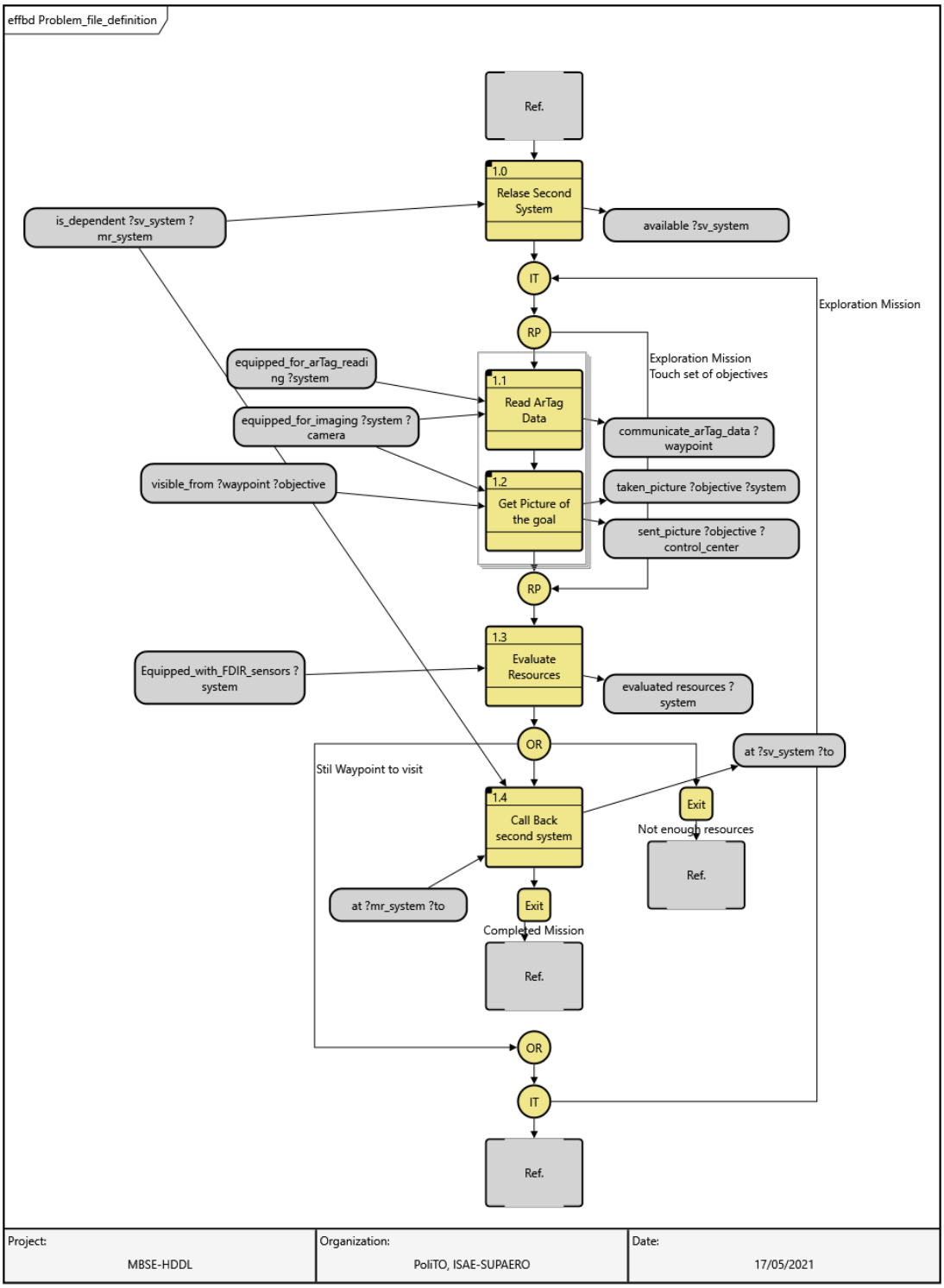


*Figure 3: EFFBD Igluna*

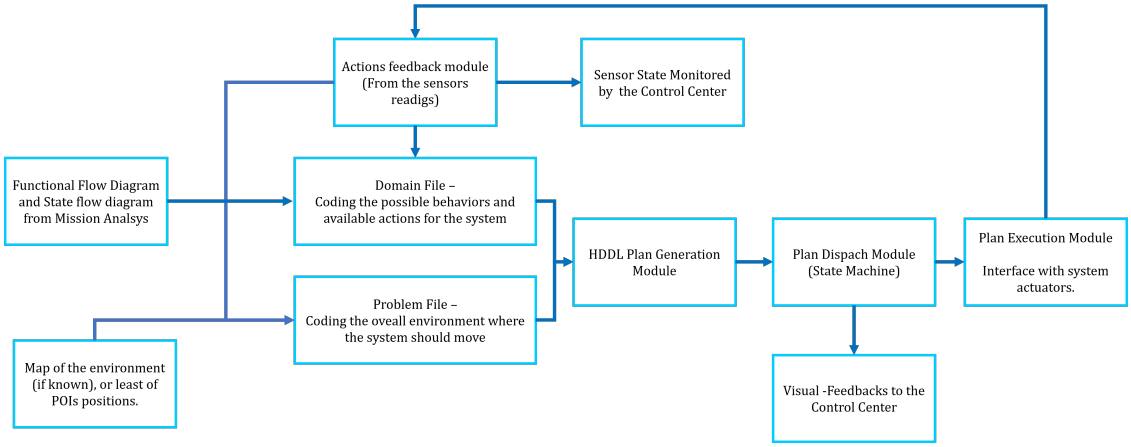


*Figure 4: Task planner block diagram*

a hierarchy of functions accomplished by the system allows the description of complex scenarios. A clear and straightforward explanation of both HDDL and PDDL can be found in [15]. The important concepts to read and understand HDDL files are:

- *:task()* - those entries define what we expected the system under study to be able to perform.
- *:method()* - those entries define how we can accomplish a *:task()* through ordered sub-tasks. These subtasks can then be *compound tasks* or *primitive tasks*. The first are subtasks that other methods can further decompose. The latter are subtasks with a direct effect on the system. They are usually called *:action()*.
- *:action()* - those entries are the ones that change the state of the system using predicates, true or false statements.
- *:predicates()* - those entries are a true or false statement that describes the state of the system. Those are the sentences that effectively advance the plan.

The collaborative mission of the drone and rover is quite complex, making it perfect to model with hierarchy and HDDL. The objective is to optimize the tasks that both systems need to perform simultaneously. In the ideal scenario, the two systems would move simultaneously, collaborating to find targets, read the arTag, and take pictures of the objectives. The rover and the drone have different capabilities and viewpoints regarding obstacles and targets. These capabilities should be considered when evaluating their plan. That has been effectively added to the *problem.hddl* file indicating their visibility from the neighboring position with respect to the point of view of the observing system. To explain the use of HDDL, we will use a simple benchmark scenario shown in Figure 5. The scenario is a 3 m x 3 m terrain (discretized at 1 m) with three points of interest. Both systems start at *waypoint0*. The rover must activate the drone, but it can move freely within the scenario without path constraints. On the other hand, the rover cannot reach all the points due to the obstacles (colored squares). Therefore, the two systems should collaborate to accomplish the mission: visit all points of interest while optimizing the overall path length. So, the solver will try to find the overall shortest path to optimize the actions of both systems. The model used to move from one cell to another and to determine when the system can photograph an objective is shown in Figure 6. The predicates that describe movement and "visibility" of a point of interest are directly derived from the recognition phase grid map and arTags list.

Nevertheless, writing and testing HDDL files is usually quite a challenging task. Therefore, the analysis started from the mission analysis functional layer and ended in coding the *domain.hddl*

