Feasibility Study of LUFAR
Bachelor End Project Final Report

July 5, 2019

Team:	Dana van Hassel	Lead Testing
	Maxim Liefaard	Lead Programmer
	Raoul Bruens	Producer
	Sterre Noorthoek	Chief Editor
TU Coach:	dr. Thomas Abeel	Assistant Professor at Department of Bioinformatics, TU Delft
BEP Coordinator:	Ir. Otto Visser	Teacher at the Distributed Systems Group, TU Delft
BEP Coordinator:	Huijuan Wang	Assistant Professor at Department of Intelligent Systems, TU Delft
Client:	Maneesh Kumar Verma	Operations Manager at Lunar Zebro Lead Robotics at Stellar Space Industries
Client:	dr.ir. Chris Verhoeven	Associate Professor at Department of Microelectronics, TU Delft

STELLAR SPACE INDUSTRIES

TU Delft

Foreword

This feasibility study is part of the Bachelor End Project for the four author's Bachelor's study in Computer Science and Engineering. While this project was not initially part of the proposals, it has been accepted as a project. One of the authors came into contact with the client while waiting for a meeting. This sparked so much interest in the author that the proposal was requested from the client a few weeks later.

Soon after, the four authors started spending ten weeks working full-time on the project. The authors had already quite some experience working on different projects, however, this was the largest and longest project they participated in as of yet. Therefore, teamwork was essential to the success of the assignment. The one thing the team learnt most from this project would be rethinking how a project team should cooperate. The team focused on what each individual needed to be the most productive. Helping each other by being flexible and communicating clearly, made sure that the team was working as creative and productive as possible. The main take away is that the best results develop from enthusiasm, collaboration, and focus and that in a team everyone is responsible for creating such an atmosphere.

In addition, at the start of the project most team members did not have any experience with C++, ROS, C#, or Unity. Still, the team managed to figure out an adequate approach to tackle the project. In the end, each team member learnt how to work with these programming languages and frameworks. This is valuable experience for the future, especially in the field of Computer Science where it constantly required to work with new systems.

To this end, the team wants to express gratitude towards dr.ir. Chris Verhoeven, since he held the initial meeting that started this all and gave us a great launch of the project. The team would also like to thank the client, Maneesh Kumar Verma for his guidance during the project and providing knowledge on space missions and systems engineering. Last, the team would like to give a special thanks to Dr. Thomas Abeel for taking the time to supervise the project and leading the way when whenever members of the team lost communication with the lunar lander.

Contents

1 Summary

The aim of this project is to investigate the feasibility of Lunar Low Frequency Antennas for Radio Astronomy (LUFAR) [1]. LUFAR is a specific method of measuring frequencies that could aid the field of Radio Astronomy significantly when deployed on the moon. To deploy a LUFAR system on the moon a group of lunar rovers would need to be able to efficiently navigate to a lunar crater. The Lunar Zebro rover is being developed at the TU Delft with this mission in mind.

LunarSim is the software package developed during this project. It provides a simulation platform to design and validate lunar space missions for a variable amount of simulated Lunar Zebro rovers. This document showcases the research, software architecture, simulation environment software, rover behaviour software and research results.

The simulation environment is based in Unity, written in the C# programming language. To test rovers within this simulation environment, a rover behaviour algorithm was developed. The rover behaviour algorithm is written in C++, supported through a Robotic Operating System (ROS), and connected with Unity through an interface called ros_bridge. Through this bridge, data can be sent from the environment to the behaviour algorithm, this data contains sensor data from the simulated rovers. Similarly, data can be sent from the behaviour algorithm to the environment in the form of action commands. These action commands result in the simulated rover performing navigation tasks or other mission-related tasks like lunar crater observation.

The behaviour algorithm consists out of prioritized rules that make the rover avoid obstacles and other roves. The goal of the behaviour algorithm is to navigate a rover towards a simulated lunar crater, once arrived the rover can take pictures of the crater with its camera or navigate to a new location.

A user can interact with a user interface within the Unity simulation to design their own custom mission scenario. Entities like craters, rovers, the lander and mission goals can be configured through the easy to use interface.

To validate the functionalities of the simulation, several scenarios have been developed that cover and validate core rover behavioural functionalities. This is done by recording a set of statistics that function as evaluation criteria. From these criteria, the results and discussion were formulated to judge how representative the simulation is of a real scenario and to assess the feasibility of LUFAR.

2 Introduction

With the steady increase in space missions, enabled through technological advances and increase of commercialisation within the space flight industry, both more and increasingly complex missions can be designed for space. To this end, the Lunar Zebro project competes within this field through its small lunar rover design, drastically decreasing deployment costs and risk of the mission. The road map of Lunar Zebro aims to have a multitude of rovers deployed on the Moon, being able to complete several tasks like exploring, observing, and mapping.

Since this concept of rover cooperation adds a novel level of complexity to the mission, a feasibility study is required to look into the difficulties of navigating the Moon with a larger group of rovers. LunarSim is the software package developed during this project. LunarSim aims to facilitate a simulation environment in which Lunar Zebro rovers and space mission designs can be tested and validated. To legitimise the workings of the simulation, a few scenarios have been developed to test the core functionalities of the software product. These scenarios are based on phases in a practical mission plan that consists out of navigating to and observing a crater location. The scenarios is evaluated through examination of a set of defined fitness criteria.

In this report, the reader will find documentation on the development process of LunarSim: the simulation in Unity, the ROS back-end, and the bridge between these two systems. Additionally, the report elaborates how the developed software was used to aid in the feasibility study of LUFAR.

First, initial research and requirements are formulated to define the scope of the simulation, after which the software architecture is introduced. Then, the systems implemented for the simulation are explained. Subsequently, the implemented rover behaviour algorithm that was used for testing is explained, with additional resources on how to develop a new custom rover behaviour. After this, an evaluation is given of the simulation based on the initial requirements and research with future research and concluding remarks. At the end of the report, the technical specifications in terms of software architecture, simulation environment, and rover behaviour are defined to give an in-depth view of LunarSim.

3 Preliminary Research

This section focuses on all research done at the start of the project. The stakeholders and their needs for the mission are introduced. Based on this, the requirements and research question are formulated. Subsequently, the state of the art for both the missions simulations and behaviour algorithms is discussed. Finally, the working method gives an idea of how the goal of simulating the mission is approached. The reader is referred to Appendix E for the full literature report.

3.1 Stakeholder Requirements

In this section, the several stakeholders that are involved within Lunar Zebro and LunarSim are described. Mission needs are formulated, from which the requirements are defined. Note that these section will be discussed briefly. The details are mentioned in the initial literature study, see sections 2 and 3 of Appendix E.

3.1.1 Stakeholders

The main stakeholder for this project is identified to be Stellar Space Industries (SSI), since this project is an initiative from this company in collaboration with Delft University of Technology. In cooperation with students from the TU Delft, the Lunar Zebro group has been erected from this.

In the long term, the rovers are developed and eventually sent to the Moon for the purposes of LUFAR [1]. As the project continues, more stakeholders will become involved. Though these requirements are out of scope for the project and more relevant for the entire Lunar Zebro team, being aware of their involvement will help to explore the problem space of the rover simulation. They include the International Telecommunication Union (ITU) and space agencies like European Space Agency (ESA) and National Aeronautics and Space Administration (NASA). The complete overview can be found in subsection 2.2.3 Other Organisations of Appendix E.

3.1.2 Mission Needs

Stellar Space Industries defined the overall mission goal as: *To design, develop, and demonstrate a lunar rover that can walk on a lunar surface and communicate with Earth.* Eleven needs are defined for the Lunar Zebro project that simulate rovers in terms of goals, capabilities, and behaviour. The first one is that a group of rovers needs to navigate to the edge of a specific crater. The exact location of the crater is unknown, only the general direction towards the crater from the lander location is known. The other ten needs can be found in subsection 2.2.1 Mission Needs of Appendix E.

3.1.3 Requirements

The project requirements based on the formulated mission needs are defined. The requirements are prioritised based on the *MoSCoW* method [2] in terms of must, could, should, and won't haves. They have a tag associated with it, based on the needs of Lunar Zebro (N1-N11). The first must have requirement is that rovers must be capable of navigating from location A to location B. The second must have requirement is that during navigation the rovers must avoid collision with obstacles and each other. An overview can be found in section 3 Requirements of Appendix E.

3.2 Research Question

The research question that this report aims to answer is: "To what extend can a lunar rover mission scenario be reproduced reliably and accurately within a simulated environment?" This question is answered in section 8, to in the end validate the LUFAR scenario.

3.3 State of the Art

This section describes the state of the art of both missions simulations and behaviour algorithms.

3.3.1 Missions Simulators

In order to illustrate and explore the state of the art of mission simulations, a few examples of comparable software products have been gathered. Next to a short description, pros and cons will be formulated based on what can be taken away from the commodity and what will not be in scope of the final product.

Mission Simulation Support Software (MS3)
MS3 [3] is a simulator that provides complete and accurate interfaces for all components of a space mission. The simulator is divided into three parts: the Simulation Monitoring and Control, the Flight Deck, and the Mission Control Center. This also means that there are three devices needed at least for a complete simulation to be run.

Space Simulation (SS)
Note that this game is still unreleased in in Early Access, meaning that functionalities described here might not be representative of the final product. SS is a simulator developed by Stuka Games Inc [4], in which historical missions can be recreated, complete from launch to orbit, landing on a planet (or moon) and the journey back to earth. The simulator also supports a custom free-flight scenario and has realistic physics.

NASA's Curiosity Experience
Apart from developing a Mars Rover Game [5] to promote Curiosity's mission on Mars, NASA developed the Curiosity Experience [6] which showcases locations that Curiosity has traversed. Users can click on a location, to which the simulated Curiosity will travel to. Additionally, all the moving components can be explored and moved manually by the user.

3D Virtual Rover Operation Simulator (VROS)
VROS [7] is a computer-simulated virtual reality environment for planetary rover operation and testing. This simulation environment comes very close to the functionalities of LunarSim, though there are a few distinct services lacking that make the software product not fit for rover simulation as defined in subsection 3.2. That is, this software neither has support for a communication system for rovers, nor a user-friendly scenario editor for designing custom missions.

Pros	Cons
Mission Simulation Support Software	
Custom mission design	Multiple devices needed to run simulation
Lunar surface model	Realistic physics and representative control systems
	Complete mission simulation (from earth and back)
Space Simulation	
Correct scaling of models	Model of the entire solar system
Surface models based on NASA's data	Realistic physics
NASA's Curiosity Experience	
Surface models with height map	Interaction with mechanical components
User can define target location for rover to move to	Information on explored surface locations
Realistic rover model to scale	
3D Virtual Rover Operation Simulator	
Sensor interface as environment	Motor control visualization
User interface	Information on surface locations
Navigation system: localization, obstacle avoidance and path planning	
Terrain model with height map	

Table 1: Overview of pros and cons regarding state of the art instances.

3.3.2 Behaviour Algorithms

This section gives a short overview of two approaches for programming rover behaviour (swarming and multi-agent systems) and discusses whether they could be applied in this project's context.

Swarming is a phenomenon in nature where large groups of the same species, for example bee colonies or ant colonies, work together. Working as a team in large groups has shown to be an effective strategy to survive in nature and hence swarming has evolved over many years of evolution. Individual members of a swarm cannot achieve much, but when they work together they can form a collective intelligence. This report's definition of swarming includes a large group of individuals with a low complexity that perform simple tasks. However, the mission of this project is more complex and exists of multiple different tasks that cannot just be performed by a single swarming algorithm, as concluded in the literature study on this topic in section 6 Rover Behaviour of Appendix E. The Lunar Zebro rovers have a high computational power to their disposal and should have to use this in order to be prepared for different tasks of the mission.

A multi-agent system is a system that is self-organised and consists of multiple intelligent agents. Most multi-agent systems share similar characteristics: agents are autonomous, only have local views, and are decentralised as they cannot control the others. Swarming and multi-agent systems share similar advantages: both are scalable and robust to failing individuals. A difference between swarming and multi-agent systems is that multi-agent systems allow for a higher complexity of individuals. This allows agents in MAS to have different tasks. In the mission this can be used during the migration phase, where some rover will walk and some rovers will become part of the daisy chain.

3.4 Working Method

This section describes the working method by first elaborating on the mission's course of action and then describing the approach of simulating the mission, after which the evaluation criteria will be introduced.

3.4.1 Mission Description

In the mission description, the course of what our mission will look like is defined through a main mission goal, along with a scenario description of what this looks like in practice. Since the initial study (see Chapter 4 of Appendix E), the scope has been narrowed down to better fit the goal of the simulation. Concretely, Phase 1 and Phase 2 have been adjusted.

The mission goal is to map a location B on the far side of the Moon, which for example can be a Moon crater, such that further actions can be deduced from this map. From this, subsequent mission scenarios can ensue, like exploring the crater or cave at location B, or mapping of a new location C.

The scenario can be described as follows. A lander loaded with a set amount of rovers will descend upon an arbitrary location on the edge of the far side of the Moon. This is referred to as location A. From there, the goal is to have all rovers move out and explore the path to a certain location B on the far side. Subsequently, the rovers should capture the surrounding of location B and send it back to earth.

For the purpose of the specific mission that is simulated, the team defined a series of simplified mission phases derived from the mission planning of the Lunar Zebro project. Below, four mission phases are described in terms of their goals. That is, the goals act as requirements for the completion of the specific phase. The phases are visualised in Figure 1.

3.4.2 Phase 1: Dispatch

- Rovers have entered the Moon's surface.
- Rovers are spread around the lander.

3.4.3 Phase 2: Calibration

- Rover communication channels are initialised.
- Rover get information on goal direction.

3.4.4 Phase 3: Migration

- Rovers are in formation before migration.
- Rovers have arrived at location B.
- The path between location A and location B is mapped.
- Communication with the lander has been maintained.

3.4.5 Phase 4: Observation

- Photos of the surroundings of location B have been taken.
- The pictures of location B should be send to Earth.
- The next goal direction could be sent to the rovers.

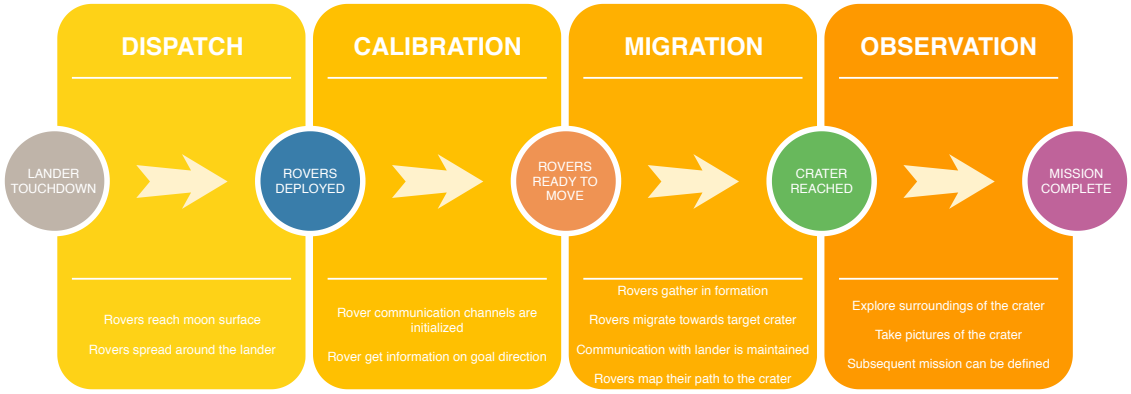


Figure 1: A diagram of the mission phases and their subgoals.

3.4.6 High-Level Overview of Approach

Our approach to be able to both simulate and validate the aforementioned mission description is to create a simulation environment in which the mission can be designed and subsequently simulated. There would be two approaches to this: make the simulator such that it can only simulate the one scenario that is defined, or make it such that anyone can design their own scenario in the simulator. While the latter is more work, it would be more preferable since it ensures that the simulation can be used for further research.

The simulation environment should be able to simulate the incoming sensor data, just like how a physical rover would have an incoming data stream from its sensors on which its algorithms make decisions. Hence, the decision was made to use Unity (for the environment) in combination with ROS (for the behaviour algorithm), as the two can simulate well what happens in a practical situation and both can be decoupled either to be connected to hardware or to be connected to a different behaviour algorithm.

