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 becomes increasingly important. One of the most innovative missions is the exploration of lunar lava tubes. Those volcanic architectures should serve as natural shelters for future crewed missions in the Moon's equatorial region. 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 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 (previously Swiss Space Center). The rover mission should explore an unknown environment and reach multiple 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 lava tubes, where a series of rovers and thruster-flying/hopping bots will join forces to provide information about their environment to a control center. Therefore, during the analog mission, the rover will be assisted by a small drone equipped with a tracking camera and LiDAR, capable of short flights of a few minutes to simulate the lunar bot. The systems will be in contact with a control center 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 indicate their positions with respect to the rover's 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 mission execution, avoid obstacles, and count the number of path or plan re-evaluations. This paper presents the hierarchical task planner's performance and its integration with the mapping, path planning, and path execution modules.

Keywords: Autonomous Operations, HDDL, Path Planning, IGLUNA, State Machine

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 inferior. Nevertheless, missions are becoming more complex, and the capability to plan, control, and predict all potential mission events is decreasing. Therefore, autonomous decision-making capabilities on board future exploration systems are considered critical technologies for this new era of space exploration. That aim can be achieved through technological advances in artificial intelligence over the last decade [1]. In this context, space agencies, companies, and universities are engaged in defining 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 primary 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 Moon's geological and mineralogical history. On the other hand, they may serve as ideal shelters for future human missions in the Moon's equatorial region. Indeed, scientists have hypothesized that those volcanic tunnels may offer an additional layer of protection for astronauts against radiation and micro-meteoroids.

However, a robotic exploration mission is envisioned before any human mission can be allowed in the lava tubes. The exploration systems should be able to autonomously map, navigate, and operate within the tubes and 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 connected to the sensors and the system's actuators to implement the plan.

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

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. The testing terrain was 10 meters by 10 meters.

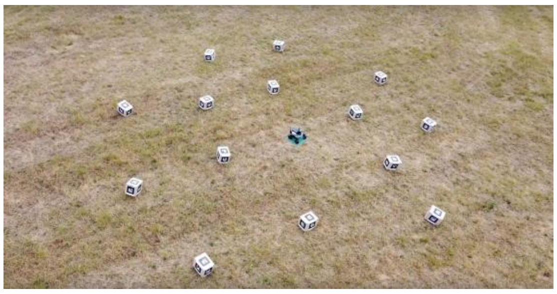


Figure 1: Set up of the Outdoor test in Toulouse.

It was populated with points of interest simulated as boxes with arTags on the sides. Moreover, the entire operational chain was tested remotely, relying on a network that simulated Moon-Earth communication provided by Space Innovation. The mission was divided into two main phases:

- Recognition phase: during the recognition phase, the drone flew following a preplanned path while mapping the environment and recognizing the points of interest
- Mission phase: during the mission phase, the drone and the rover started exploring the unknown environment while going to different points of interest, taking pictures, and reading the arTags on them.

During both phases, the systems were autonomous: the objective was to test the algorithms and the strategies that one day would permit the level of autonomy required to explore the lunar 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. During the

[1] https://space-innovation.ch/

testing, an ”Earth drone” was used to simulate the operations of a lunar bot. During the envisioned lava tube exploration lunar mission, a small, propelled bot will navigate the tunnel environment, avoiding obstacles. 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. Even if the lunar bot should have a flight autonomy of around 45 minutes [6]. The concepts of operations of the analog mission are illustrated in Figure 2.