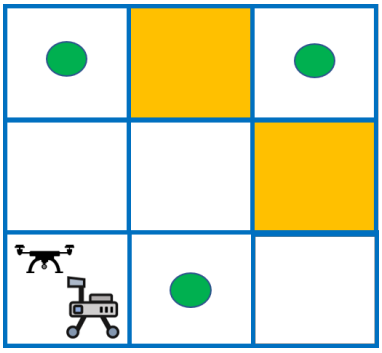

*Figure 5: Simple HDDL problem benchmark*

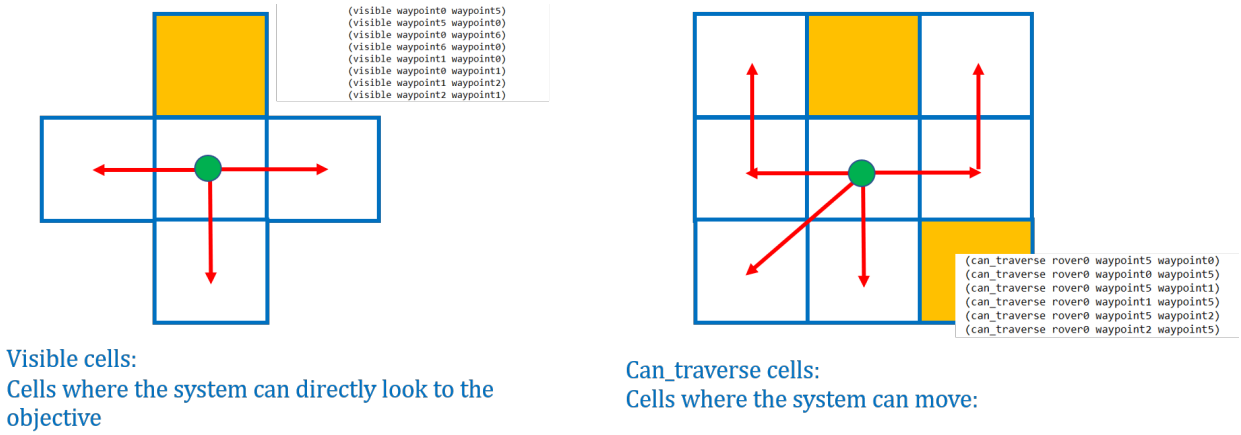


*Figure 6: Movement and targeting model in the HDDL problem file*

file.

The final resulting plan for the benchmark is shown in Listing 1.

*Listing 1: Complete plan based on the HDDL problem file*

```
(make_available drone1)
(visit waypoint0 drone1)
(navigate drone1 waypoint0 waypoint1)
(unvisit waypoint0 drone1)
(read_arTag rover0 waypoint0 objective2 camera0)
(communicate_arTag_data rover0 general objective2)
(take_image drone1 waypoint1 objective2 camera3 fisheye)
(communicate_image_data drone1 general objective2 fisheye)
(navigate drone1 waypoint1 waypoint4)
(visit waypoint4 drone1)
(navigate drone1 waypoint4 waypoint6)
(unvisit waypoint4 drone1)
(read_arTag drone1 waypoint6 objective0 camera2)
(communicate_arTag_data drone1 general objective0)
(take_image drone1 waypoint6 objective0 camera3 fisheye)
(communicate_image_data drone1 general objective0 fisheye)
(navigate drone1 waypoint6 waypoint4)
(visit waypoint4 drone1)
(navigate drone1 waypoint4 waypoint8)
```

```
(unvisit waypoint4 drone1)
(read_arTag drone1 waypoint8 objective1 camera2)
(communicate_arTag_data drone1 general objective1)
(take_image drone1 waypoint8 objective1 camera2 depth)
(communicate_image_data drone1 general objective1 depth)
(visit waypoint0 rover0)
(navigate rover0 waypoint0 waypoint5)
(unvisit waypoint0 rover0)
(read_arTag rover0 waypoint5 objective0 camera0)
(communicate_arTag_data rover0 general objective0)
(take_image rover0 waypoint5 objective0 camera0 depth)
(communicate_image_data rover0 general objective0 depth)
(navigate drone1 waypoint8 waypoint4)
(visit waypoint4 drone1)
(navigate drone1 waypoint4 waypoint5)
(unvisit waypoint4 drone1)
(get_data_from_sensors rover0)
(send_system_state rover0 general)
(get_data_from_sensors drone1)
(send_system_state drone1 general)
```

Even in a simple scenario like the benchmark, 38 steps are needed to achieve the envisioned goal.

### *3.1 State Machine*

Starting from the plan generated by the HDD solver, it is possible to rapidly code the system's state machine. This state machine is the heart of the *Plan Execution Module*. Figure 7 shows the state machine for the rover mission. . On the other hand, the drone recognition phase is a welldefined problem. Therefore, it is straightforward to code the drone recognition phase state machine after a small workflow analysis. The drone path was preplanned; thus, the HDDL planner was not used. Instead, a simple gridbased path planner was employed.

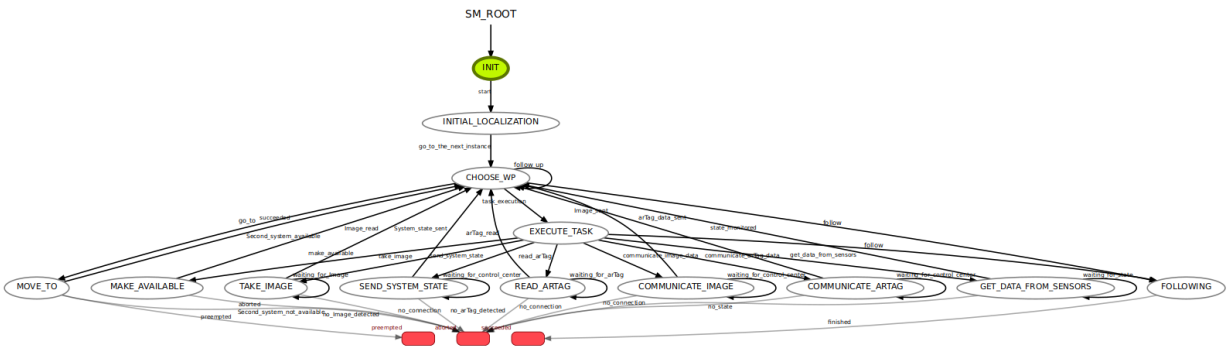


*Figure 7: Resulting state machine for the rover task planning*

Each entry of the state machine is defined as a Python class and interfaced with ROS. Each entry can have a successful outcome or a failed one. The failed outcome may depend on different preconditions:

- MAKE AVAILABLE: it is the entry that automatically turns on the drone and establishes a communication link between the rover and the drone. The failed outcome is triggered if the

drone cannot be "activated".