The final product was developed in an agile way through weekly sprints using the scrum [8] methodology, as can be seen in section 3.4 in Appendix E. Apart from these weekly meetings, both the client and the supervisor were updated weekly on the changes that had been made and the features that were to be implemented.

3.4.7 Evaluation Criteria

The simulation will be evaluated through a set of criteria, based on the measurements of fitness defined in section 6.3.3 in Appendix E. In this section, these criteria will be defined and elaborated on. Note that distances / lengths are given in meters, as the simulation will be realistically scaled. Only the velocity of the rover will not representative of its real-world movement speed.

- **Relative path length**: Length of the mission, the distance as the crow flies between all consecutive mission goals.
- **Average distance covered**: The total distance covered divided by the amount of rovers in the simulation, the average traversed path length.
- **Average path efficiency**: The relative path length divided by the average distance covered, the ratio of the ideal path over the average traversed path.
- **Rovers that reached final goal**: The amount of rovers both have reached their final goal and are at the correct final location.
- **Amount of rover collisions**: Amount of times that the rovers collide with one another.
- **Amount of rovers getting stuck**: Amount of rovers that are stuck in their 'Waiting' state. I.e. a rover having reached its final goal but not being at the right location does not count towards this metric.
- **Total mission duration**: A timer is started when a mission is launched. This timer ends when all rovers stand still, so when they either are stuck or have reached their final goal.

4 Software Architecture

This section introduces the software architecture of all the systems that are developed. To meet the project requirements, a simulation is developed that allows for testing rover behaviour algorithms. First, subsection 4.1 and subsection 4.2 explain the design decisions and the software architecture of both the simulation environment and the rover behaviour algorithm respectively. Then, subsection 4.3 elaborates on the architecture of the connection between the two processes. Finally, subsection 4.4 shows how the system can be decoupled such that components of the software can be reused or adapted for different projects.

4.1 Simulation Environment

A simulation environment had to be developed in order to simulate the rovers on the Moon in a way that is representative of the actual mission. The simulation environment is a system that allows simulation of rovers in a physical lunar environment. The rovers can both traverse and perceive the environment.

An interface for the simulation environment is developed, which allows a rover behaviour algorithm to interact with the simulated rover by means of *Action*s. These *Action*s vary from a command to navigate to a specific location within line of sight of the rover, to communicating to another rover using radio communication. In turn, the simulation sends data back to the rover behaviour algorithm which can be information about what the rover perceives in its environment, incoming communication, and more. The scope of the simulation environment, the assumptions that were made in terms of rover capabilities and the interface of the environment are all explained in detail in section 5.

A common choice for developing robot simulations is *Gazebo*. *Gazebo* offers very detailed simulation of robotic joints and path planning. Furthermore, it easily connects to ROS, which is commonly used for behaviour algorithms for robots. Since, the feature set of *Gazebo* is limited, it becomes more difficult to use with large numbers of rovers and it is not user-friendly to non-developers that might want to run simulations in the future.

This lead to the decision to use *Unity* as the platform for the Simulation Environment. While the simulation of rigid bodies and locomotion is way less realistic in *Unity*, it allows for much quicker development because it is well-documented, easy to use, and offers many built-in features that can be used for the development of the simulation. Furthermore, within the scope of the project it is not important to simulate the mechanical properties and dynamics of the rover. Therefore, it is more productive to make the simulation less realistic and to focus more on the actual multi-rover behaviour under the assumption that the rover is able to traverse the environment.

4.2 Rover Behaviour Algorithm

The rover behaviour algorithm is separated from the simulation environment. The algorithm controls a single rover by processing incoming simulated sensor data from the simulation environment; making decisions based on this data and finally sending an action to the simulation environment which causes the simulated rover to for instance navigate, communicate, or wait.

Each simulated rover runs the same rover behaviour algorithm and this allows for a variable amount of rovers to fulfill the simulated mission that is defined in subsubsection 3.4.1. Development of the behaviour algorithm provides insight into which behaviour strategies can be feasible as the Lunar Zebro project continues.

As mentioned before, ROS is commonly used for behaviour algorithms for robots. There is not a specific requirement for the use of a framework when developing the rover behaviour algorithm, it is possible to implement it in any programming language that best suits the algorithm or developer. However, ROS provides several benefits to both the scalability and the maintainability of rover behaviour algorithm. During the simulation, a scalable amount of rovers are simulated at once. For this, ROS provides the concept of ROS nodes. Each rover can be connected to a set of ROS nodes that run the algorithm for that specific rover. If more rovers were to be simulated then additional ROS nodes for those rovers could be added with ease.

Furthermore, early in development it was clear that there would be very distinct data streams between the simulation environment and the rover behaviour algorithm. The simulation environment provides sensor data to the rover behaviour algorithm and in return the algorithm provides

actions to the simulation environment. When a series of data streams are required for a single rover and when many rovers are simulated at once, it becomes more difficult to both manage and monitor all these data streams. To manage all this data ROS topics were used. ROS topics allow processes to publish messages to a topic with a specific name and all the processes that are subscribed to that topic will receive all the messages published to the topic. For more information on the concepts of ROS that are used throughout this report, the reader is referred to the article *A gentle introduction to ROS* [9].

With these benefits in mind, the team made the decision to use ROS as a basis for the behaviour algorithm. A detailed description of the rover behaviour design and implementation can be found in section 6.

4.3 Connecting the Simulation Environment to the Rover Behaviour Algorithm

Based on the decision to use Unity as a simulation environment and ROS for the rover behaviour algorithm there has to be a way to connect these two processes. Several existing projects have successfully combined Unity and ROS [10] [11] [12]. Most of these projects rely on *ros_bridge*. Which is a ROS API that allows other applications to connect to a web socket in order to connect with ROS.

Additionally, the company *Siemens* developed a plugin for Unity called *ros-sharp* [1] which allows Unity to connect to a *ros_bridge*, including an API to directly use ROS topics from Unity. The connection between ROS and Unity was completely based on the *ros-sharp* plugin. The *ros-sharp* plugin is open source and during the project modifications were made to the plugin to allow the team to more efficiently communicate between the two processes. Figure 2 shows how a single rover connects to its ROS behaviour algorithm. Some topics are used for making the simulated rover perform certain actions, while other topics are used to provide information from various simulated sensors and systems to the rover behaviour algorithm. All of these topics are explained in detail in section 5.

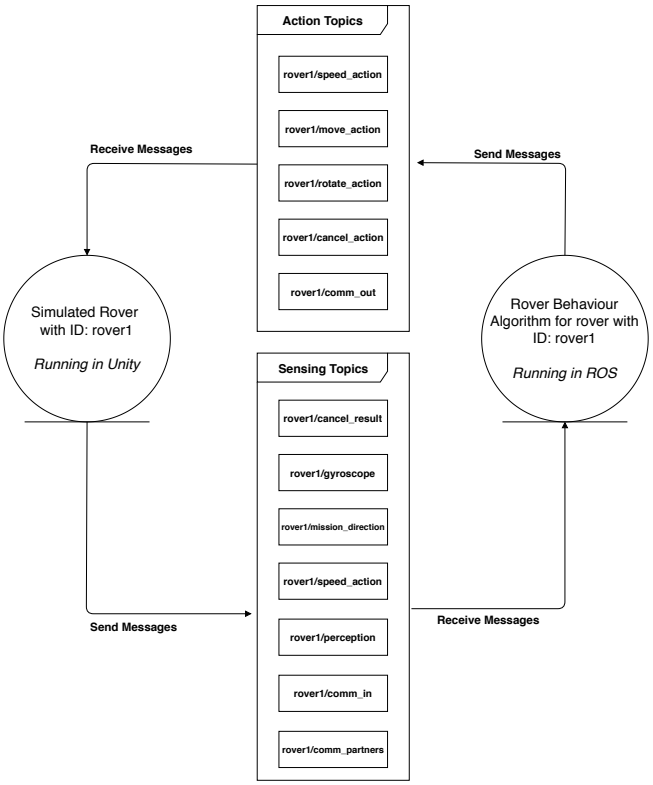


Figure 2: A visualisation of all the ROS topics that connect one simulated rover to its matching rover behaviour algorithm.

[1] https://github.com/siemens/ros-sharp

Furthermore, there was the need to separate rover behaviour algorithms. The behaviour algorithm of rover X must not conflict the instance of the behaviour algorithm that is simultaneously running for rover Y. For this ROS namespaces were used. The namespace of a rover is the unique ID that is assigned to that rover. Each instance of the behaviour algorithm runs in its own namespace and each simulated rover in the simulation environment connects to one single ROS namespace. This structure is visualised in Figure 3.

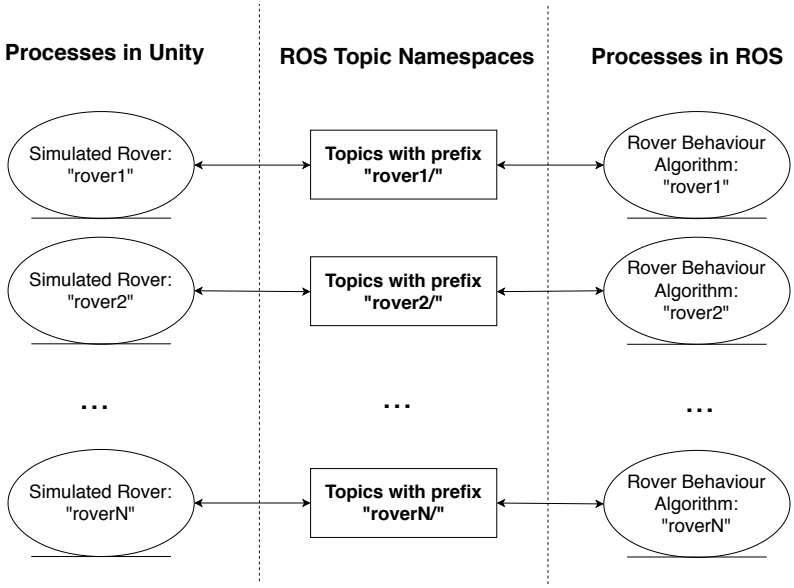


Figure 3: A visualisation of how N simulated rovers connect with their own rover behaviour algorithm instance. Each pair of processes is assigned a separate ROS topic namespace.

4.4 Decoupling the system

Due to the strictly defined interface that the set of ROS topics between Unity and ROS provide, it is possible to decouple both systems. If there is a need to develop a completely different behaviour algorithm then it is possible to create a completely different behaviour implementation. The sole requirement is that this implementation publishes correctly on the action topics and that it subscribes to the sensing topics. Similarly, if a different simulation were to be developed. The rover behaviour algorithm could be used on this system if the interface functions in the same way.

It is also possible to decouple the behaviour algorithm and to apply it to actual rover hardware. The behaviour algorithm works based on the assumption that there is a perception system and an existing low-level navigation system. These assumptions are described in detail in section 5. In future work it is possible to implement a perception system and a navigation system. When these two systems would function with actual hardware while still providing the same ROS topic interface then it is possible to run the existing ROS nodes from this project in combination with the new nodes. This would allow the rover behaviour algorithm to control a real rover instead of a simulated rover.

5 Simulation Environment

This section gives an overview of all the systems that were developed for the simulation. Technical details on all of these systems can be found in section 10.

subsection 5.1 first provides an overview of what an entire run through a simulation looks like from launching the application to simulating the rovers. Next, subsection 5.2 introduces the systems that were needed to simulate a rover.

5.1 Simulation Overview

Within the scope of the simulation, the following components are simulated: the lunar environment, rovers, a lander, lunar craters, and a configurable mission. Once the program starts, the simulation first connects with ROS. All the configurable parameters are loaded into the simulation and the behaviour algorithm is launched for each simulated rover. Next, it is possible to configure the simulation in the Scenario Editor. An image of the Scenario Editor can be found in Figure 4. The user can configure positions of rovers and the lander. Furthermore, the user can mark where craters are located on the terrain and determine a mission by means of specifying a series of mission goals.

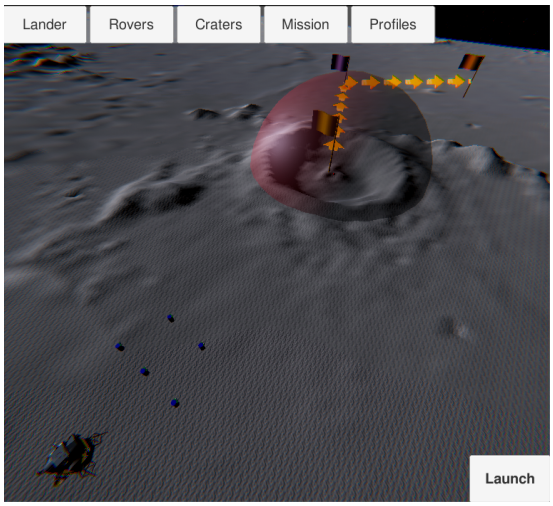


Figure 4: Image of the Scenario Editor. The lander and five rovers can be seen. Additionally one crater has been marked, as visualised by the red sphere. A mission consisting out of three steps can be seen in the distance, each flag represents one step.

Now that the user has created a scenario that is to be simulated, the simulation can be launched. The rovers will receive their mission and attempt to reach their mission location. Figure 5 shows a group of rovers walking towards their mission location. The eventual goal of the simulation is to have the rovers reach a crater such that the rovers can take pictures of this crater. Figure 6 shows a group of rovers that have reached a crater.

5.2 Simulated Rover

The most important entity in the simulation is the simulated rover. The rover has a series of simulated sensors and the rover is capable of performing specific action commands.

To begin with, the rover has a system that simulates the perception of the rover. In a real scenario this would be done by performing image processing on camera output. Within the simulation, the rover is able to perceive other entities within a specific field of view and range. The technical details of this system are described in subsection 10.1. Furthermore, the rover is able to navigate the simulation. The rover interprets movement commands and can move to a movement goal that is within line of sight. The navigation system allows moving, rotating in place, and cancelling an ongoing movement goal. More details can be found in subsection 10.2. The combination of simulated perception and navigation allows any behaviour algorithm to act by sending navigation commands based on the incoming perception data. Multiple rovers will be deployed

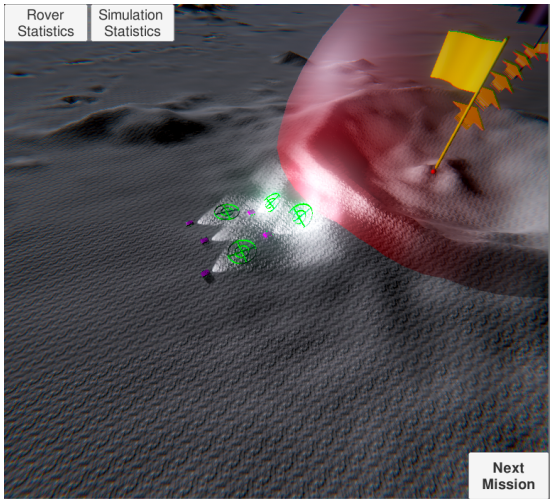


Figure 5: Image of the simulation in progress. The rovers are being controlled by their behaviour algorithms and are walking towards their first mission goal.

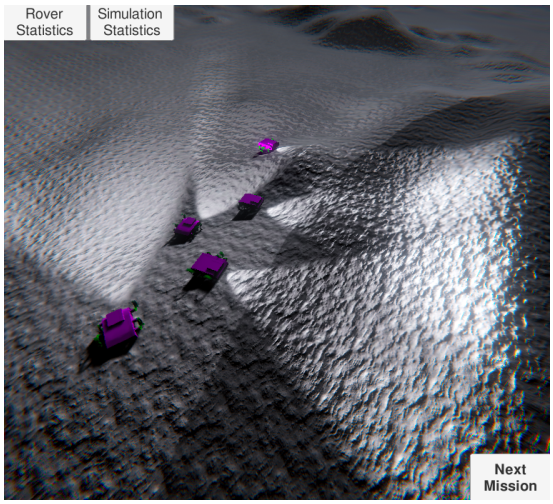


Figure 6: Image of a group of rovers that have achieved their first mission goal: reaching the crater.

simultaneously to collectively complete a mission. Therefore, these rovers need to be able to communicate. Communication is simulated by allowing rovers to send messages to one another when within communication range and within line of sight. Many assumptions about existing communication hardware and protocols are made and these are described in subsection 10.4. Finally, the rover also has a simulated gyroscope sensor (subsection 10.3) and a system for taking photos as a final step of the simulated scenario (subsection 10.5). Within the scenario editor it is possible to configure the positioning of the configured rovers, along with a lander, craters and mission goals. This allows a user to set up and simulate various scenarios to test the rover behaviour algorithm in specific edge cases. Details on the scenario editor can be found in subsection 10.6.