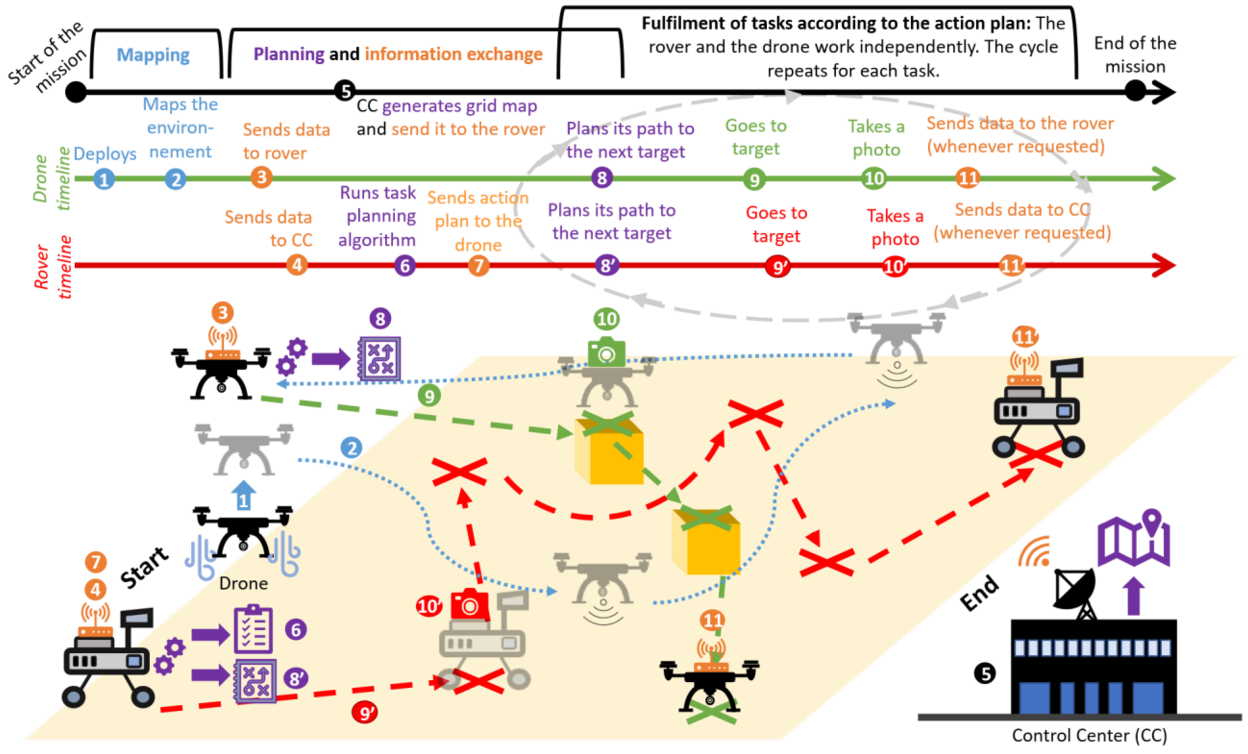

Start of the mission
Mapping
Planning and information exchange
Fulfilment of tasks according to the action plan: The rover and the drone work independently. The cycle repeats for each task.
End of the mission
Drone timeline
Rover timeline
Deploys
Maps the environ-nement
Sends data to rover
CC generates grid map and send it to the rover
Plans its path to the next target
Goes to target
Takes a photo
Sends data to the rover (whenever requested)
Sends data to CC
Runs task planning algorithm
Sends action plan to the drone
Plans its path to the next target
Goes to target
Takes a photo
Sends data to CC (whenever requested)
Start
Drone
End
Control Center (CC)

Figure 2: Concept of Operations of the CORODRO team during the IGLUNA mission.

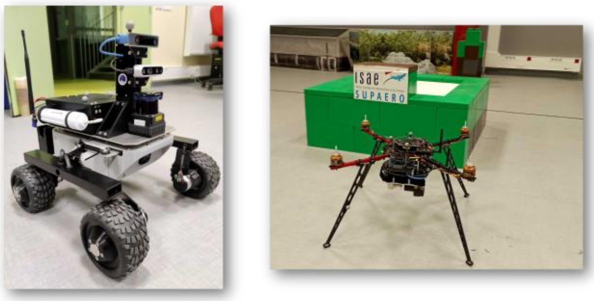
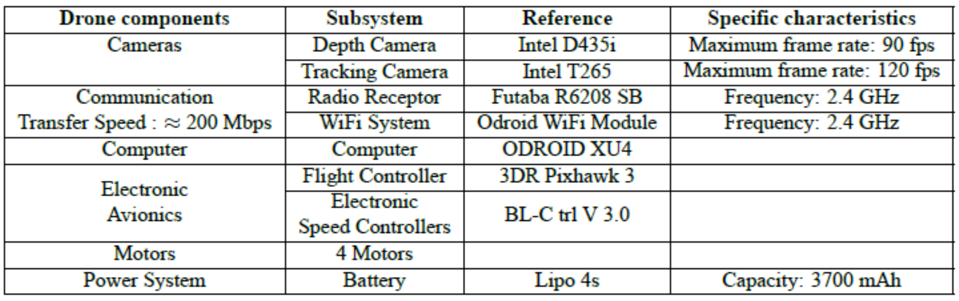