- MOVE TO: it is the entry interfaced with the path planning code. The failed outcome can be generated if:
  - No path to the objective is found.
  - More than two-wheeled' motors stop working.
  - The consumed energy is anomalous. The system is consuming too much battery. That usually indicates a non-working wheel motor.
- TAKE IMAGE: the entry that captures a "screenshot" of the depth camera's video stream. The fail outcome is triggered if:
  - The system cannot access the depth camera data.
  - If the image is evaluated as "not compliant" by the control center.

  The anomaly will be registered with some delay in the latter case: the image should arrive at the control center, be evaluated, and a "non-compliant message" is sent to the system.
- GET DATA FROM SENSORS: it is the entry that generates a report on the health of the system with battery level, encoders reading from the last point to point trajectory, localization data, recorded data frequency from the cameras, and LiDAR. The failure occurs when the system cannot retrieve a stream of data from a sensor.
- SEND SYSTEM STATE: it is the entry that forwards to the control center a report generated by "GET DATA FROM SENSORS". Its failed outcome is triggered if the system cannot communicate with the control center.
- READ ARTAG: the entry responsible for reading and interpreting arTag information. Its failed outcomes are the same as "TAKE IMAGE".
- COMMUNICATE IMAGE: it is the entry that takes care of communicating the image taken by "TAKE IMAGE" to the control center. Its failed outcome is triggered if the system cannot communicate with the control center.
- COMMUNICATE ARTAG: it is a similar entry to "COMMUNICATE IMAGE", but it takes care of communicating the arTAG to the control center. The failed outcomes are the same as "COMMUNICATE IMAGE".
- FOLLOWING: it is an entry that brings the system to a safe configuration (far from obstacles) and stops it.

If the rover fails an outcome of the state machine, it stops. It will re-evaluate its resources, update the predicates in the HDDL problem file's initial conditions, and try to find a new plan to visit as many points of interest as possible. If no plan can be found, the autonomous mission is stopped. The system will contact the control center and wait for their intervention. On the other hand, for the drone's autonomous mission, if any anomaly occurs, the system is stopped. It will land, launch its diagnostics, and, if it cannot find a solution, contact the control center.

## 4. Localization and Real-Time Mapping and Path Planning Module for Autonomy

This section provides a brief overview of the two main modules interfaced with the task planner to create fully autonomous systems: the Localization and Real-Time Mapping module and the path planner module.

- The Localization and Real-Time Mapping module takes care of the mapping during the recognition phase and the localization during the mission phase.

- The Path Planning module is focused on planning and executing a safe path for both systems.

### *4.1 Localization and Real-Time Mapping*

The software has four main modules:

- Internal SLAM Module: It provides the position of the drone thanks to the V_SLAM algorithm embedded in the T265.
- Depth Filtering Module: It filters depth readings to reduce the computational cost of DEM map generation.
- Data Merging Module: It uses position and depth data to generate a DEM of the drone's local environment.
- DEM Map Matching Module: It integrates the generated local DEM map with the global DEM map.

The Localization and Real-Time Mapping algorithms use C++ as the main coding language. The software will then be interfaced with the hardware through ROS topics, services, and action libraries.

During testing, the drone detected 80% of obstacles with a height of 6 cm or greater. The limitation is that small obstacles (about 6 cm high) are mixed with noise and non-obstacle features (such as grass). That is why the *Obstacle Avoidance Module* of the rover comes to remedy an eventual deficiency in detecting the 20% estimated remaining undetected obstacles. Moreover, if the drone is unable to generate a map or achieve the expected level of accuracy, the Collaborative Localization and Real-Time Mapping becomes useful. Indeed, in that case, the rover's Localization and Real-Time Mapping algorithms enable the rover to operate on the field even without a global map. As a collaboration, the goal of the drone's Localization and Real-Time Mapping algorithm is to detect at least the big features of the environment that influence global trajectories for the rover (e.g., big rocks, rift, steep slope). A map is generated based on the rover's mobility capabilities and sent to the rover. The rover's Path Planning algorithms compute an optimal path relative to the large features. As the rover starts moving, the goal of its Localization and Real-Time Mapping algorithm is to confirm and/or detect the presence of small features in the rover's local environment. If unregistered small obstacles are detected, the path-planning software would locally adjust the rover's trajectory. The points of interest detected by both the drone and the rover are used to compute the transformation between the rover's and the drone's coordinate frames to localize the systems in the environment.