6 Rover Behaviour

The rover behaviour algorithm is the decision making process that determines the next action of the rover. The rover behaviour algorithm is developed using ROS and written in the C++ programming language. The algorithm that each rover executes is identical. Decisions are based on incoming data streams from the simulation environment, for instance from the perception of the rover. Decisions are executed by sending *Actions* to the simulation environment, such as *Actions* to move to a certain position.

The ultimate goal of the rover behaviour is to find and observe a certain crater. In order to achieve this goal multiple rovers need to work together. The following subsections go through the consecutive design steps made of the algorithm. The first step is to make the behaviour algorithm work for a single rover. This includes high-level path finding and static obstacle avoidance. The next step is to expand this to multiple rovers. Now rovers also need dynamic obstacle avoidance to avoid collisions with each other. Subsequently, the rovers need to move to the edge of the crater to observe it. Communication was added to implement a daisy chain, which is a technique to keep communication between the lander and the rovers. Finally, support for multiple mission goals were implemented such that rovers can go to a next mission goal after another has been completed. The addition made in the last step results in the complete rover behaviour as found in the current version of the simulation.

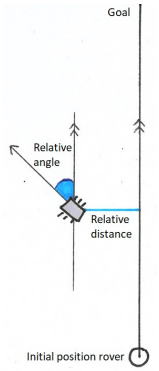
6.1 Single Rover Behaviour

The first scenario consists of only one rover that can perceive static obstacles. Its goal is to move to a user-defined destination without bumping into obstacles. In order to do this, a path finding algorithm with obstacle avoidance is needed.

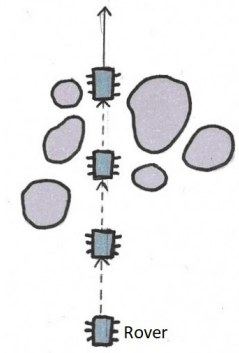
The single rover behaviour algorithm is explained by first elaborating on the goals, discussing the challenges that were encountered, and explaining the solutions to these challenges. Detailed descriptions of the calculations are given in subsection 11.1 and *[Redacted]* found in the chapter devoted to technical specifications of the rover behaviour. Finally, the link between all details come together in an overview of the pseudocode of the algorithm.

6.1.1 Path Finding

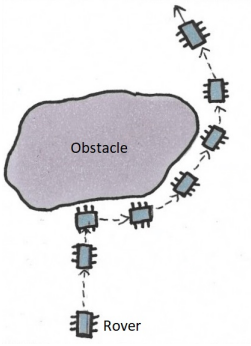
Whilst traversing in small steps at a time, the rover needs to find the next mission goal. One of the challenges is to deal with the fact that there is no GPS on the Moon and rovers are not aware of their absolute position on the Moon. To overcome this issue, the relative position of a rover to a certain path is kept track off, as can be seen in Figure 7a. The path is the line between the initial position of the rover and its goal location. In an ideal situation, the rover can simply follow this path. However, obstacles cause the rover to walk away from its path. This raises another issue on how to decide in what direction it should walk next. Continuing straight ahead after the rover has rotated would only lead the rover being further away from its relative path. Therefore, the rover aims to always walk back to its original path between its starting position and its goal location. Details of the implementation are discussed in subsection 11.1.



(a) Image of the calculations required for the rover to estimate its position relative to its goal.



(b) Obstacle avoidance: the rover perceives that there is a way to pass through the obstacles.



(c) Obstacle avoidance: the rover perceives that it cannot pass through the obstacle. It walks towards the obstacle and walks around it while staying near the obstacle.

Figure 7: Illustrations of the components implemented for the single rover behaviour.

6.1.2 Static Obstacle Avoidance

While moving, the rover needs to avoid static obstacles. On the Moon, a typical obstacle to avoid is a large rock. One challenge is that the rover does not have a map of the environment and only perceives obstacles once it is within its perception range. It should therefore only move to positions in its perception range. Another challenge is that the rover needs to know where it can navigate to without colliding with an obstacle. To solve this problem rovers scan the area by means of raycasting, as described in subsection 10.1. If a ray hits an obstacle, then the rover knows where the obstacle is. With this information it can calculate where it can go. Figure 7b shows how a rover moves around an obstacle. First it walks close to the obstacle, then it turns towards the right (never to the left), and continues. Figure 7c shows a case with multiple obstacles. If there is enough space, the rover will walk past them. If the rover cannot move forward because there are obstacles everywhere in its view, the rover rotates in place and tries to find a new path. Technical details can be found in *[Redacted]*.

6.1.3 Algorithm Overview

The pseudocode gives an overview of the algorithm of the single-rover behaviour. It shows how all details concerning path finding and static obstacle avoidance described in subsection 11.1 and *[Redacted]* come together.

▷ Update the rover's relative position
Update the angle towards the path ▷ Left is negative, right is positive
Update the distance towards the path ▷ Left is negative, right is positive

▷ Calculate new coordinates
if rover is close to the path **then** ▷ Close is defined as 2 metres
    New coordinates are straight ahead
else
    New coordinates are towards the path with a fixed angle ▷ Fixed angle is set on 20 degrees
end if

▷ Check new coordinates
if the new coordinates are out of view **then**

Update new coordinates to walk to the edge of its view to make a sharp turn move towards the path
end if

▷ Obstacle avoidance
while true **do**
  if the rover can walk freely **then** ▷ There are no obstacles
    Do move to the new coordinates
    break
  end if
  if There is a far obstacle in view **then** ▷ The obstacle is far
    Walk close to the obstacle
    break
  end if
  Scan a new angle of the field of view (from left to right) ▷ The obstacle is close
  if The field of view is completely scanned and obstacles are everywhere in its view **then**
    Do rotate ▷ The rotate angle is set on 60 degrees
    break
  end if
end while

6.2 Multi-Rover Behaviour

[Redacted]

6.2.1 Dynamic Obstacle Avoidance

[Redacted]

6.2.2 Cancel Action

[Redacted]

6.2.3 Formation

[Redacted]

6.2.4 Algorithm Overview

[Redacted]

6.3 Crater Observation

Now that multiple rovers can migrate at the same time while finding a path and avoiding (static and dynamic) obstacles, it is time to involve the crater. The three goals are moving to crater, standing still on the edge of the crater, and finally observing the crater by capturing pictures.

6.3.1 Move to the Crater

There are several goals for taking into account crater observation within the rover behaviour. The first goal is to walk to the crater. Since general path finding is already implemented, few adaptations need to be made. The challenge here is to improve the efficiency of the path finding. When the rovers are spawned they know the direction and approximate distance towards the crater, provided that it has been labeled as mission goal. The direction is used to calculate their path towards the crater. This completes the first goal of reaching the crater. To make the path more efficient, the rovers are not programmed to stay close to the path anymore (i.e. move with a set angle towards the path if it is too far away), but it is programmed to move in the angle directly towards the crater. How this is done exactly can be read in subsection 11.3.

6.3.2 Stop on the Edge of the Crater

After the rovers have reached the crater, they should stop on the edge of the crater. The assumption was made that the rovers can perceive Terrain as CraterTerrain when they observe the crater bubble as shown in Figure 8.

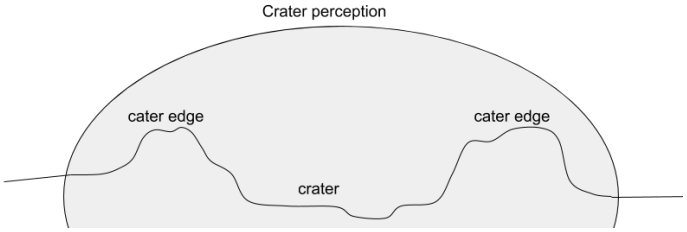
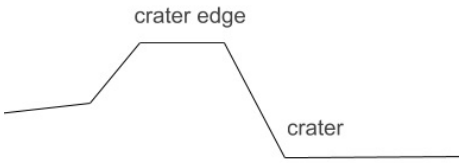
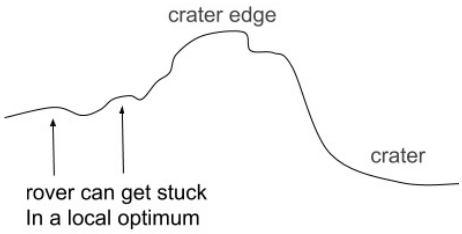


Figure 8: Image of a crater. The rovers perceive the crater as CraterTerrain.

This second goal is reached by implementing a gyroscope, which measures the angle of the rover compared to the Moon. This way the rover can know in what angle it is moving vertically. A difficulty with the use of a gyroscope is that it is not always accurate and that the crater edge can be bumpy. In the ideal situation, the rovers should keep moving when they gyroscope is > 0 (moving uphill) and stop moving when it is < 0 (moving downhill) after they saw the crater. However, the rovers could then stop at a local optimum rather than the global optimum on top of the edge as is visualised in *[Redacted]*. To solve this problem the rovers should walk at least their maximum view distance, after they have perceived the crater. When the gyroscope is $<= 0$ it is assumed that the rover is at the top.



(a) An ideal situation for determining whether the rover is on the crater edge.



(b) A more realistic crater edge with local optima that make it difficult to determine whether a rover is on the crater edge.

Figure 9: Illustration of the ideal crater edge and the difficulties that arise when simulating more representative crater terrain.

6.3.3 Observe the Crater

The third goal is observing the crater. When a rover is on the edge, it should rotate and take pictures of the environment. These pictures will be sent to the lander, which has more computational power and can verify that the rover has indeed found the edge of the crater. This goal is implemented by adding a 'ReachedDestination' state.

6.4 Communication

As of this point, the rovers are able to complete the mission of traversing to a goal location. Even though there are multiple rovers, there is no form of interaction except for avoiding collisions with one another. By working together, the performance could be improved. In order to work together, communication between the rovers is essential. An entire back-end, on the Unity and the ROS side, is implemented to enable this. At this point, when a rover perceives a crater, it broadcasts this to all rovers within its communication range. In addition, the daisy chain, which is described

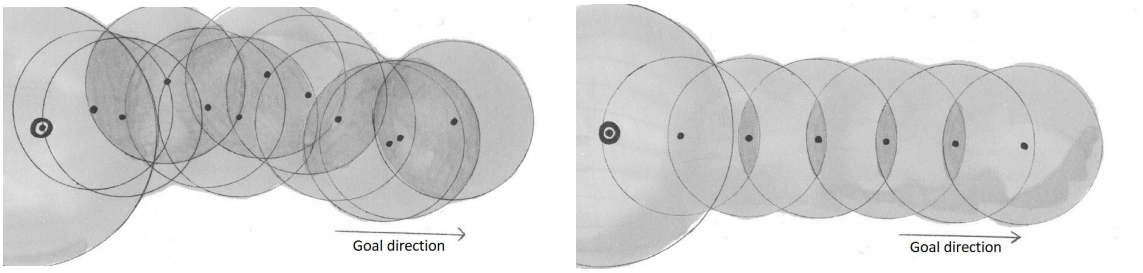
next, is dependent on this communication system. Communication between rovers can be used for many other improvements. In subsubsection 7.2.2 these are discussed.

6.5 Daisy Chain

A daisy chain is defined as a series of associated or connected people or things. This was used as an inspiration to let the rovers maintain communication with each other and the lander. Every rover only has a limited communication range and if a rover splits up from the group, all communication is lost. It is important that the communication stays intact since the lander should be able communicate to the rovers. It can either give the rovers a new mission, let the rovers return to the lander's position, or let the rovers stop moving. A daisy chain was implemented so that the rovers stand next to each other where the first rover is close to the lander and all consecutive rovers within the communication range of the previous one. This section describes what an optimal daisy looks like, what complication arise during implementation, and how a simpler version was created. In subsubsection 7.2.2 future improvements on how to implement an optimised daisy chain are discussed. Note that daisy chains have not been taken into account for the final scenarios.

6.5.1 Optimised Version

In an optimised daisy chain Figure 10b the rovers stand in a straight line from the lander to the goal location. The distance between the rovers is maximised. A rover is just inside of the communication range of the previous rover. This way the least amount of rovers is needed to reach the furthest.



(a) A non-optimal daisy chain that results in more rovers being part of the daisy chain than required.

(b) An optimal daisy chain where the communication range of each rover is maximized.

Figure 10: Examples of daisy chains formed from the lander (marked with a circle on the left) to the direction of the mission goal. The dots represent rovers that are part of the daisy chain and the circle around each rover represents their communication range.

6.5.2 Challenges Optimised Version

There are some challenges that arise when implementing this optimised version. First of all, the rovers should always move in a formation to be able to form a functional daisy chain. Without any obstacles this criteria is very manageable to implement. However, if there are (many) obstacles, the rovers move in different directions and ultimately split from the group. Secondly, if we assume that the above issue has been solved, there still remains the challenge that rovers do not move in a straight line one after the another. Figure 10a is much more realistic in that case than the optimal version from Figure 10b. Finally, it is assumed that the above problems are solved and the daisy chain works perfectly. A new challenge arises with the introduction of multiple missions. The current daisy chain could stay in place. This way it still functions, however, it might be inefficient if the next mission is on the other side of the lander. The rovers that have joined the daisy chain should start moving again in order to form a new daisy chain without losing connection to the other rovers. This is an entire new situation. Possible ways to overcome these challenges can be read in subsubsection 7.2.2.

6.5.3 Proof of Concept

The implementation of an optimised version was out of scope for this project. As mentioned before, the challenges and ideas on how to overcome than are listed. In addition, a proof of concept was implemented where the rovers form a (non-optimised) daisy chain like the one displayed in Figure 10a. This version is based on the idea that a rover closest to the lander should join the daisy chain and receives the task of assigning the next rover to join. The distance to the next rover should be maximised but still within communication range of the rover to last join the chain. Therefore, the rover that last joined keeps track of how many rovers it can communicate to. If this number decreases, it means that one rover travelled outside of its communication range and the rovers assumes that the other rovers will soon leave the range too. It quickly assigns the next rover to join the daisy chain and the process is repeated. Details of this implementation can be found in subsection 11.4.

6.6 Consecutive Missions

The rovers should be able to complete multiple missions in a row. There are two types of goals: a crater goal and a non-crater goal. When a rover has a crater goal it should stop on the edge of the crater. When the rover receives a non-crater goal it should stop moving when the goal is within its field of view. In that case it should not rotate or take pictures.

An overview of the states is given in Figure 11.

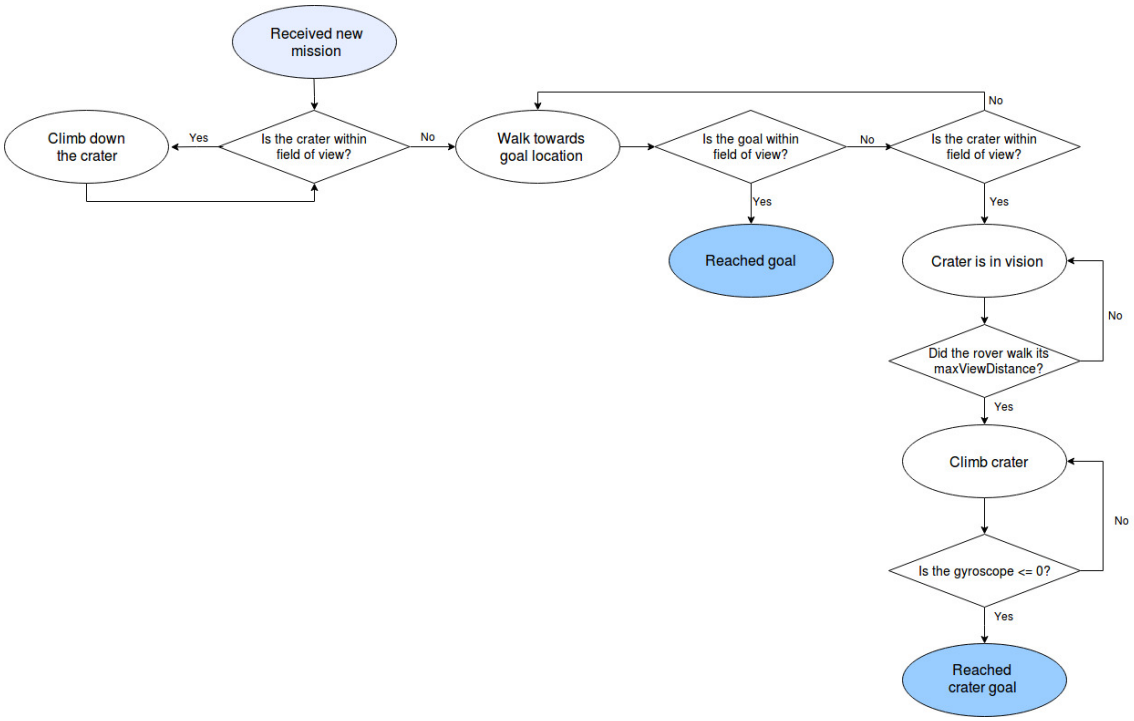


Figure 11: The flowchart displaying the mission states of the rover.