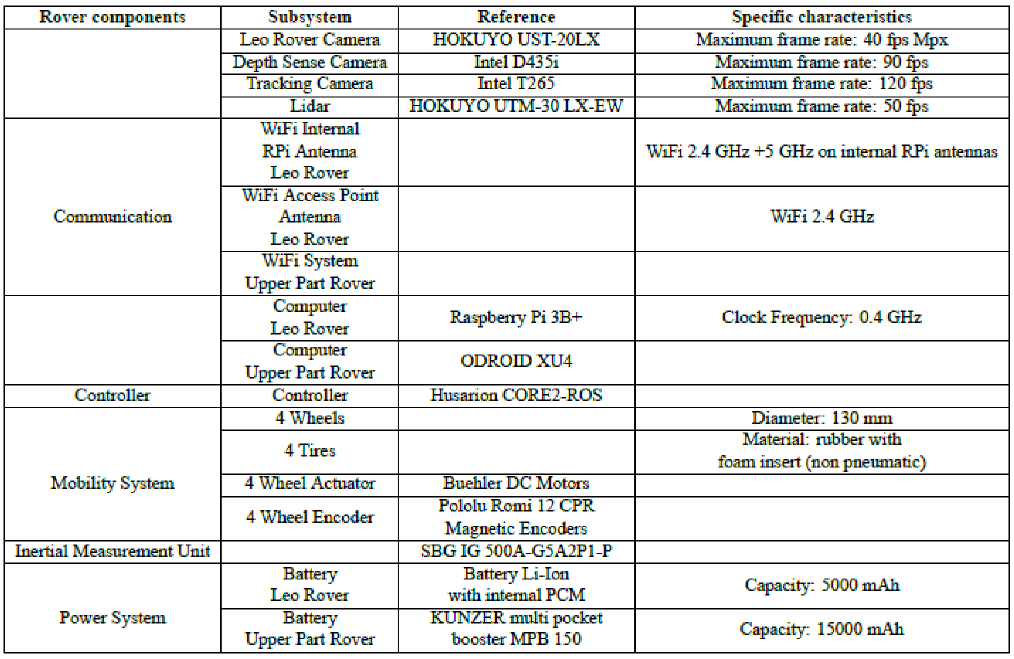
Figure 3: Rover and drone that will be used during the IGLUNA Mission.

The systems used during testing are shown in Figure 3, and their components are described in Tables 1 and 2.



Drone components	Subsystem	Reference	Specific characteristics
Cameras	Depth Camera	Intel D435i	Maximum frame rate: 90 fps
	Tracking Camera	Intel T265	Maximum frame rate: 120 fps
Communication Transfer Speed : ≈ 200 Mbps	Radio Receptor	Futaba R6208 SB	Frequency: 2.4 GHz
	WiFi System	Odroid WiFi Module	Frequency: 2.4 GHz
Computer	Computer	ODROID XU4	
Electronic Avionics	Flight Controller	3DR Pixhawk 3	
	Electronic Speed Controllers	BL-C trl V 3.0	
Motors	4 Motors		
Power System	Battery	Lipo 4s	Capacity: 3700 mAh

Table 1: List of equipment of the Drone.



Rover components	Subsystem	Reference	Specific characteristics
	Leo Rover Camera	HOKUYO UST-20LX	Maximum frame rate: 40 fps Mpx
	Depth Sense Camera	Intel D435i	Maximum frame rate: 90 fps
	Tracking Camera	Intel T265	Maximum frame rate: 120 fps
	Lidar	HOKUYO UTM-30 LX-EW	Maximum frame rate: 50 fps
Communication	WiFi Internal RPi Antenna Leo Rover		WiFi 2.4 GHz +5 GHz on internal RPi antennas
	WiFi Access Point Antenna Leo Rover		WiFi 2.4 GHz
	WiFi System Upper Part Rover		
	Computer Leo Rover	Raspberry Pi 3B+	Clock Frequency: 0.4 GHz
	Computer Upper Part Rover	ODROID XU4	
Controller	Controller	Husarion CORE2-ROS	
Mobility System	4 Wheels		Diameter: 130 mm
	4 Tires		Material: rubber with foam insert (non pneumatic)
	4 Wheel Actuator	Buehler DC Motors	
	4 Wheel Encoder	Pololu Romi 12 CPR Magnetic Encoders	
Inertial Measurement Unit		SBG IG 500A-G5A2P1-P	
Power System	Battery Leo Rover	Battery Li-Ion with internal PCM	Capacity: 5000 mAh
	Battery Upper Part Rover	KUNZER multi pocket booster MPB 150	Capacity: 15000 mAh

Table 2: List of components of the Rover.

3. Task Planning Algorithm

Task planning is the discipline that considers how a robotic system's tasks are distributed in time. It usually outputs a sequence of actions that can be interfaced directly with 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). In addition, however, it introduces the concept of hierarchy 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].

Ontologies don't need to be trained on large amounts of data to describe systems' actions and/or behavior. However, they need a logical or functional architecture to define the robot's behavior.