### *4.2 Path Planning*

The Path Planning software plans and executes the trajectory for both the drone and the rover. Its main features and interfaces can be summarized as:

- It interfaces with the Task Planning software to determine the robot's goal/destination.
- It elaborates the Localization and Real-Time Mapping software output data, grid map, and position, to compute the optimal trajectory for the system under study.
- It controls the actuators to move the system to its destination effectively.
- It also listens to the encoder odometry readings, the LiDAR readings from obstacles, and the visual odometry from depth sensors to localize the system in a global or partial map.
- It uses Python as its primary programming language. However, the localization modules will be computed in C++ for better computational velocity.
- The software interfaces with the hardware via ROS topics, services, and action libraries.
- Both systems can be stopped in case of contingency by the control center, and they can be

directly piloted.

The first challenge of the drone is to provide a map of an unknown environment. Thus, the goal of the Path Planning software for the drone mapping phase is to compute and control the optimal trajectory for the drone to map the maximum amount of terrain within a given maximum flight time (battery-dependent). A grid-like path was used to create the map. During the mission phase, the drone was provided with "go-to" commands. It was able to localize itself and move directly toward the target. There were no obstacles in line of sight, therefore it was easy for the drone to accomplish its mission. The rover was provided with the map generated by the drone and post-processed by the Localization and Real-Time Mapping algorithm. The rover used this map as a global map to navigate its environment. The software has seven main modules:

- Translation Module: The module will transform the 3D map generated by the drone to a 2D cost-map.
- Localization Module: The module will estimate the position of the robots in a known map
- Global Path Planner Module: The module computes the optimal path for the rover in the known 2D cost-map. It uses a D* algorithm.
- Local Path Planner Module: This module uses the local map provided by the Rover's Localization and Real-Time Mapping Algorithm to adapt locally the trajectory of the rover if needed. It considers the new obstacles (if there are) registered on the local map to adapt the trajectory. The local path planner will smooth the overall trajectory followed by the systems.
- Controller Module: The module interfaces the high-level goals of the path planners with the system actuators.
- Trajectory Discretization Module: The module discretizes a given trajectory into a sequence of waypoints that have to be followed by the drone.
- Position Feedback Module: With the Localization Module, the Position Feedback Module can estimate the error between the theoretical trajectory and the real one. It rectifies the rover's movement to match the planned trajectory as closely as possible.

For the drone recognition phase, the objective is to compute and control the optimal trajectory for the drone to map the maximum amount of terrain within a given maximum flight time (battery-dependent). A gridlike path is used to create the map. During the mission phase, the drone is provided with"go to" commands. It can localize itself and move directly toward the target. There were no obstacles in the line of sight; therefore, there is no need for global or local path planners.

The rover path planning is divided into two main algorithms:

- Local Path Planner
- Global Path Planner

The global path planner uses the algorithms D* [16] and A* [16] to find the shortest path from two points in a given map. Both are graph-based algorithms, but A* benefits from using a heuristic to prune some paths and reduce computational cost. Therefore, no matter how good this heuristic is, there will be a non-zero chance of pruning the actual optimal path and many near-optimal paths. The D* search space is wider, and therefore, the pruning of the optimal path is less likely to happen. In general, D* can be a better replanner than A* [17] since (i) local changes in the world do not impact much on the path, and (ii) it avoids the high computational costs of backtracking. [17].

The two algorithms' metrics for estimating the path are pretty similar, except for the A* heuristic, as they both search a given grid for the best path. Therefore, the given grid must impose constraints

to enable the algorithm to find an optimal path, given the system's size and limitations. This is performed not by feeding the algorithm the "brute" occupancy grid map from the Localization and Real-Time Mapping algorithms, but rather with a cost map. In the global cost map, the obstacles are inflated to 0.5%, and the system's footprint is taken into account to exclude paths that would not fit the robot's size. In this case, the considered metrics for the cost-map are:

- Size of the obstacles
- Size of the robot footprint

The occupancy grid map of the Localization and Real-Time Mapping code treats all slopes that cannot be traversed by the rover as obstacles.

The local path planner uses the Dynamic Window Approach (DWA) for obstacle avoidance [18]. The algorithm accounts for the overall path from A*/D*, the "inflated" obstacle radius, and the robot footprint to avoid unforeseen obstacles along the path. The idea of the local path planner is to avoid the unforeseen obstacles and then return to the optimal path. If the local and optimal paths have an overall Euclidean distance exceeding a threshold, the global path is recalculated. For us, the threshold corresponds to the robot footprint, in which case we re-evaluate the path. The basic steps of the DWA are:

- To define the robot control space as (dx,dy, dtheta).
- To analyze different local paths as different couplings of angular and linear velocities for a given short period of time in the future.
- To score its trajectory propagated in the future, weighing the parameters:
  - Distance from the obstacles (already inflated by the cost-map);
  - Distance from global optimal path,
  - Avoided unforeseen obstacles,
  - Distance from final target.
- To choose the most suitable one.