7 Results and Recommendations

The results of the project are described in this section. Discussion of the implemented features results in a series of suggested future improvements.

First, subsection 7.1 and subsection 7.2 discuss the results of the developed simulation and rover behaviour algorithm. Next, subsection 7.3 discusses the quality of the developed software and the metrics that were used to measure software quality. Finally, subsection 7.4 aggregates results from a series of simulations that were repeatedly run on representative scenarios.

7.1 Simulation Environment

The following subsections discuss the results of the development of the simulation environment. Halfway through the project the focus was shifted from adding more features to improving existing features. This resulted in a more stable platform for testing rover behaviour algorithm.

Future improvements are discussed where potential improvements to existing features and ideas for new features are elaborated.

7.1.1 Features

The navigation and perception systems were tested thoroughly during development of the rover behaviour algorithm. After lots of improvements, these systems are found to be stable and representative of a real-world scenario, provided that the assumptions made in section 10 hold.

Next to the basic perception and navigation systems, a more specialized functions of the rover is also simulated. Observation of craters is an important aspect for completing missions as described in section 3. To simulate the observation of craters a system was implemented that allows the simulated rovers to take photos from their perspective. These photos can then be inspected to judge whether the rover behaviour algorithm was capable of taking usable photos of a crater.

Furthermore, since multiple rovers are deployed at once, the rovers need to be able to communicate. A communication system was implemented that allows rovers to communicate with one another when within line of sight and communication range. However, due to time constraints, the simulated communication is not used to its full potential by the rover behaviour algorithm.

During the development of the connection between the simulation and the rover behaviour algorithm it became evident that the interface between both systems needed to be clearly defined. A clear description and unambiguous agreements on the functionality of specific actions that a simulated rover can perform are vital for efficient development. Early on in the project, miscommunication would cause development time to be lost which could have been prevented by discussing the interface more thoroughly. All the interfaces between the simulation and the rover behaviour algorithm are described in detail in section 10.

Furthermore, all of the actions that a simulated rover can perform are deterministic and always provide a clearly defined response to the rover behaviour algorithm.

The simulation software is developed such that a person without programming experience is able to run simulations. A clearly explained interface allows a user to test and experiment with a previously developed rover behaviour algorithm. Scenarios can be configured intuitively by dragging entities around with a mouse. Furthermore, scenarios can be stored and exported to allow users to re-run specific scenarios easily.

To further aid users in understanding the simulation, a set of statistics is displayed. The user can use these statistics to judge the effectiveness of the rover behaviour algorithm in specific scenarios.

Finally, the simulation was developed to be configurable. This allows for experimenting with different rover capabilities. For instance, the field of view of the rover camera can be changed to test the rover behaviour with different camera setups. All of the configurable parameters are described in detail in subsection C.2

7.1.2 Future Improvements

One issue that arose as the simulated rovers became more complex was that performance started to deteriorate. With just the perception and navigation system it was possible to simulate more

than 100 rovers. However, with the final product the performance declined significantly when more than 20 rovers are simulated. The hardware used for testing the simulation can be found in subsection C.4.

Additionally, battery, lighting and temperature simulation were planned for development at the start of the project were later defined as beyond the scope of the project. It was more productive to make the simulation more reliable and more accessible by developing a user interface.

The first system that was not implemented is a simulated battery. In this system the rovers would have a limited power budget and their battery would drain based on their actions. The rovers are capable of recharging their battery by using solar panels. Which introduces the next system, lighting simulation.

To simulate charging with solar panels the lighting on the moon would need to be simulated correctly. This is important because the mission can take place near the far side of the moon where there is limited sunlight.

Once lighting simulation is implemented, it is also useful to simulate the temperature of the rovers. The hardware of a rover needs to stay within operating temperature, otherwise it will shut down. Simulating the temperature of the rover would make the simulation more representative since the rovers would need to evade shadows.

To further improve the simulation software, additional statistics could be added to expand the information that can be gathered. This information can be used to better assess the fitness criteria of the rover behaviour algorithm more accurately.

Finally, a time scaling system could be implemented. As described in subsection 10.7, correctly scaled lunar terrain was implemented. However, it was impractical to use this during testing because it would take too long for the rovers to complete a mission on realistic scale. Thus, a system for speeding up time would help in testing on a realistic scale.

7.2 Rover Behaviour Algorithm

The following subsections discuss the rover behaviour algorithm that was developed. The most effective way to develop the behaviour was by means of simple rule based strategies. Four different iterations were developed of the rover behaviour algorithm which each increase the complexity of the algorithm by expanding the scope of the mission. The four development stages of the algorithm were: single rover behaviour, multiple rover behaviour, finding the edge of a crater and rover communication.

7.2.1 Features

The behaviour of a single rover and obstacle avoidance was the first stage of developing a rover behaviour algorithm. In this scenario, the rover is only required to evade stationary obstacles since there are no other rovers. Extensive testing lead to the conclusion that a single rover is always able to reach its goal without colliding with obstacles.

However, sometimes the path that is taken is sub optimal. This happens because the rover only acts based on its directly perceived surroundings. During development, more simple rules resulted in a higher chance of the rover reaching its goal. More complex strategies resulted in more edge cases that had to be covered, which was difficult since the lunar terrain can very significantly.

The multiple rover behaviour algorithm functions similarly to the single rover behaviour. Rovers now have to take other rovers into account that also continuously move. A simple traffic rule was the most effective to prevent rovers from colliding with one another. Any traffic to the right of a rover has right of way. Thus, when a rover perceives a rover to its right then stops moving and lets the other rover pass.

This could still result in situations where rovers were waiting for each other to continue moving. To resolve this the rovers rotate in place when a rover to its right does not move for some time. This allows the rover to look for a different path to move away from the rover. Additionally, rovers try to keep more distance to other rovers than they would to obstacles.

As discussed in the previous section, the performance of the simulation environment would deteriorate when more than 20 rovers were simulated at once. This also had effects on the rover

behaviour algorithm. The ROS nodes that are used for the algorithm would automatically gain a lower update frequency due to the lack of available processing time.

The lower update frequency would decrease the accuracy of the localization calculations in the algorithm and at times this causes rovers to completely lose track of their position.

To complete a mission, rovers need to find a crater edge, stand on top of it and take photos. Determining whether a rover is on top of a crater edge is difficult. It can be seen as the classic hill-climbing problem where it literally needs to be determined whether a rover is at the top of a hill and not at a local optimum.

To better determine the edge of a crater a gyroscope is simulated. Using the data of this gyroscope the rovers often end up on the edge of the crater but sometimes a rover will get stuck on a local optimum. The rovers are also programmed to walk slower when climbing the edge of the crater, this allows for more accurate gyroscope sensor readings.

Communication is currently only used to notify other rovers that the crater has been found. Furthermore, a proof of concept version was developed of the daisy chain system to maintain contact with the lander.

7.2.2 Future Improvements

The most important improvement that can be made at this point is a better usage of the communication. Communication can help rovers to stay together as a group. It can also help to reduce errors, for example: The rovers always estimate their position relative to the lander or their mission goal. Errors will accumulate in this estimation, the longer a rover walks, the higher it's error will become. This can be reduced by letting the rovers communicate their locations with each other and averaging the results.

The daisy chain concept that is required to maintain contact with the lander has not been completely implemented. A proof of concept version was made that showed that the system has potential but it is far from optimal. The developed concepts for daisy chaining are described in detail in subsection 6.5.

Another point of improvement is the rovers perception. Currently the rover can only look straight ahead and not up or down. If the rover would be able to also look downwards or upwards, then decision making could improve and the chance that rovers will collide with one and other will be reduced even further.

The issue that rover stop moving at a local optimum rather than a global one could be solved as follows. To verify that a rover is indeed located on the edge of the crater, the rover could rotate in place to check if it can see any terrain. If the rover perceives terrain while rotating, then it has not reached the highest point on the edge of the crater yet.

7.3 Software Quality Metrics

The following subsections describe the efforts that were made to improve code quality. Software testing, static analysis, code style consistency and documentation were all enforced to improve code base maintainability. Several quality metrics were monitored and continuous integration helped with keeping track of these metrics.

7.3.1 Code Testing Coverage

The rover behaviour algorithm was tested extensively and this has proven to be very valuable throughout the project because it uncovered some bugs that were difficult to reproduce.

Line coverage of the rover behaviour algorithm is 100%. Note that a large portion of ROS related code has been excluded from the testing coverage reports. The reason for this is that testing ROS code was difficult because it requires the ros_bridge server (described in section 4) to be launched. Running a server during testing was difficult to do locally and we were unable to set it up for continuous integration within the scope of the project.

To enforce code quality, continuous integration was used for rover behaviour algorithm during the entire project. The continuous integration was required to pass compilation checks before a pull request could be merged. This ensured that the master branch always compiled correctly.

Finally, the simulation code was not tested due to difficulties with testing Unity code. It was not possible to add Unity to continuous integration because this required an expensive Unity Pro license. Testing was left beyond the scope of the project because testing game code requires the Unity game to be running and the tools available for this were too complex to set up.

7.3.2 Static Analysis Tools

The simulation was developed in C# using the Rider IDE [2]. Rider uses Resharper as a refactoring and static analysis tool. Resharper provided warnings of problematic code, these were addressed to improve the quality of the code. However, due to the use of Unity APIs, there were some false positives and these were ignored.

The rover behaviour algorithm was developed in C++ using the CLion IDE [3]. CLion has a built-in static analysis tool called Clang-Tidy. Warnings from Clang-Tidy were addressed to improve code quality. Additionally, CppCheck was used as a second static analysis tool. All the warnings from CppCheck were addressed and the final product has no remaining CppCheck warnings.

7.3.3 Code Style Checking

As mentioned in the previous subsection. The Rider IDE was used with Resharper to develop the simulation. Resharper offers code reformatting tools for C. Throughout the project the reformatting tools were used to ensure a coherent code style.

For the rover behaviour algorithm, CLion was used. CLion also offers a built-in refactoring tool which was also used to ensure coherent code style for the C++ code base.

Documentation style and consistency was enforced through extensive code reviews during pull requests.

7.3.4 Documentation

All methods and classes in the code bases of the simulation and the rover behaviour algorithm are documented. During pull requests it was enforced that all code had to be documented. Reviewing the documentation also aided in ensuring that the entire team understood all new implementations. Furthermore, when documentation was unclear, a change would be requested.

7.4 Mission Simulation Results

To test the core functionalities of simulation, three scenarios have been set up, each focusing on a different rover behavioural task:

1. Observation of a crater: for validating the observational requirement.
2. Navigation through an obstacle maze: to validate the path planning.
3. Observing a crater and returning to the lander: to validate a mission cycle.

The evaluation criteria, as defined in subsubsection 3.4.7, will be used to evaluate a scenario. Based on these results, a conclusion will be formulated in section 8. The simulation results can be found in table Table 2.

[2] `https://www.jetbrains.com/rider/`
[3] `https://www.jetbrains.com/clion/`

Scenario .Run	Relative path length	Average distance covered	Average path efficiency	Total distance covered	Rovers that reached final goal	Amount of rover collisions	Amount of rovers getting stuck	Total mission duration in mm:ss
Scenario 1: Observing a crater								
1.1	144,7477	1944,34	0,0744457	9721,7	3/5	0	0	1:50
1.2	144,7477	3036,77	0,0476650	15318,81	5/5	0	0	2:18
1.3	144,7477	1946,88	0,0743485	9734,38	5/5	0	0	1:35
1.4	144,7477	1961,95	0,073777	9809,73	4/5	0	0	2:19
1.5	144,7477	2597,13	0,0557337	12985,67	5/5	0	0	1:20
Average:		2297,414	0,0651940	11514,058	4/5	0	0	1:52
Scenario 2: Obstacle aviodance								
2.1	236,4748	2694,7	0,087755	13473,47	5/5	0	0	3:22
2.2	236,4748	2225,74	0,106245	11128,87	5/5	0	0	4:33
2.3	236,4748	1249,75	0,189217	12463,75	4/5	0	0	3:29
2.4	236,4748	4090,91	0,057805	20454,53	4/5	2	0	3:46
2.5	236,4748	1561,4	0,151450	7809,21	5/5	0	0	1:33
Average:		2364,5	0,118494	13065,966	5/5	0	0	3:20
Scenario 3: Returning to lander								
3.1	152,8776	625,45	0,244428	3127,26	2/5	0	0	2:40
3.2	152,8776	1449,78	0,105448	7248,9	2/5	0	0	2:40
3.3	152,8776	1259,18	0,121410	6295,89	2/5	0	0	2:44
3.4	152,8776	1279,99	0,119437	6399,98	2/5	0	0	3:33
3.5	152,8776	834,54	0,18319	4172,69	5/5	0	0	2:03
Average:		1089,788	0,154782	5448,944	2/5	0	0	2:44

Table 2: Several runs of scenarios within LunarSim.

8 Conclusion

The developed simulation environment provides a basis that can be used to test a variety of mission scenarios for the Lunar Zebro rover. The user interface allows users to experiment and test existing rover behaviour algorithms with different hardware configurations. Hardware configurations can be specified through an extensive list of parameters. The simulation of the navigation and perception is representative that were specified for these systems hold. However, lighting, temperature and battery levels can be simulated to make the simulation software more representative of the challenges in an actual space mission.

Furthermore, the rover behaviour algorithm that was developed to test the simulation provides a reference point for other behaviour algorithms that could be developed in the future. Several techniques have been implemented and described extensively. Simpler navigation rules inspired by traffic rules resulted in behaviour that is intelligent enough to consistently move simulated rovers towards their goal. More complex approaches often create more edge cases that can cause unforeseen problems. Maintaining contact with the lander is an important aspect of the simulated mission, for this the daisy chain concept was developed. However, more development time is needed to implement and test this proposed solution effectively.

Ensuing the development of a simulation environment to test rover mission scenarios, an assessment of the performance of the behaviour algorithm has been made by setting up various scenarios that test essential partitions of the simulation. From the results of these scenarios in Table 2 the following can be concluded:

1. Observation of the crater happens accurately, however not entirely reliably, since in a total of over five runs 88% of the rovers had been able to complete this scenario successfully.
2. Obstacle avoidance works close to flawlessly, since 92% of the rovers were successfully able to traverse an overly complex obstacle scenario before reaching their final goal location. The ones that did not make the final goal did make it through the obstacle maze but did not arrive at the correct final goal, so from this we can conclude that the obstacle avoidance is both accurate and reliable.
3. The performance of the overall mission cycle is neither accurate nor reliable from the point of returning to its original point, since the behavioral algorithm gets stuck in observational mode once having observed a crater. By implementation, the rovers prioritize observing over going to their next goal, which makes the amount of rovers that are able to reach their final goal decline. This however, can be improved to the extent that the rovers prioritize reaching their next goal, such that more are able to go back to their starting location.

From the above mentioned findings, it can be concluded that the core functionalities of the simulation environment and the rover behaviour algorithm perform adequately, though the rover behaviour algorithm can be expanded to support returning to its initial location. For this to happen, more testing has to be done and improvements have to be made to the rover behaviour algorithm. A proof of concept has been recorded by means of validating individual behavioural processes where future research can build upon. With the current state of the simulation, the team is confident that the simulation can be expanded such that a LUFAR scenario can be implemented and validated.