More specifically, hierarchical planners can analyze a set of actions at different levels of abstraction [9]. In more detail, the goal objectives are derived by refining the initial task network [10], yielding an executable sequence of actions. Many real-world tasks already have built-in hierarchical structures [10]. Therefore, the use of HDDL naturally follows from the system's functional analysis and its expected behavior during the mission [11]. Going back to our case study, the task planning software defines the overall behavior of the two collaborative systems: (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 is provided in Table 3.

Figure4showstheblockdiagramoftheenvisionedtask planner. The software is interfaced with path planning, mission analysis, and localization and mapping modules as follows:

- The concept of operations and the functional analysis derived from mission analysis define the content of the domain file; therefore, the system's expected behavior under study.
- The occupancy grid from the localization and real-time mapping module is incorporated in the ”problem.hddl” file. It indicates the map's traversability and the location of the point of interest.
- The Action Library of ROS is configured for both the rover and the drone to execute the commands given by the execution layer.
- The SMACH Monitor State of ROS [12] as the execution layer. The code developed with the SMACH classes monitors the sensors and stops, changes, or reconfigures the system's plan.

The HDDL *plan solver* is written in Python and provided by our sponsor, ONERA. It is called HiPOP [13] [14]. This code outputs a plan that is saved as a ”.txt” file. This plan is then parsed by Python code running as a ROS node. Then, this ROS node is interfaced with ROS Smach [12], the ROS library, to implement and visualize state machines. The last step is the robotic platform's effective action: the robotic system's actions are followed by logged information displayed in the terminal. The operators will effectively see the control center's logging information during the field campaign. However, the visualization through a state machine is preferred because it is easier to understand and follow.

The important concepts to read and understand HDDL files are:

- *:task()* - those entries define what the designer expected the system under study to be able to perform. They are usually high-level functions that can be further decomposed.
- *:method()* - those entries define how a high level *:task()* can be accomplish through ordered sub-tasks (hierarchical system). These subtasks can then be *compound tasks* or *primitive tasks*. The firsts are subtasks that other methods can further decompose. The latter are subtasks that directly affect 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 true or false statements that describe 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 ideal for modeling using hierarchical and HDDL approaches. The main objective is to optimize the tasks that both systems need to perform simultaneously. The optimization considers the distance between the point of interest, the positions of the obstacles, and the initial positions of the systems. The solver seeks the shortest path to visit all waypoints while satisfying constraints (e.g., battery consumption). Therefore, this study is the natural evolution of [15].

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, the rover and the drone, start at *waypoint0*. The rover must call the drone; however, it can move freely in 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 and optimize 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, as well as to define when the system can photograph an objective, is shown in Figure 6. The predicates describing movement and ”visibility” of a point of interest are derived from the recognition phase grid map and arTags list.

The final benchmark plan is shown in Listing 1. Even in a simple scenario like the benchmark, 38 steps are needed to achieve the envisioned goal.

```
0 (make_available drone1)
1 (visit waypoint0 drone1)
2 (navigate drone1 waypoint0 waypoint1)
3 (unvisit waypoint0 drone1)
4 (read_arTag rover0 waypoint0 objective2 camera0)
5 (communicate_arTag_data rover0 general objective2)
6 (take_image drone1 waypoint1 objective2 camera3 fisheye)
7 (communicate_image_data drone1 general objective2 fisheye)
8 (navigate drone1 waypoint1 waypoint4)
9 (visit waypoint4 drone1)
10 (navigate drone1 waypoint4 waypoint6)
11 (unvisit waypoint4 drone1)
12 (read_arTag drone1 waypoint6 objective0 camera2)
13 (communicate_arTag_data drone1 general objective0)
14 (take_image drone1 waypoint6 objective0 camera3 fisheye)
15 (communicate_image_data drone1 general objective0 fisheye)
16 (navigate drone1 waypoint6 waypoint4)
17 (visit waypoint4 drone1)

18 (navigate drone1 waypoint4 waypoint8)
19 (unvisit waypoint4 drone1)
20 (read_arTag drone1 waypoint8 objective1 camera2)
21 (communicate_arTag_data drone1 general objective1)
22 (take_image drone1 waypoint8 objective1 camera2 depth)
23 (communicate_image_data drone1 general objective1 depth)
24 (visit waypoint0 rover0)
25 (navigate rover0 waypoint0 waypoint5)
26 (unvisit waypoint0 rover0)
27 (read_arTag rover0 waypoint5 objective0 camera0)
28 (communicate_arTag_data rover0 general objective0)
29 (take_image rover0 waypoint5 objective0 camera0 depth)
30 (communicate_image_data rover0 general objective0 depth)
31 (navigate drone1 waypoint8 waypoint4)
32 (visit waypoint4 drone1)
33 (navigate drone1 waypoint4 waypoint5)
34 (unvisit waypoint4 drone1)
35 (get_data_from_sensors rover0)
36 (send_system_state rover0 general)
37 (get_data_from_sensors drone1)
38 (send_system_state drone1 general)
```