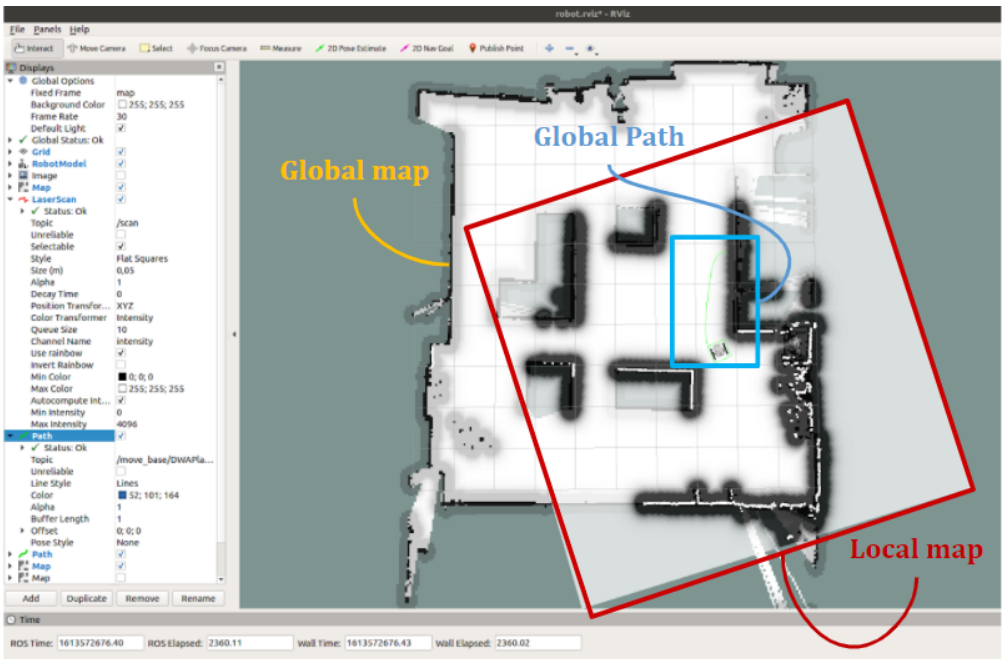


*Figure 8: Path Planning set-up for the rover*

## 5. Main Results and Conclusions

The task planner, along with the related modules, the path planner, and real-time mapping and localization algorithms, have been tested during the IGLUNA field campaign. The test video can be viewed at https://youtu.be/HWYhED8Z17Y. The first tests were conducted in the ISAE-SUPAERO laboratory and gym. During the first indoor testing in the gym at the beginning of the VFC, drone mapping tests were unsuccessful at a 3m flight height. Therefore, the drone flight height was lowered to 2m to achieve a high-quality 3D map. The drone's speed was also set to 0.5 m/s. During our final outdoor tests in Esperce, the 2m height and 0.5m/s conditions did not lead to a successful mapping phase. Thus, the flight height was lowered to 1.5m and the speed to 0.25m/s. The flight duration was about 4minutes. The 3D map is shown in Figure 10, the resolution was 3cm. After the drone landed, it autonomously began post-processing the 3D map and generated a 2D occupancy grid suitable for the rover's use, as shown in Figure 11( the resolution was 8cm). Overall, the results were satisfactory, and the provided map was sufficient to enable the rover to navigate the terrain.

*Figure 9: Set up of the Outdoor test in Esperce.*

The provided map was then analyzed by the task planner and discretized as a series of predicates that defined how the systems could move in the environment, which sensors were active on board, and the position of the point of interest in the map. The provided optimal plan is shown on Listing 2.