9 Ethical Implications

In this chapter the ethical implications of the project are discussed. The main concerns are that the feasibility study must be representative in order to give a good representation of the problem. The other concern is the impact that the project can have on the environment and on other projects.

9.1 Representative Feasibility Study

The main goal of the project is to create a representative feasibility study. The developers of the simulation and rover behaviour have a responsibility to give a fair and honest representation of what is achievable and what is not. It is easy to unintentionally 'cheat' in a simulation. The simulation can be used to send information to the simulated rovers that real rovers on the moon could never possibly know. However, it is beyond the scope of the project to simulate everything in low level detail. In section 5 assumptions about the capabilities of the actual rovers are discussed in detail, in order to give a fair representation of what the rovers can do. These assumptions were based on the literature study that was done at the start of the project, which can be found in Appendix E.

9.2 Impact on the Environment

Space missions usually have a negative impact on the environment. A lot of CO2 is generated on a space mission [13] and there is also the problem of space debris [14], which happens when parts of mission equipment remain in space. This project does not directly cause there problems, but contribution to this is inevitable when the actual physical mission takes place. Therefore it is important for the users of the software to think about the impact that this project has on the environment. There is a trade-off between the environmental impact and the value of the research. This feasibility study shows that sending 20 rovers to the moon is better than sending one when you look at the research value. However if 20 rovers are send this will have a larger impact on the environment than if only one rover is send. To minimize the environmental impact it important that the software is thoroughly tested before it is send to space. The chance that rovers get stuck, lost of break down should be minimized as much as possible, such that the rovers do not pollute the moon [15]. It is very difficult to remove the rovers from the moon in case something goes wrong, so reliability is a factor that plays a large factor into the extent of ethical implications of the mission.

10 Technical Specifications Simulation Environment

This section explains how the simulation environment works in detail. This is the continuation of section 5. The simulation systems that are required to simulate the rover are discussed in detail. For each system, a series of assumptions is made about the capabilities and the hardware of the actual rover. Furthermore, the implementation and its interface are described. The systems are rover perception (subsection 10.1), rover navigation (subsection 10.1), the gyroscope sensor of the rover (subsection 10.3) and communication between rovers (subsection 10.4). Finally, subsection 10.6 explains the user interface of the simulation and subsection 10.7 describes what went into ensuring scalability and accuracy of the simulation.

10.1 Perception

The perception system simulates the vision of the rover. The Lunar Zebro rover will be equipped with RGB camera(s) that allow it to make actions based on its immediate surroundings. The perception of the rover must be simulated since it serves as a basis for most decisions in the behaviour algorithm.

10.1.1 Assumptions

- The actual rover is capable of identifying other entities in its environment using image processing.
- The actual rover can distinguish between: other rovers, obstacles (rocks), terrain, the lander, and the edge of a crater.
- The actual rover can estimate the distance of each object that it distinguishes in its camera.
- There is a range in which the rover is able to reliably identify its surroundings.

10.1.2 Implementation

The perception system was implemented by performing a series of raycasts from the rover to its environment. When a raycast hits an object that is marked as perceivable, i.e. another rover or the terrain surface, then this results in a perception that is sent to the rover behaviour algorithm. This process is shown in Figure 12. Because the client is interested in experimenting with different setups for the camera all of the following was made configurable: the field of view of the camera, the number of raycasts within the field of view (the accuracy of the simulated perception), and the range of the perception (the length of the raycasts). Note that the rays are only cast on the rover's XZ plane. In other words, the rover does not have perception up or down, only straight ahead. This was done to simplify both the simulation and the behaviour algorithm for quicker iterations.

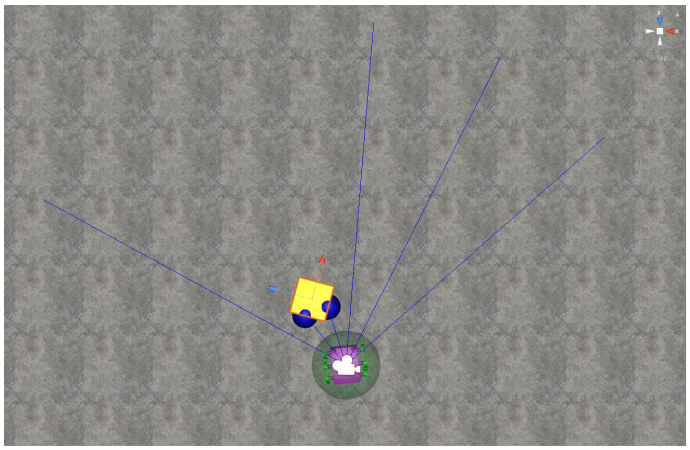


Figure 12: Image of the rover perceiving an object using the perception system. The blue lines represent the perception rays. Two of the rays intersect with the yellow cube at the location of the two blue spheres.

10.1.3 Interface

ROS Topic Name	"roverX/perception"
ROS Message Type	PerceptionData
Publisher	Simulation Environment
Subscriber	Rover Behaviour Algorithm

Published at a fixed rate by the simulation environment to the rover behaviour algorithm. For each perception it contains: the angle of the perception relative to the rover, the distance from the rover to the perception and finally what the rover perceives.

10.2 Navigation

The navigation system allows the simulated rover to traverse the simulated environment. The Lunar Zebro is capable of walking around on the moon and it can rotate in place.

10.2.1 Assumptions

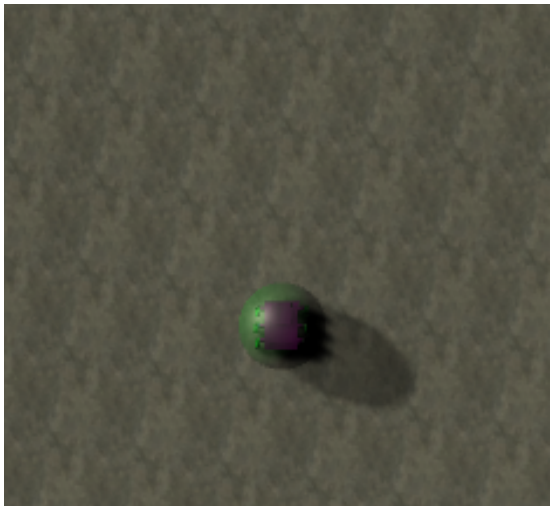
- The actual rover can traverse flat terrain without the risk of getting stuck due to small debris or dust.
- The actual rover has a low-level path planning algorithm that translates a goal to move to a location to commands for its motors.
- The actual rover can walk to a location that is within its field of view.
- The actual rover will reject a movement goal from a higher level path planning algorithm when the movement goal is outside of its field of view.
- The actual rover is able to estimate its distance to its current movement goal at all times.
- The actual rover is able to cancel its current movement goal.
- The actual rover is can regulate the speed at which it moves.

10.2.2 Implementation

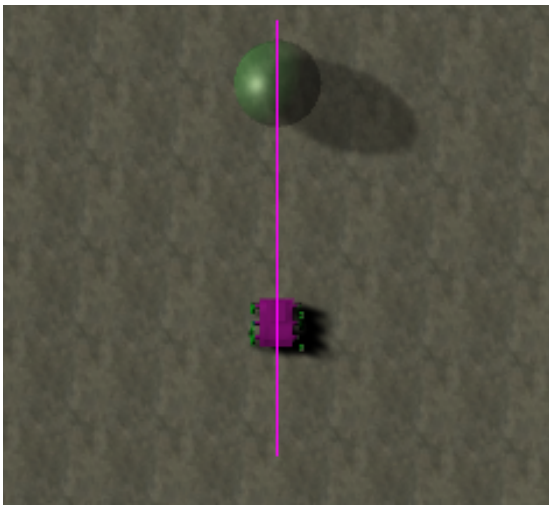
The navigation system was implemented by using the *NavMesh* system of Unity. This built-in navigation system consists out of two main components: *NavMeshes* and *NavMeshAgents*. A *NavMesh* has to be generated for the terrain geometry and when a *NavMeshAgent* (the simulated rover) is placed on this *NavMesh* then Unity can perform all the path planning in the background.

For the project it is more interesting to concentrate on the behaviour of the swarm on a higher level than implementing low-level path planning. Which is why Unity was used as a basis for the path planning.

Movement goals can be sent to the simulated rover, which will then perform checks on the goal to see whether it is valid. The assumption was made that the rover can move to a location within its field of view. Thus, a series of checks is performed to ensure that a new movement goal is within field of view and not obstructed by any obstacles. These checks are done with trigonometry and raycasts. If these checks were not added, then the *NavMeshAgent* would walk around the obstacle, which would not be representative of what the actual rover could do. Figure 13 illustrates a case where all the checks pass and the rover starts walking towards the movement goal. Figure 14 shows an example of a rejected movement goal. The movement goal is behind the red testing obstacle, thus the rover cannot see it.

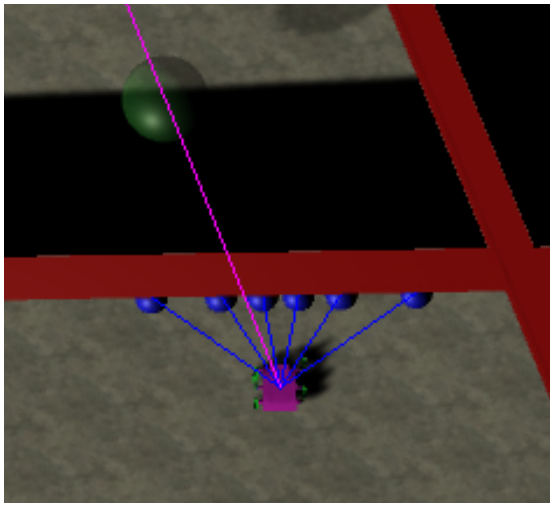


(a) The rover completed its previous movement goal and is idle.

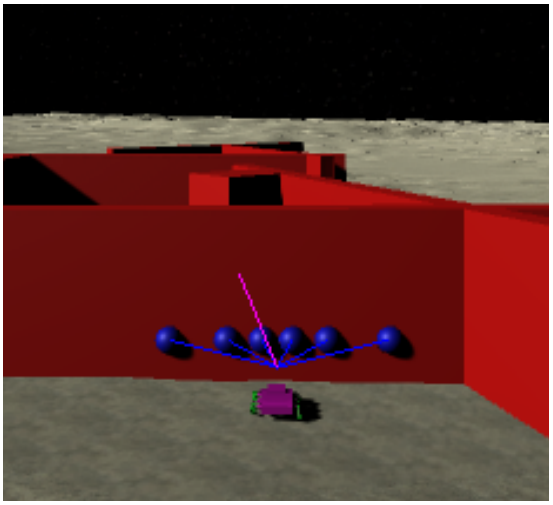


(b) The rover is walking towards a new movement goal that is up and to the right.

Figure 13: Images of the rover successfully accepting a movement goal and walking towards it.



(a) Topview of the rejected movement goal.



(b) The perspective of the rover.

Figure 14: A movement goal that is rejected because the rover cannot directly see where it should move.

Furthermore, the rover behaviour algorithm requires the ability to cancel ongoing movement goals. This is often required to stop the rover when the rover unexpectedly encounters an obstacle or another rover. Finally, it is beneficial to walk slower in situations with many obstacles and other rovers, which is why a system to change the speed of the rover was implemented.

10.2.3 Interface

ROS Topic Name	"roverX/move_action"
ROS Message Type	MoveAction
Publisher	Rover Behaviour Algorithm
Subscriber	Simulation Environment

Published by the behaviour algorithm when a rover should move to specific a location within its field of view. A MoveAction contains an X and a Z coordinate relative to the rover, to which the rover should move. Cannot be sent while another MoveAction or RotateAction is in progress.

ROS Topic Name	"roverX/rotate_action"
ROS Message Type	RotateAction
Publisher	Rover Behaviour Algorithm
Subscriber	Simulation Environment

Published by the behaviour algorithm when a rover should rotate in place. A RotateAction contains the number of degrees that the rover should rotate. A negative value indicates rotation to the left, a positive value indicates rotation to the right. Cannot be sent while another MoveAction or RotateAction is in progress.

ROS Topic Name	"roverX/navigation_status"
ROS Message Type	NavigationStatus
Publisher	Simulation Environment
Subscriber	Rover Behaviour Algorithm

Published while the rover is performing a MoveAction or a RotateAction. During a MoveAction it contains the distance in meters towards the movement goal. Similarly, during a RotateAction it contains the number of degrees that the rover must still turn to complete its RotateAction. NavigationStatus is guaranteed to send a status of zero when the action is finished.

ROS Topic Name	"roverX/cancel_action"
ROS Message Type	CancelAction
Publisher	Rover Behaviour Algorithm
Subscriber	Simulation Environment

While a rover is performing a MoveAction, it is possible to cancel this MoveAction. Which is done by publishing a CancelAction message. The message is empty.

ROS Topic Name	"roverX/cancel_result"
ROS Message Type	CancelResult
Publisher	Simulation Environment
Subscriber	Rover Behaviour Algorithm

Once a CancelAction is received, the rover slows down until its stationary. At this point the simulation environment publishes a CancelResult to indicate that the CancelAction was completed. The CancelResult contains the X and Z coordinates and the angle at which the rover has stopped. These values are calculated relative to the starting position of the MoveAction which was cancelled.

ROS Topic Name	"roverX/speed_action"
ROS Message Type	SpeedAction
Publisher	Rover Behaviour Algorithm
Subscriber	Simulation Environment

The SpeedAction allows the behaviour algorithm to adjust the movement speed of the rover. There are three different configurable speed levels. The SpeedAction contains an integer between 1 and 3 which indicates which speed level should be used.

10.3 Gyroscope Sensor

Part of the mission consists of the rover having to locate a crater and take pictures of that crater. The rover should stand on top of the edge a crater in order to take good pictures of the crater. Hence, a way to determine whether we the rover is climbing a slope is required to estimate when the rover is on the highest point of the edge of the crater.

10.3.1 Assumptions

- The actual rover is equipped with a gyroscope sensor, possibly included in an Inertial Measurement Unit (IMU).
- The actual rover can calibrate the gyroscope when on the moon.

- There exists an algorithm that processes the gyroscope output and provides usable orientation estimates with reduced noise in measurements.

10.3.2 Implementation

In the simulation a raycast is performed from the center of the simulated rover to the terrain surface beneath. The normal vector of the terrain is used to determine the direction orientation of the rover to the XZ plane.

The sensor data that is sent to the rover behaviour algorithm is simplified for easier use. The yaw and roll axes are ignored and only the pitch axis is used.

10.3.3 Interface

ROS Topic Name	"roverX/gyroscope"
ROS Message Type	Gyroscope
Publisher	Simulation Environment
Subscriber	Rover Behaviour Algorithm

The Gyroscope message contains the pitch of the simulated rover in degrees. A positive number of degrees indicates that the rover is pointing upwards. A negative number of degrees indicates that the rover is pointing downwards.

10.4 Communication

The communication system allows rovers to send messages to each other. While this is mainly used within our simulation to have a rover signal that a mission goal has been found, this can be extended to several other functionalities, such as having rovers communicate no-go zones or communicate individual rover malfunction. Apart from the rovers, the lander is also able to communicate. The lander always communicates the mission direction to all the rovers.

10.4.1 Assumptions

- Both the actual lander and all actual rovers are in possession of a working communication channel, such as an antenna or infra-red signalling.
- The communication channel has a communication radius that is at least twice as big as the size of the actual rover.
- The actual rover knows which other rovers are within communication radius by means of an underlying communication protocol.
- The actual rover can send message to rovers that are within their communication range.
- The propagation time of a message is zero. I.e. if rover A sends something to rover B then B will receive this immediately providing B is within communication range of B. This then also assumes that error detection and correction is in place and always works.
- Since in the actual rover, all communication protocols are in place, including routing, such that the rovers can send a message to another rover that is inside of the communication range of a rover that is within its communication range. I.e. communication is a transitive relation. Concretely, let $X \longrightarrow Y$ denote rover X can communicate with rover Y. Then if $X \longrightarrow Y$ and $Y \longrightarrow Z$, then $X \longrightarrow Z$.