*Listing 1: Complete for the 3*3 meters benchmark scenario.*

lightgray 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: - Sensor readings supplied the Action Feedback Module - Initial generated grid map and position of the systems in the map OUTPUTS: - HDDL Plan Generation Module	None
Domain File Module	This module outlines all the possible actions that the system can execute It considers even a cluster of possible actions that particular situations can trigger	HDDL	OUTPUTS: - HDDL Plan Generation Module	INPUTS: - Expected behavior of the system - 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: - Domain file - Problem file OUTPUTS: - ".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: - ".txt" file from the HDDL Plan Generation Module	INPUTS: - None for the rover - 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: - State machine generated by the Plan Dispatch Module	None
Actions Feedback Module	The module will store and manage the inputs from the sensors It will supply the readings to the Problem File if new obstacles are detected or if a way-point cannot be reached	Python	INPUTS: - Sensor readings from ROS topics and services	INPUTS: - None for the rover - WiFi-like network for the drone to send its sensor readings to the rover

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

3.1 State Machine

Starting from the plan generated by the HDD solver, it is possible to code the system's state machine rapidly. 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 well-defined 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 grid-based path planner was employed.

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-wheel 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: it is the entry that takes a "screenshot" of the video stream of the depth camera. 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 and be evaluated. 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 failed outcome is triggered when the system fails to get a

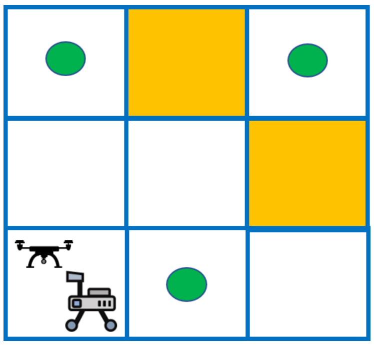


Figure 5: Simple HDDL problem benchmark.

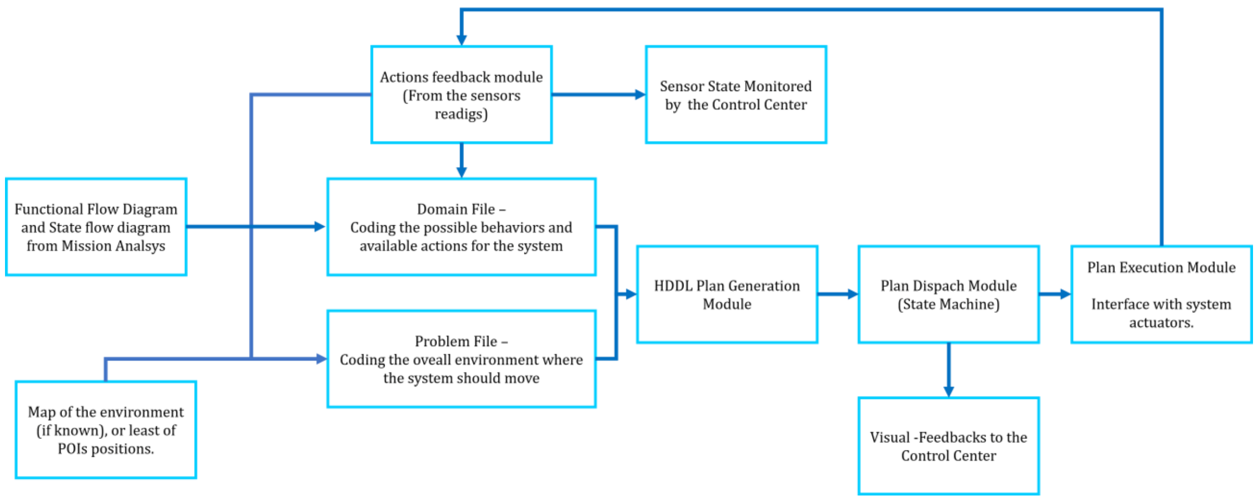
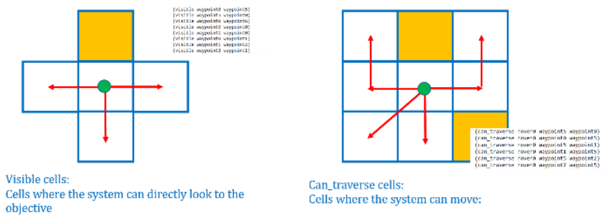