*Listing 2: Example of obtained final plan*

```
0 (make_available drone1)
1 (navigate rover0 waypoint0_7_1 waypoint111)
2 (visit waypoint111 rover0)
3 (navigate rover0 waypoint111 waypoint161)
4 (unvisit waypoint111 rover0)
5 (read_arTag rover0 waypoint161 objective1 camera0)
6 (communicate_arTag_data rover0 objective1)
7 (take_image rover0 waypoint161 objective1 camera0)
8 (communicate_image_data rover0 objective1)
9 (navigate drone1 waypoint0_10_1 waypoint161)
10 (visit waypoint161 drone1)
11 (navigate drone1 waypoint161 waypoint142)
12 (unvisit waypoint161 drone1)
```

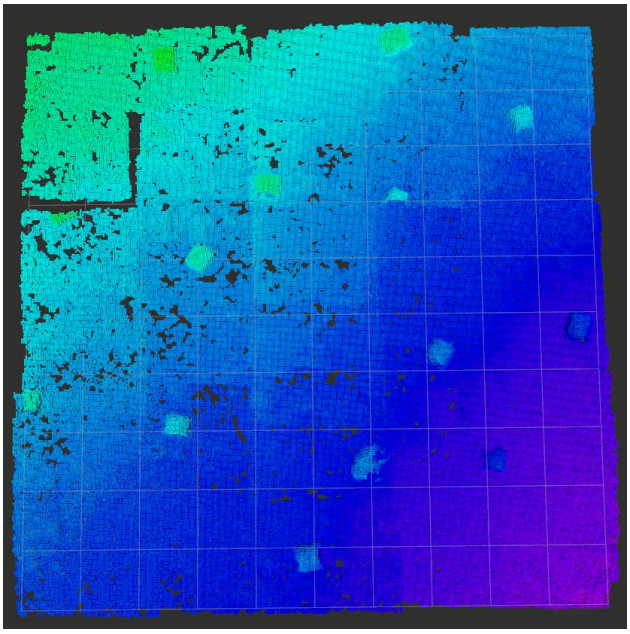

*Figure 10: 3D map of Outdoor testing in Esperce. Resolution 3cm*

```
(read_arTag drone1 waypoint142 objective4 camera2)
(communicate_arTag_data drone1 objective4)
(take_image drone1 waypoint142 objective4 camera2)
(communicate_image_data drone1 objective4)
(navigate rover0 waypoint0_10_1 waypoint161)
(visit waypoint161 rover0)
(navigate rover0 waypoint161 waypoint139)
(unvisit waypoint161 rover0)
(read_arTag rover0 waypoint139 objective6 camera0)
(communicate_arTag_data rover0 objective6)
(take_image rover0 waypoint139 objective6 camera0)
(communicate_image_data rover0 objective6)
(navigate rover0 waypoint0_10_1 waypoint161)
(visit waypoint161 rover0)
(navigate rover0 waypoint161 waypoint111)
(unvisit waypoint161 rover0)
(read_arTag rover0 waypoint111 objective7 camera0)
(communicate_arTag_data rover0 objective7)
(take_image rover0 waypoint111 objective7 camera0)
(communicate_image_data rover0 objective7)
(navigate drone1 waypoint0_10_1 waypoint161)
(visit waypoint161 drone1)
(navigate drone1 waypoint161 waypoint102)
(unvisit waypoint161 drone1)
(read_arTag drone1 waypoint102 objective2 camera2)
(communicate_arTag_data drone1 objective2)
(take_image drone1 waypoint102 objective2 camera2)
(communicate_image_data drone1 objective2)
(navigate rover0 waypoint0_10_1 waypoint161)
(visit waypoint161 rover0)
(navigate rover0 waypoint161 waypoint118)
```

*Figure 11: 2D Occupancy grid of Outdoor testing in Esperce. Resolution 8cm*

```
(unvisit waypoint161 rover0)
(read_arTag rover0 waypoint118 objective8 camera0)
(communicate_arTag_data rover0 objective8)
(take_image rover0 waypoint118 objective8 camera0)
(communicate_image_data rover0 objective8)
(navigate drone1 waypoint0_10_1 waypoint161)
(visit waypoint161 drone1)
(navigate drone1 waypoint161 waypoint98)
(unvisit waypoint161 drone1)
(read_arTag drone1 waypoint98 objective9 camera2)
(communicate_arTag_data drone1 objective9)
(take_image drone1 waypoint98 objective9 camera2)
(communicate_image_data drone1 objective9)
(navigate rover0 waypoint0_10_1 waypoint161)
(visit waypoint161 rover0)
(navigate rover0 waypoint161 waypoint96)
(unvisit waypoint161 rover0)
(read_arTag rover0 waypoint96 objective5 camera0)
(communicate_arTag_data rover0 objective5)
(take_image rover0 waypoint96 objective5 camera0)
(communicate_image_data rover0 objective5)
(read_arTag rover0 waypoint93 objective10 camera0)
(communicate_arTag_data rover0 objective10)
(take_image rover0 waypoint93 objective10 camera0)
(communicate_image_data rover0 objective10)
(navigate drone1 waypoint0_10_1 waypoint161)
(visit waypoint161 drone1)
(navigate drone1 waypoint161 waypoint60)
(unvisit waypoint161 drone1)
(read_arTag drone1 waypoint60 objective12 camera2)
(communicate_arTag_data drone1 objective12)
(take_image drone1 waypoint60 objective12 camera2)
```

```
(communicate_image_data drone1 objective12)
(navigate rover0 waypoint111 waypoint0_7_1)
(visit waypoint0_7_1 rover0)
(navigate rover0 waypoint0_7_1 waypoint75)
(unvisit waypoint0_7_1 rover0)
(read_arTag rover0 waypoint75 objective11 camera0)
(communicate_arTag_data rover0 objective11)
(take_image rover0 waypoint75 objective11 camera0)
(communicate_image_data rover0 objective11)
<==
```