10.4.2 Implementation

Within the simulation implementation, the simulated rover has its own communication component. Within this component, two channels are implemented: *comm_out*, where outgoing traffic and *comm_in*, where incoming traffic is published. Communication messages are sent from a rover X's outbox to a rover Y's inbox. The same message can be sent to multiple target rovers. Within the communication component, there is also a raycaster component. The raycaster checks all objects that come into the communication range of the rover. If this object is another rover, it is recorded that these two rovers have a communication link. The result of this raycasting process is published

to the *comm_partners* channel. The rover behaviour algorithm uses this to determine which rovers it can communicate with.

To keep track of the communication status among all rovers, the simulation is in possession of a communication manager (or: manager) component. This component has its own communication oracle (or: oracle). The manager can prompt this oracle to check whether there is a connection among two certain rovers. This is possible because the oracle keeps track of all individual connection by means of a graph. This graph is updated through every rover's raycaster.

To guide all the rovers, there is a channel that constantly sends a mission direction. This happens in the *MissionDirectionPublisher*. In practice, this is something that would be done through the lander, but having the environment do this suffices for the aim of our simulation.

10.4.3 Interface

ROS Topic Name	"roverX/comm_in"
ROS Message Type	InboundComm
Publisher	CommManager
Subscriber	None

The InboundComm message has an incoming message payload in string form, and also has an integer that indicates from which rover the message has been sent.

ROS Topic Name	"roverX/comm_out"
ROS Message Type	OutboundComm
Publisher	roverX
Subscriber	CommManager

The OutboundComm message has a string message payload and a list of integers to which the message should be sent to.

ROS Topic Name	"roverX/comm_partners"
ROS Message Type	CommPartners
Publisher	roverX
Subscriber	None

The raycaster of the rover publishes all rovers that are in communication range of roverX. This is done in terms of in integer array, so $[1, 2]$ Would mean that rover1 and rover2 are in range of roverX.

ROS Topic Name	"roverX/mission_direction"
ROS Message Type	MissionDirection
Publisher	roverX
Subscriber	Rover Behaviour Algorithm

The MissionDirection is used by the behavioural algorithm for calculation on where the rover should to. The two factors that are used in these calculations are the angle relative to destination in degrees and the distance relative to the destination in meters.

10.5 Observation

10.5.1 Assumptions

- The actual rover is in possession of a camera that can take pictures on command.
- The actual rover is able to rotate in place.

10.5.2 Implementation

The simulated rover has a *Camera* object on the front pointing forwards. This object is disabled by default such that it does not render to the screen. Once a *PhotoAction* is received, the interpreter of the message captures the view from this camera, with a configurable resolution, and saves the user's disk.

10.5.3 Interface

ROS Topic Name	”roverX/photo_action”
ROS Message Type	PhotoAction
Publisher	Rover Behaviour Algorithm
Subscriber	roverX

Once the rover is on the edge of a crater, it will start to rotate. Every x degrees (30 by default) it will take a photo from its camera.

10.6 User Interface

To facilitate an intuitive environment to use the simulation, a user interface (UI) has been made. When launching the simulation, the interfaces described below are shown, with a description of the functionalities.

10.6.1 Connecting to ROS

The interface indicating the ROS connector is shown in Figure 15. This screen is the first screen shown once the simulation has been launched.

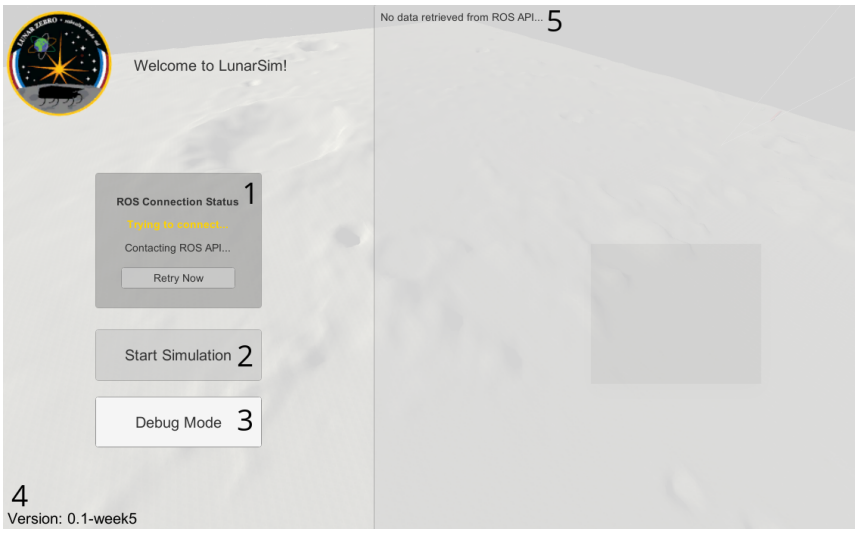


Figure 15: Image of the main connector screen. Explanation of the numbers can be found in subsubsection 10.6.1

1. **ROS Connector Status Indicator**: Shows the connection status between ROS and the simulation environment. This indicator can show three states: ”No Connection!”, ”Trying to connect...” or ”Connected!”. The connector tries to reconnect after a configurable amount of seconds when no connection has been established yet.
2. **Start Simulation Button**: Can only be pressed when a is connection established with ROS. When pressed, the loading screen (see: Figure 16) will be shown after which the scenario editor will be launched.
3. **Debug Mode Button**: Once pressed, it launches the simulation in debug mode. There is no connection with ROS needed to activate debug mode, as this is solely to check if other functionality of the environment works properly. This button effectively skips the scenario editor and will launch straight into the environment simulation. There is a rover that can be controlled in this debug mode for testing purposes.
4. **Version Indicator**: Shows the current LunarSim version, used for determining code version when troubleshooting.
5. **ROS Data Console**: All the parameters of the incoming ROS data are displayed here, once a connection is established with ROS. It displays information on ROS nodes, ROS topics and ROS parameters.

Figure 16: Image of the loading screen. The loading screen has a loading indicator and shows which process is running in the background.

10.6.2 Scenario Editor

The UI for the scenario editor can be seen in Figure 17. This editor has buttons to show several menus to guide the user in the mission design within the editor. All menus have general explanations of the entities that can be configured within their respective menu. The menus also explain controls of the editor.

1. **Lander Menu**: The lander menu provides information about positioning the lander and has a button which can be clicked to center the lander in the camera. An image of this menu can be found in Figure 18.
2. **Rovers Menu**: The Rovers Menu shows a list of all the rovers in the simulation. Next to every rover in the list, there is a button to center the certain rover in the camera. An image of this menu can be found in Figure 19.
3. **Craters Menu**: The Craters Menu shows an overview of all previously added Craters. Craters can be added in this menu by defining a center point and a crater radius. Craters can also be centered in the camera with a button. An image of this menu can be found in Figure 20.
4. **Mission Menu**: The Mission Menu consists out of a ordered series of mission goals, which can be edited, reordered or deleted within this menu. An image of this menu can be found in Figure 21.
5. **Profiles Menu**: The Profiles Menu allows the user to store and load specific scenario configurations. It also allows exporting and importing between computers. An image of this menu can be found in Figure 22.
6. **Main View**: The view of the main camera within the scenario editor. The arrow keys can be used to adjust this view and to zoom in on the simulation.
7. **Launch Button**: After showing a short loading screen that launches all rovers, this button will redirect the user to the main simulation environment. The button can only be pressed when the mission has been configured.

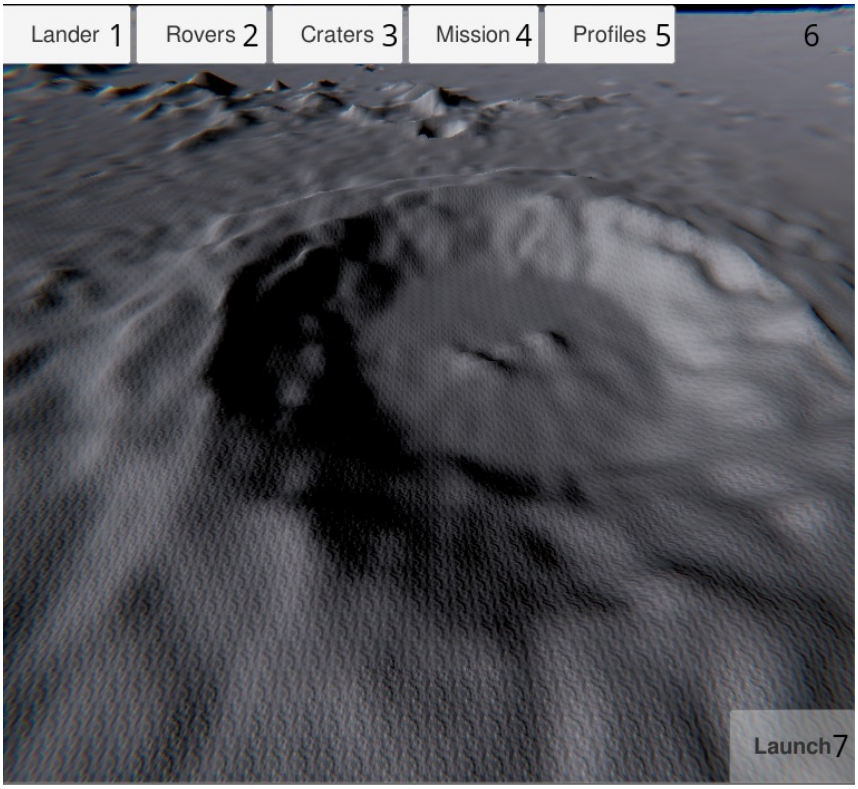


Figure 17: Image of the scenario editor screen. Explanation of the numbers can be found in subsubsection 10.6.2

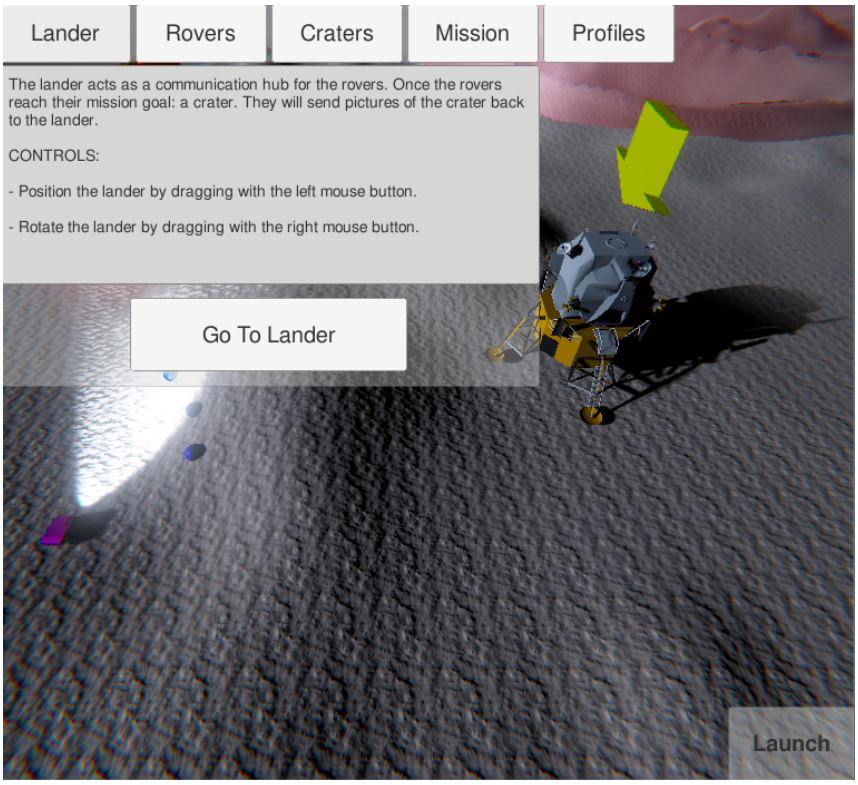


Figure 18: Image of the Lander menu in the Scenario Editor.

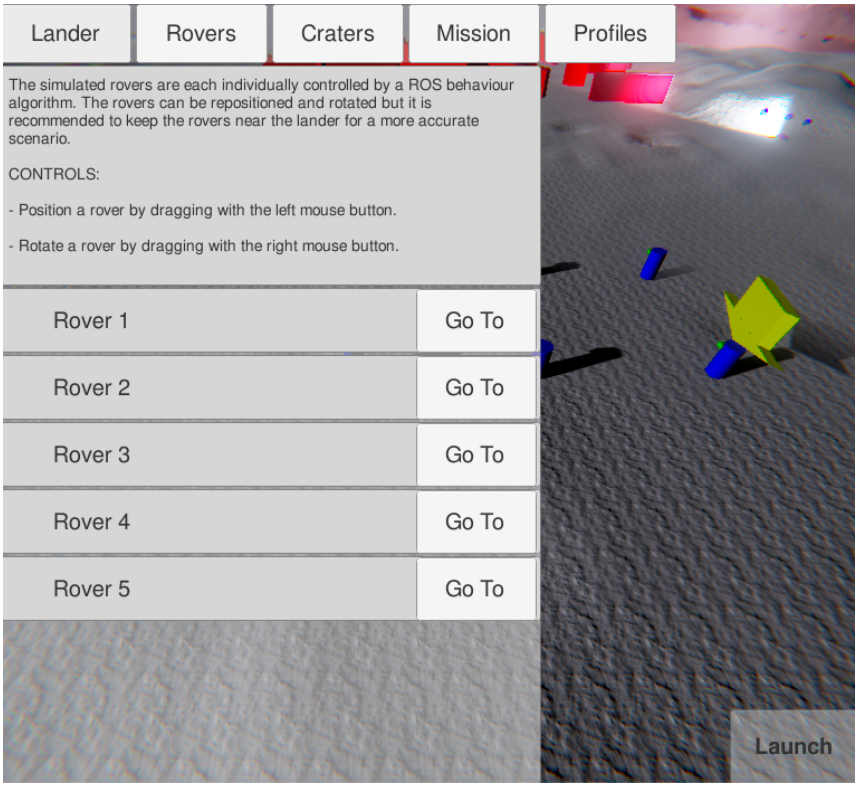


Figure 19: Image of the Rover menu in the Scenario Editor.

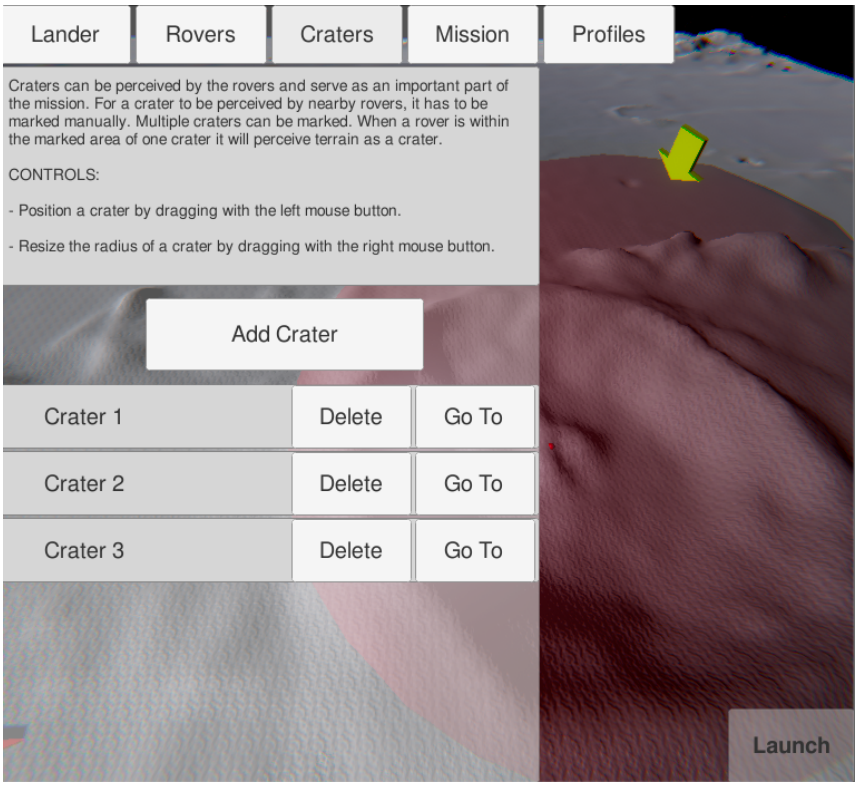


Figure 20: Image of the Crater menu in the Scenario Editor.