Actions feedback module (From the sensors readigs)
Sensor State Monitored by the Control Center
Functional Flow Diagram and State flow diagram from Mission Analsys
Domain File – Coding the possible behaviors and available actions for the system
Problem File – Coding the oveall environment where the system should move
Map of the environment (if known), or least of POIs positions.
HDDL Plan Generation Module
Plan Dispach Module (State Machine)
Plan Execution Module
Interface with system actuators.
Visual -Feedbacks to the Control Center

Figure 4: Task planner block diagram


Visible cells:
Cells where the system can directly look to the objective
Can_traverse cells:
Cells where the system can move:

```
(visible waypoint0 waypoint5)
(visible waypoint4 waypoint8)
(visible waypoint8 waypoint6)
(visible waypoint5 waypoint0)
(visible waypoint1 waypoint0)
(visible waypoint8 waypoint1)
(visible waypoint1 waypoint2)
(visible waypoint2 waypoint1)
```

```
(can_traverse rover0 waypoint5 waypoint0)
(can_traverse rover0 waypoint0 waypoint5)
(can_traverse rover0 waypoint5 waypoint1)
(can_traverse rover0 waypoint1 waypoint5)
(can_traverse rover0 waypoint5 waypoint2)
(can_traverse rover0 waypoint2 waypoint5)
```

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

Stream of data from one sensor.

- SEND SYSTEM STATE: It is the entry that forwards a report generated by ”GET DATA FROM SENSORS” to the control center. The failure outcome is triggered if the system cannot communicate with the control center.
- READ ARTAG: it is the entry in charge of reading and interpreting the arTag information. Its failed outcomes are the same as those of “TAKE IMAGE”.
- COMMUNICATE IMAGE: it is the entry that takes care of communicating the image taken by ”TAKE IMAGE” to the control center. The failure 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 is available, the autonomous mission is stopped. The system will contact the control center and wait for their intervention. On the other hand, for the autonomous drone mission, the system is halted in the event of an anomaly. It will land, run diagnostics, and contact the control center if it cannot resolve the issue.

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

This section will briefly overview the two main modules interfaced with the task planner to create totally 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.

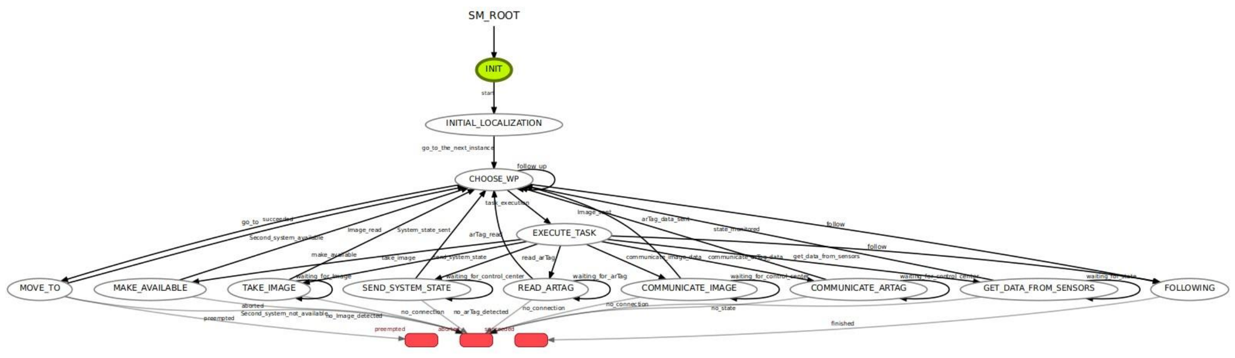


Figure 7: Resulting state machine for the rover task planning.

4.1 Localization and Real-Time Mapping The software has four main modules:

- The ”V_SLAM” algorithm embedded in the T265 indicates position to both the drone and the rover.
- The ”Depth Filtering Module” decreases the computational cost of the DEM map generation.
- The ”Data Merging Module” uses the position and the depth data to generate a DEM map of the drone’s local environment.
- The ”DEM Map Matching Module” integrates the generated local DEM map with the global DEM map.

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

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:

- From the task planning modules, the path planner acquired the initial and goal positions.
- It is interfaced with the localization and real-time mapping software to acquire the system position in space and to compute the optimal trajectory for the system under study.
- It is interfaced with the wheel actuators to give direct commands.
- It listens to the LiDAR and the depth sensors' readings of the obstacle to perform obstacle avoidance maneuvers.