After the task planner provided the positions of the obstacles, the path planner computed a trajectory that touched the different waypoints. During the IGLUNA field campaign, the task planner interfaces were tested. The mission went quite smoothly, and no replanning was needed. The overall plan was computed in around three minutes:

- The grounding process typical of hierarchical networks took 3.4 minutes.
- The solving duration was 0.005 seconds.

However, to obtain those results on a test terrain of 10*10 meters, we had to simplify the problem file and accurately post-process the map to determine the "safe" spaces where the rover could go and the distances of the different points of interest with respect to these safe spaces. The future work will focus on improving the path planner of both the rover and the drone:

- The rover path-planning parameters can be tuned to enhance performance. Moreover, the objective is to code a path planner that does not rely on the LIDAR for obstacle detection but instead on its cameras.
- The drone path planner should start taking into account possible obstacles on its trajectory. Therefore, it is necessary to implement an obstacle avoidance module.

The work on the task planner now has the objective of complete harmonization with MBSE tools. From the operational and behavioral design of the system engineer, it is possible to directly test and see the effectual output on the robotic platform. Therefore, the objective is to establish a bidirectional, directed feedback loop between the MBSE and HDDL. Moreover, it is envisioned to soon test the entire connection chain of the HDDL planner and its modules in a contingency situation to assess the method's adaptability.

## References


[1] C. Frost, "Challenges and opportunities for autonomous systems in space," 2011.

[2] Horizon-2020, *European robotic goal-oriented autonomous controller (ergo)*.

[3] ESA Advanced Concepts Team, *Ai in space workshop*, 2013.

[4] S. Chien, R. Knight, R. Stechert A.and Sherwood, and G. Rabideau, "Integrated planning and execution for autonomous spacecraft," *IEEE Aerospace Conference*, 1999.

[5] T. Kaku and et al, "Detection of intact lava tubes at marius hills on the moon by selene (kaguya) lunar radar sounder," *Geophysical Research Letter*, 2017.

[6] J. Thangavelautham, M. S. Robinson, A. Taits, T. McKinney, et al., "Flying, hopping pit-bots for cave and lava tube exploration on the moon and mars," *arXiv preprint arXiv:1701.07799*, 2017.

[7] D. Höller, G. Behnke, P. Bercher, S. Biundo, et al., "HDDL: An Extension to PDDL for Expressing Hierarchical Planning Problems," in *AAAI Conference on Artificial Intelligence (AAAI)*, NYC, NY, USA, 2020. DOI: 10.1609/aaai.v34i06.6542

[8] S. Russell and P. Norvig, "Artificial intelligence: A modern approach," 2002.

[9] M. Ghallab, D. Nau, and P. Traverso, *Automated Planning: theory and practice*. Elsevier, 2004.

[10] I. Georgievski and M. Aiello, "An overview of hierarchical task network planning," *arXiv preprint arXiv:1403.7426*, 2014.

[11] J. Rimani, C. Lesire, S. Lizy-Destrez, and N. Viola, "Application of mbse to model hierarchical ai planning problems in hddl," 2021.

[12] P. Z. Schulte, "A state machine architecture for aerospace vehicle fault protection," *PhD Dissertation, Georgia Institute of Technology*, 2018.

[13] C. Lesire and A. Albore, "Pyhipop – hierarchical partial-order planner.," *International Planning Competition*, 2020.

[14] P. Bechon, C. Lesire, and M. Barbier, "Hybrid planning and distributed iterative repair for multi-robot missions with communication losses," *Auton. Robots*, vol. 44, no. 3-4, pp. 505–531, 2020. DOI: 10.1007/s10514-019-09869-w

[15] D. Schreiber, *Getting started with automated planning*, 2019.

[16] *Finding the shortest path, with a little help from dijkstra*, https://medium.com/basecs/finding-the-shortest-path-with-a-little-help-from-dijkstra-613149fbdc8e, Accessed: 2021-01-02.

[17] *Robotic motion planning: A* and d* search*, http://www.cs.cmu.edu/~motionplanning, Accessed: 2021-01-02.

[18] *Dynamic window algorithm motion planning*, , Accessed: 2021-01-02.