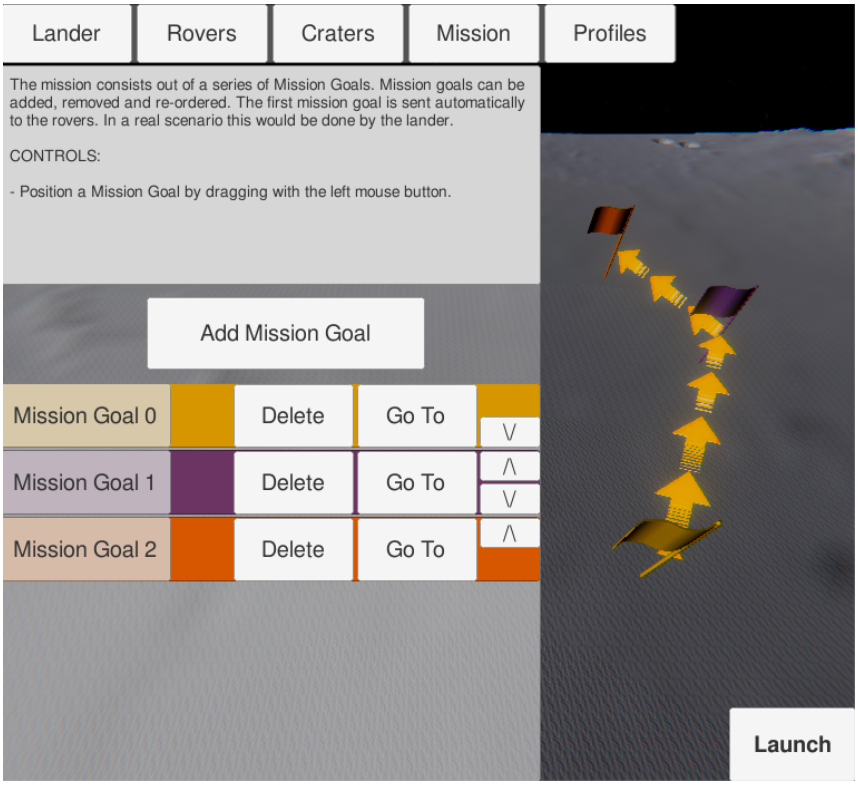


Figure 21: Image of the Mission menu in the Scenario Editor.

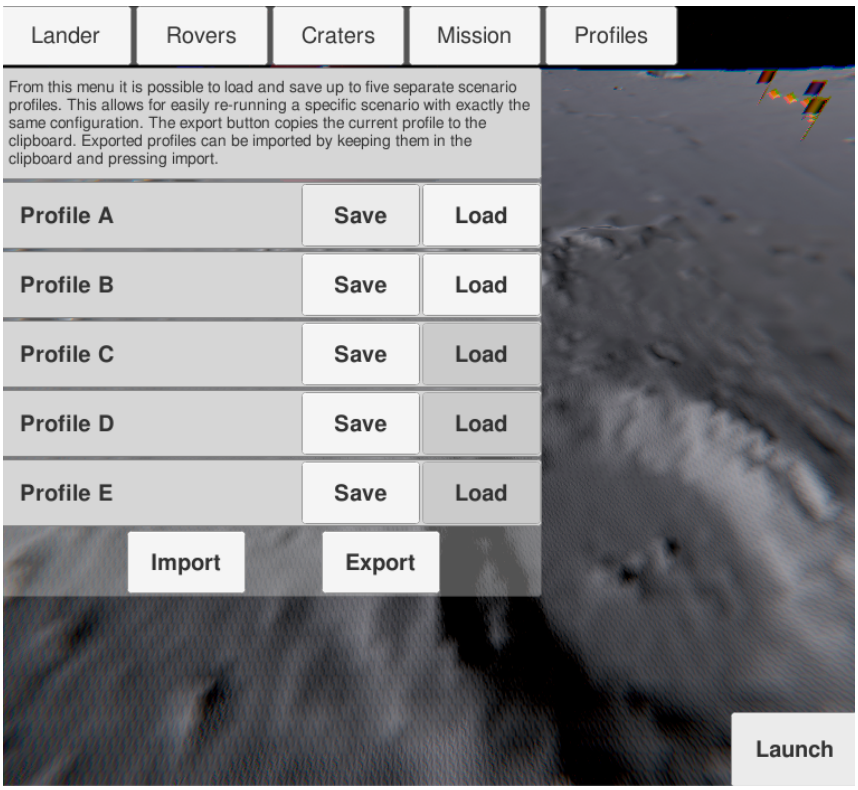


Figure 22: Image of the Profiles menu in the Scenario Editor.

10.6.3 Statistics

During the simulation a series of statistics are displayed to assess the performance of a rover behaviour algorithm on a specific scenario.

As seen in Figure 23, the *SimulationStatistics* panel shows statistics of the entire simulation. While the *RoverStatistics* panel allows the user to open a window that displays statistics of a specific rover. On the right the statistics window of *rover2* is opened.

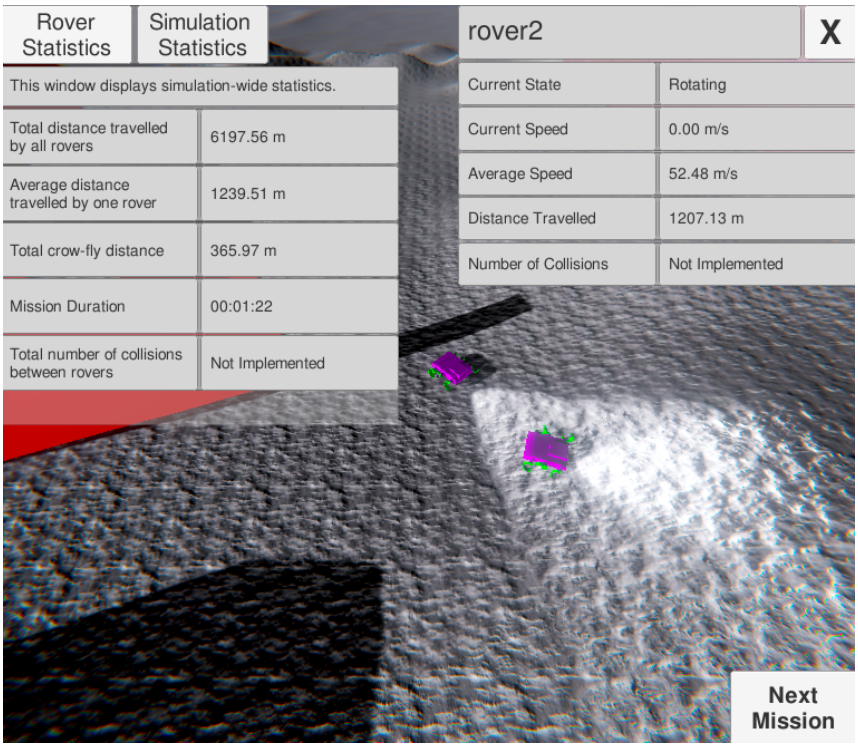


Figure 23: The interface that can be seen after launching the simulation. Statistics are displayed for the simulation and for each individual rover.

10.7 Scalability and Accuracy

It is important that the simulation is representative of the actual mission. The following subsections elaborate on what was done to increase the accuracy of the simulation. A significant challenge was to simulate the rovers and the lunar environment at a realistic scale. Additionally, to increase the chances of the simulation remaining relevant in the future, many properties of the rover are configurable.

10.7.1 Simulating large scale terrain

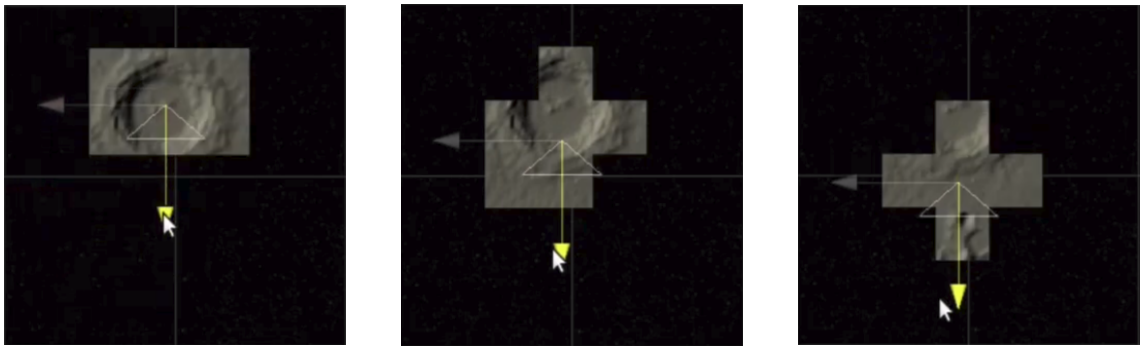
Simulating the terrain and the rovers on a realistic scale was a significant challenge during the project. The Lunar Zebro rover is about 30 centimeters long and missions could take multiple weeks in which the rovers traverse multiple kilometers of terrain. Additionally, the top speed of the Lunar Zebro rover is predicted to be one centimeter per second. This contrast in scale requires that the simulated terrain is large.

Early tests showed that Unity would crash on terrain that was larger than a $5km^2$. This was not caused by all the rendering that was required for the terrain. The bottleneck was the large *NavMesh* [4] that had to be in-memory for the navigation system.

An obvious solution would be to separate the terrain into chunks such that only the relevant chunks of terrain have to be in-memory during the simulation. However, due to Unity engine limitations, it is not possible to divide the *NavMesh* of the terrain into chunks.

[4] https://docs.unity3d.com/Manual/Navigation.html

The ability to divide and load separate chunks of the *NavMesh* during run time was required to have terrain on a realistic scale. In order to implement this, the Unity engine was extended with a customized *NavMesh* baking system that allows baking *NavMeshes* for each chunk of terrain. Figure 24 shows the terrain chunk system and how chunks are loaded based on the position of a simulated rover.



(a) The initial position of the rover.

(b) The rover is moved downwards.

(c) The rover is moved even further.

Figure 24: Top view of the terrain chunk activation system. Chunks are loaded into memory based on the position of the rover.

However, another problem was that Unity assumes that *NavMeshes* are not loaded into memory dynamically at runtime. Which leads to the problem that the rover (the *NavMeshAgent* [5]) was unable to move from one chunk of terrain to another chunk, because the *NavMeshAgent* cannot transition to another *NavMesh* that was just loaded into memory. The default *NavMeshAgent* implementation had to be extended such that it allows transitioning between dynamically loaded *NavMeshes*. The problem is illustrated in Figure 25.

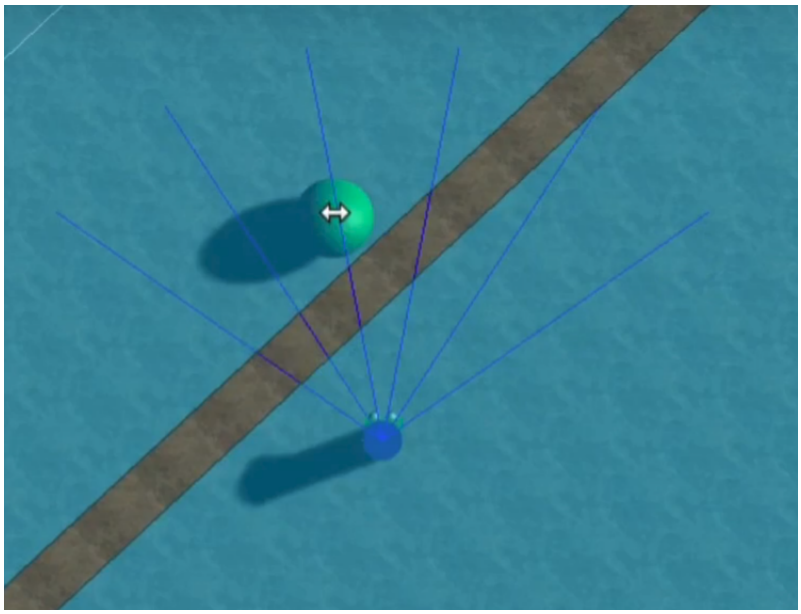


Figure 25: An image of the seam that occurs between *NavMeshes* for separate terrain chunks. While the terrain connects, the *NavMeshes* do not, due to engine limitations. The blue area visualizes the *NavMeshes*. The blue dot represents a simulated rover and the green sphere represents the location that the rover needs to navigate to.

The customized terrain system improves performance considerably and the simulation software

[5] https://docs.unity3d.com/Manual/class-NavMeshAgent.html

can now use terrains that are $50km^2$ when separated into 256 separate chunks. Furthermore, NASA provides several models of lunar environments which cover an area of up to $50km^2$. All these models can be imported and processed by the customized NavMesh baking and terrain systems. Which allows for a variety of different simulation scenarios.

10.7.2 Configuring simulation parameters

Some of the capabilities of the Lunar Zebro rover are not yet set in stone. Which is why it is important to allow for experimentation within the simulation. This is done through allowing the user to configure as much as possible about the simulated rover.

For the navigation system that was described previously, it is possible to configure maximum linear velocity, in-place angular velocity, acceleration, move action validity check strictness and more.

Similarly, it is possible to configure the discussed perception system. It is possible to adjust the perception range, the accuracy of the perception, the field of view of the simulated camera and more. An example of the effects of the parameters on the perception system can be found in Figure 26. Here the amount of perception rays and the field of view is greater than in Figure 12.

All of the configurable parameters are listed in subsection C.2.

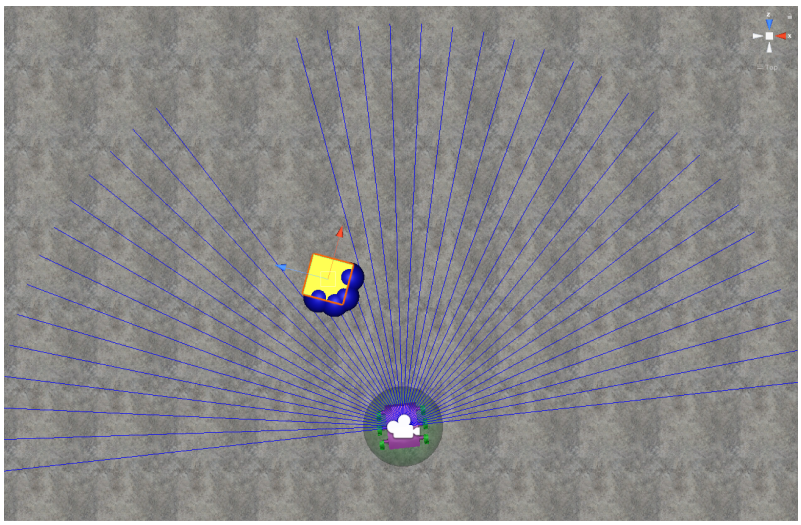


Figure 26: The perception system with differently configured parameters. The amount of perception rays has been increased which results in more accurate simulated perception but also worse performance. The field of view is set to 180 degrees.

11 Technical Specifications Rover Behaviour

[Redacted]

11.1 Single Rover Behaviour

[Redacted]

11.2 Multi-Rover Behaviour

[Redacted]

11.3 Crater Observation

[Redacted]

11.4 Daisy Chain

[Redacted]

References

[1] Lunar Zebro, "Experience curiosity." http://zebro.space/mission-2/. Online; accessed June 27, 2019.

[2] Agile Business Consortium, "Moscow prioritisation." https://www.agilebusiness.org/content/moscow-prioritisation. Online; accessed June 20, 2019.

[3] Binary Start Ltd., "Bsl - ms3 lunar." http://www.binarystarltd.com/products_ms3_lunar.html. Online; accessed June 24, 2019.

[4] Valve Corporation, "Space simulator." https://store.steampowered.com/app/529060/Space_Simulator/. Online; accessed June 24, 2019.

[5] NASA, "Mars rover game." https://mars.nasa.gov/gamee-rover/. Online; accessed June 24, 2019.

[6] NASA, "Experience curiosity." https://eyes.nasa.gov/curiosity/. Online; accessed June 24, 2019.

[7] C.-q. Yu, H.-h. Ju, and Y. Gao, "3d virtual reality simulator for planetary rover operation and testing," in *2009 IEEE International Conference on Virtual Environments, Human-Computer Interfaces and Measurements Systems*, pp. 101–106, IEEE, 2009.