Most of the path planner is coded in Python. The use of ROS allows easy interfacing between software written in different languages, e.g., Python and C++, through topics, services, and actions. Therefore, the Python modules of the path planner can be easily interfaced with the C++ localization module.

The software has seven main modules for the rover:

- Translation Module: The module will transform the 3D map generated by the drone into 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 costmap. 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 the trajectory of the rover locally if needed. It considers the new obstacles (if there are) registered on the local map to adjust 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 movement needed for the rover 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 grid-like path is used to create the map. During the mission phase, the drone is provided with“goto” 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.

More in detail, 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 than the A*, and therefore, the pruning of the optimal path is less likely to happen. In general, D* can be a better re-planner 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].

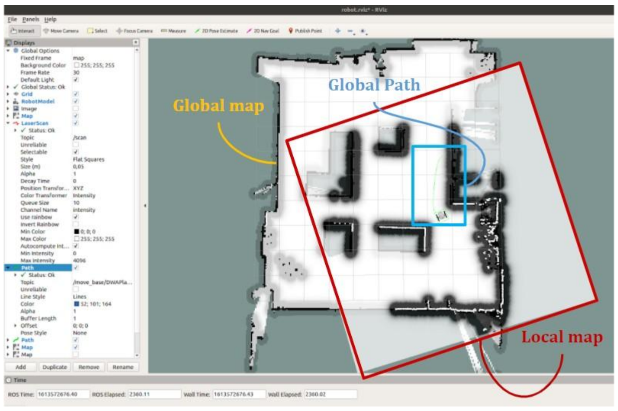
The metrics used by the two algorithms to estimate the path are pretty similar, except for the A* heuristic, as they both search for the best path in a given grid. Therefore, the given grid must impose constraints to enable the algorithm to find a good path, given the system's size and limitations. This is performed not by feeding the algorithm the “brute” occupancy grid map from the Localization and Real-Time Mapping algorithms, but by feeding it a cost map. In the global cost map, obstacles are inflated to 0.5% and the system’s footprint is accounted for to exclude paths that do 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 in the Localization and Real-Time Mapping code treats all slopes that the rover cannot traverse as obstacles.

The local path planner uses the Dynamic-Window Approach (DWA) for obstacle avoidance [18]. The algorithm takes into account the overall path from A*/D*, the ”inflated” radius of the obstacles, and the robot footprint to avoid unforeseen obstacles on its path. The idea of the local path planner is to avoid unexpected 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. The threshold for us is 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 the optimal path,
  - Avoided unforeseen obstacles, – Distance from the final target.
- To choose the most suitable one.



```
Global Options
  Fixed Frame            map
  Background Color       255; 255; 255
  Frame Rate             30
  Default Light          ✓
Global Status: Ok
Grid                     ✓
RobotModel               ✓
Image
Map                      ✓
LaserScan                ✓
  Status: Ok
  Topic                  /scan
  Unreliable
  Selectable             ✓
  Style                  Flat Squares
  Size (m)               0,05
  Alpha                  1
  Decay Time             0
  Position Transfor...   XYZ
  Color Transformer      Intensity
  Queue Size             10
  Channel Name           intensity
  Use rainbow            ✓
  Invert Rainbow
  Min Color              0; 0; 0
  Max Color              255; 255; 255
  Autocompute Int...     ✓
  Min Intensity          0
  Max Intensity          4096
Path                     ✓
  Status: Ok
  Topic                  /move_base/DWAPla...
  Unreliable
  Line Style             Lines
  Color                  52; 101; 164
  Alpha                  1
  Buffer Length          1
  Offset                 0; 0; 0
  Pose Style             None
Path                     ✓
Map                      ✓
Map

ROS Time: 1613572676.40   ROS Elapsed: 2360.11   Wall Time: 1613572676.43   Wall Elapsed: 2360.02
```

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

5. Main Results and Conclusions

The task planner and the related modules —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 laboratory and in the gym of ISAE-SUPAERO. During the first indoor test in the gym at the beginning of the testing campaign, 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, 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 4 minutes. The 3D map is shown in Figure 9; the resolution was 3cm. After the drone landed, it autonomously began post-processing the 3D map and generated a 2D occupancy grid suitable for rover use. Figure 10 shows that the resolution was 8cm. Overall, the results were satisfactory, and the provided map was sufficient to enable the rover to navigate the terrain. During testing, the drone detected 80% of obstacles with a height of 6 cm or more. The limitation is that small obstacles (about 6 cm height) are mixed with noise and non-obstacle features (such as grass). That is why the *Obstacle Avoidance Module* of the rover is designed to address an eventual deficiency in detecting the 20% estimated remaining undetected obstacles. Moreover, if the drone is unable to generate a map or reach the expected level of accuracy, the Collaborative Localization and Real-Time Mapping reveals its usefulness. 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 part of the collaboration, the drone's Localization and Real-Time Mapping algorithm aims to detect at least the most significant environmental features that influence the rover's global trajectories (e.g., large rocks, rifts, steep slopes). A map is generated based on the rover's mobility capabilities and sent to it. The rover's Path Planning algorithms compute an optimal path relative to significant features. As the rover starts moving, its Localization and Real-Time Mapping algorithm aims to confirm and/or detect the presence of small features in its local environment. If small unregistered obstacles are detected, the path-planning software would locally adjust the rover's trajectory. The point of interest detected by both systems is used to compute the transformation between the rover's and the drone's coordinate frames to localize the systems in the environment.

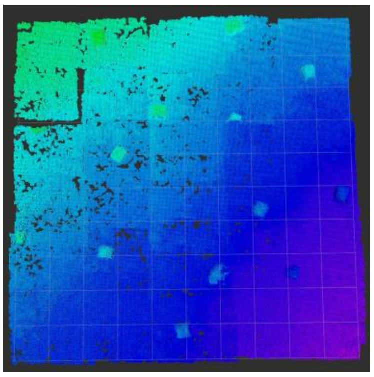


Figure 9: 3D map of Outdoor testing (Resolution 3cm)

The task planner then analyzed the provided map and discretized it into 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. The provided optimal plan is shown in Listing **2.**

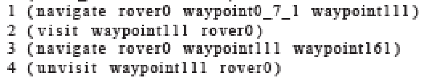
After the task planner provided the obstacle positions, the path planner computed a trajectory that passed through the waypoints. During the IGLUNA field campaign, the task planner interfaces were tested. The mission was quite smooth, and no re-planning was needed. The overall plan was computed in around three to four 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 testing terrain of 10*10 meters, we had to simplify the problem file and accurately post-process the map to get information on the "safe" spaces where the rover could go and the distance 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 planner parameters can be tuned to enhance its performance. Moreover, the objective is to code a path planner that relies not on the LIDAR for obstacle detection but 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 all the connection chains of the HDDL planner and the various modules under contingency conditions to assess the method's adaptability.



```
0 (make_available drone1)
1 (navigate rover0 waypoint0_7_1 waypoint111)
2 (visit waypoint111 rover0)
3 (navigate rover0 waypoint111 waypoint161)
4 (unvisit waypoint111 rover0)
```

```
(read_arTag rover0 waypoint161 objective1 camera0)
(communicate_arTag_data rover0 objective1)
(take_image rover0 waypoint161 objective1 camera0)
(communicate_image_data rover0 objective1)
(navigate drone1 waypoint0_10_1 waypoint161)
(visit waypoint161 drone1)
(navigate drone1 waypoint161 waypoint142)
(unvisit waypoint161 drone1)
(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)
(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)
```

Listing 2: Example of the obtained final plan.

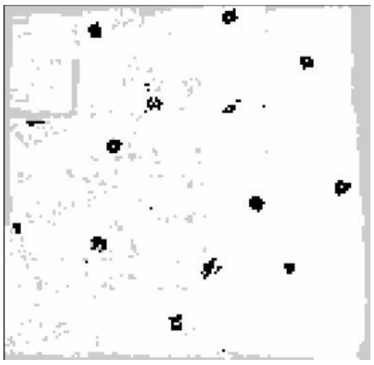


Figure 10: 2D Occupancy grid of Outdoor testing (Resolution 8cm)

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. Sherwood, and G. Rabideau, “Integrated planning and execution for autonomous spacecraft,” *IEEE Aerospace Conference*, 1999.

[5] T. Kaku et al, “Detection of intact lava tubes at Marius Hills on the moon by Selene (Kaguya) lunar radarsounder,” Geophysical Research Letters,2017.

[6] J. Thangavelautham, M. S. Robinson, A. Taits, T. McKinney, *et al.*, “Flying,hoppingpit-botsforcave 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 multirobot missions with communication losses,” *Auton. Robots*, vol. 44, no. 3-4, pp. 505–531, 2020. DOI: 10.1007/s10514-019-09869-w.

[15] J. Rimani, S. Lizy-Destrez, and N. Viola, “A novel approach to planetary rover guidance, navigation and control based on the estimation of the remaining useful life,” 2020.

[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.