[8] Schwaber and Sutherland, "The definitive guide to scrum: The rules of the game." https://www.scrumguides.org/docs/scrumguide/v2017/2017-Scrum-Guide-US.pdf#zoom=100. Online; accessed June 27, 2019.

[9] J. M. O'Kane, "A gentle introduction to ros," 2014.

[10] R. Codd-Downey, P. M. Forooshani, A. Speers, H. Wang, and M. Jenkin, "From ros to unity: Leveraging robot and virtual environment middleware for immersive teleoperation," in *2014 IEEE International Conference on Information and Automation (ICIA)*, pp. 932–936, IEEE, 2014.

[11] W. Meng, Y. Hu, J. Lin, F. Lin, and R. Teo, "Ros+ unity: An efficient high-fidelity 3d multi-uav navigation and control simulator in gps-denied environments," in *IECON 2015-41st Annual Conference of the IEEE Industrial Electronics Society*, pp. 002562–002567, IEEE, 2015.

[12] Y. Mizuchi and T. Inamura, "Cloud-based multimodal human-robot interaction simulator utilizing ros and unity frameworks," in *2017 IEEE/SICE International Symposium on System Integration (SII)*, pp. 948–955, IEEE, 2017.

[13] ESA, "ecodesign - how much do space activities pollute?." https://www.esa.int/Our_Activities/Space_Safety/Clean_Space/ecodesign. Online; accessed June 27, 2019.

[14] H. Klinkrad, "Space debris," *Encyclopedia of Aerospace Engineering*, 2010.

[15] States, "National space law collection." http://www.unoosa.org/oosa/en/ourwork/spacelaw/nationalspacelaw/index.html. Online; accessed June 27, 2019.

A Project Proposal

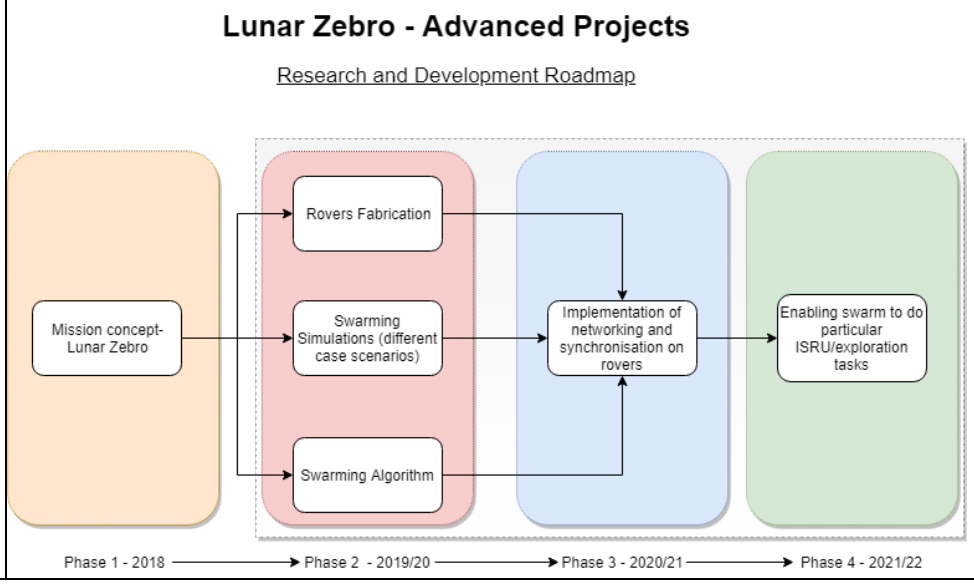
The following pages contain the original proposal of this Bachelor End Project. This is included as specified by the project requirements.

Lunar Zebro Project

BEPsys 2018-19

Researcher's Name: Raoul Bruens, Dana van Hassel, Sterre Noorthoek, Maxim Liefaard
Last Date of Update: 19/03/2019
Client: Maneesh Kr. Verma (Operations Manager, Lunar Zebro)
Responsible supervisor: ir. Dr Chris Verhoeven
Daily advisor:
Advisor: Dr. Raj Thilak Rajan

Parent Project	Lunar Zebro - Advanced Projects (aka Catalyst Section)
Subsystem	Mission Operations
Reference No.	7000.X01
Research Type	BSc. Thesis
Research period	11 Weeks
Research Theme	Simulations (Swarming)
Position	Mission Simulation Engineers
Dependent systems	Lunar Zebro, LUFAR (LUnar low Frequency ARray)
Dependent envir.	Moon
Related Engineering	Networking, ROS, Swarm intelligence, Cognitive robotics, Robot Dynamics, Learning and Autonomous Control (optional)
Input Data	Mission concept, specifications of Lunar Zebro's systems, Swarming research within Terrestrial Zebro Group
Output Data	Increase the outreach of project, organize project related events and maintain social media up to date
Comm. Protocol	n/a
Computational model	6 nodes' (rovers) locomotion, operations of payload and path planning from lander to cave, explore cave and arrive safely back to the lander.
Prime requirements	1. Excellent communication skills both orally and in writing 2. Excellent interpersonal skills 3. Experience with software quality and testing approaches 4. Programming experience: C++, Python 5. Experience with ROS 6. Affinity with interplanetary robotics and their operations 7. Excellent written and verbal English skills.
Research Output	1. Plan strategies for nodes to function as one unit for tasks 2. Study and document the mechanism of cooperation between nodes 3. Study and compare nature based vs man made techniques for swarming 4. Analyse the definition of swam intelligence to derive requirements and characteristics for nodes. 5. Advantages and disadvantages for swarming compared to traditional interplanetary rover mission to date. 6. Model swarming in ROS of nodes. 7. Analyse Self-organization 8. Define Architecture of swarm 9. Define requirements for cooperation schemes between robots and implement its behaviour in ROS.

Future works	

B Project Info Sheet

Project Title	Feasibility Study of LUFAR
Client	Stellar Space Industries for the Lunar Zebro project under supervision of Maneesh Kumar Verma
Date of the final presentation	June 4, 2019

The Lunar Zebro rover is being developed for the purposes of deploying a Lunar Low Frequency Antennas for Radio Astronomy (LUFAR) system, which can be used to perform frequency measurement on the moon. The Lunar Zebro rovers will be deployed in a group that need to be able to walk from their lander to a crater. The rovers then need to be able to observe that crater. A simulation and a behaviour algorithm for a group of simulated rover was developed to study the feasibility of such a mission.

Challenge: Developing a simulation that is representative of a moon mission and finding a behaviour strategy for moon rovers to successfully complete their simulated mission.

Research: A wide variety of aspects play a role in space missions due to the extreme conditions. The team studied similar space missions to be able to develop a more representative simulation. Additionally, swarming behaviour and multi-agent system algorithms were studied to gain an understanding of which techniques could be used for rover intelligence.

Process It was challenging in the first few weeks to gain domain knowledge on space missions. Development of the rover behaviour algorithm was more time consuming than expected and not everyone could work on the behaviour at once. Thus, instead of implementing more features in the simulation, an effort was made to make it more user-friendly and accessible.

Product A rover behaviour algorithm and a simulation environment were made for this project. The simulation environment can be used to test different scenarios.

Outlook The simulation can be used for the Lunar Zebro project to test rover behaviour algorithms in different scenarios. This can result in a better understanding of the requirements for the real Lunar Zebro rover. Furthermore, new behaviour algorithms can be developed and tested with the simulation.

B.1 Members of the project team

Name - E-mail:	Dana van Hassel - danavanhassel@hotmail.nl
Interests:	Algorithm design, AI, Robotics, Art and Design
Role & Contribution :	Lead Testing, Behaviour Algorithm Development
Name - E-mail:	Maxim Liefaard - maxim.liefaard@xs4all.nl
Interests:	Robotics, Game Development, User Interaction
Role & Contribution :	Lead Programmer, Simulation Development, UI Development
Name - E-mail:	Raoul Bruens - bruens.raoul@gmail.com
Interests:	Electrical Engineering, Social Engineering, Psychology
Role & Contribution :	Producer, Mission Design, Simulation Development
Name - E-mail:	Sterre Noorthoek - sterrechavon@hotmail.com
Interests:	Human-Computer Interaction, Information Visualisation, Automata Theory, Databases
Role & Contribution :	Chief Editor, Secretary, Behaviour Algorithm Development

B.2 Contacts

Client:	Operations Manager at Lunar Zebro, Lead Robotics at Stellar Space Industries
Maneesh Kumar Verma	operations.spacezebro@tudelft.nl
Supervisor:	Intelligent Systems - Delft Bioinformatics Lab
Dr. Thomas Abeel	T.Abeel@tudelft.nl

The final report for this project can be found at: http://repository.tudelft.nl

C Software Documentation

This appendix contains documentation about the developed software. subsection C.1 contains guides for installing, compiling and running the developed systems. subsection C.2 contains tables that explain the configurable parameters of the software in detail. subsection C.4 contains a guideline for what hardware is required to run the software.

C.1 Deployment Guide

The following two subsections contain deployment guides for the simulation codebase (LunarSimUnity) and the rover behaviour algorithm (LunarSimROS). For running both systems and connecting the behaviour algorithm both guides must be used. The order in which the guides are followed does not matter. The third subsection explains how LunarSim can be launched provided that both repositories have been installed.

C.1.1 LunarSimUnity Repository Guide

Unity simulation environment for the LunarSim software package. The LunarSimROS repository is required in combination with this repository.

LunarSimUnity makes use of the rossharp pluginhttps://github.com/siemens/ros-sharp. We have our own modified version of this plugin in the LunarSimSharp repository, which contains custom messages used for the communication between LunarSimUnity and LunarSimROS components.

Installation To work with this repository it is recommended that you have the following software installed:

- Ubuntu 16.04
- Unity Hub with Unity 2019.1.0f2
- JetBrains Rider IDE with Unity support installed

Note that the Unity project uses the ros-sharp plugin to establish a connection to ROS. This is already included in the repository to prevent compatibility issues.

To ensure that the Unity project works correctly with this git repository you need to make sure that two settings are set correctly when open the Unity project in your Unity editor for the first time:

- Edit → Project Settings → Editor → Version Control → Mode: "Visible Meta Files"
- Edit → Project Settings → Editor → Asset Serialization → Mode: "Force Text"

Never commit if you do not have this set correctly.

Open source libraries and plugins The following files were **not** custom developed but used from existing open-source projects or tools:

- All the code in the Plugins folder.
- All the code in the NavMeshComponent folders This code is from the NavMeshComponents package developed by Unity which serves as an extension to the standard NavMesh system.
- All the code in the PostProcessingStack folder. This code is from the Post Processing Stack v2 package developed by Unity which is used for visual improvements.
- The SplitTerrain script.
- The Object2Terrain script.

The rossharp plugin was **modified** for the LunarSim project. However, almost all of the code remains unchanged. All of the changes to rosssharp can be found in our LunarSimSharp repository.

All of the remaining code was custom developed for the LunarSim project.

C.1.2 LunarSimROS Repository Guide

The ROS-based lunar rover behaviour algorithm for the LunarSim software package. The LunarSimUnity repository is required in combination with this repository.
LunarSimUnity makes use of the rossharp plugin. We have our own modified version of this plugin in the LunarSimSharp repository, which contains custom messages used for the communication between LunarSimUnity and LunarSimROS components.

Installation To work with this repository it is recommended that you have the following software installed:

- Ubuntu 16.04
- ROS Kinetic with catkin
- JetBrains CLion IDE

Setup your catkin workspace according to this tutorial, we assume that you name your workspace catkin_ws.

Compiling the project

- Clone the project into catkin_ws/src/LunarSimROS.
- Install the project dependencies using: cd /catkin_ws and then rosdep install –from-paths src –ignore-src -r -y.
- You should now be able to build the project using: cd /catkin_ws and then catkin_make.
- Any issues can be caused by not having your ROS installation or workspace sourced.

Using the CLion IDE with ROS By default CLion will not do code completion for ROS code. This can be fixed by creating a custom alias to launch CLion with your ROS code sourced.

- Edit your bash_aliases file: Run cd and nano .bash_aliases.

In this file you want to add a command that varies per system, here is an example:
alias clion='source /opt/ros/kinetic/setup.bash; source $HOME/catkin_ws/devel/setup.bash; ./.local/share/JetBrains/Toolbox/apps/CLion/ch-0/182.4505.18/bin/clion.sh'

Check the following before adding your alias to the .bash_aliases file:

- source /opt/ros/kinetic/setup.bash refers to your ROS installation location (this is the default).
- source $HOME/catkin_ws/devel/setup.bash refers to the location of your catkin workspace.
- ./.local/share/JetBrains/Toolbox/apps/CLion/ch-0/182.4505.18/bin/clion.sh refers to your CLion executable, the numbers 182.4505.18 vary per system so make sure to change this.
- Once you have added the alias command to the .bash_aliases file, close all your open terminals.
- Now open a new terminal and run alias, your newly created clion alias should be listed.
- You can now correctly start CLion via terminal by running your new clion command.

Generating code testing coverage For generating code coverage we use lcov. Install this tool using: sudo apt-get install lcov.
You can now run the generateCodeCoverage.sh script in order to create a coverage report. Provided that you have google chrome installed, the coverage report will autonomously open. If not, you can find the report in catkin_ws/CodeCoverageReport and open it manually.

C.1.3 Running LunarSim

Provided that you have *LunarSimUnity* and *LunarSimROS* have been completely installed you can run the software as follows:

- Open the UnityProjectFiles folder in Unity.
- In Unity, open Scene/TestScene.
- Open a terminal and run LunarSimROS: roslaunch lunar_sim sim.launch
- Press the play button in Unity.
- You can now interact with the simulation within Unity.
- When relaunching the system: restart both Unity *and* ROS.

C.2 Parameters

The simulation environment and the rover behaviour algorithm are both highly configurable

This section describes the parameters that were used to spawn the rovers, to set the rover properties and the environment properties. These can be found in Table 3, Table 4, Table 5, Table 6. This section also describes parameters that were used in the ROS nodes. These can be found in Table 7, Table 8 and Table 9.

Parameters for spawning the rover			
Name	Type	Initial value	Description
rover_count	int	10	The amount of rovers that are spawned in the simulation.
rover_spawn_interval	float	0.1	The interval in seconds at which a rover should be spawned in Unity.
start_spawn_x	float	-510.0	The x coordinate where the rovers start spawning in Unity.
start_spawn_z	float	100.0	The z coordinate where the rovers start spawning in Unity.
spawn_spacing_x	float	10.0	The space between spawned rovers on the x axis.
spawn_spacing_z	float	10.0	The space between spawned rovers on the z axis.
spawn_group_orientation	float	90.0	The rotation of the shape in which the group spawns. (in degrees)
spawn_individual_orientation	float	180.0	The rotation of each individual rover in degrees. 0.0 is facing along the positive z-axis. 90 is facing along the positive x-axis.

Table 3: The parameters that are used for spawning the rovers

Parameters Rover Navigation			
Name	Type	Initial value	Description
in_place_ rotation_speed	float	2.0	The rotation speed of a rover while the rover is standing still. (in m/s).
move_action_ tolerance_factor	float	0.1	The allowed inaccuracy in the simulation when checking whether a move action is within perception range (in meters).
strict_line_of_sight	bool	false	When set to true, the line of sight checks in the simulation are less strict and less accurate.
visualize_navigation	bool	true	When set to true, the navigation debug rays are visualized in Unity.
visualize_collision	bool	true	When set to true, collisions are visuallised in Unity.
speed_mode_one	float	1.5	Very slow speed mode: Linear speed in m/s of the first mode of the rover that can be set with the speed action.
speed_mode_two	float	2.5	Slow speed mode: Linear speed in m/s of the second mode of the rover that can be set with the speed action.
speed_mode_three	float	3.5	Normal speed mode:Linear speed in m/s of the third mode of the rover that can be set with the speed action.
angular_speed	float	120.0	Angular speed in m/s while the rover is moving. (note: in-place angular speed is specified by in_place_rotation_speed)
acceleration	float	20.0	Acceleration of the rovers move in m/s^2 this is both linear and angular acceleration.
publish_frequency	int	5	The amount of times the "navigation_status" should be published by Unity per second.

Table 4: The parameters that are used for the rover navigation

Parameters Rover Perception			
Name	Type	Initial value	Description
field_of_view_degrees	float	210.0	The field of view of the camera of the rover (in degrees).
perception_radius	float	10.0	The range of the vision of the rover (in meters).
perception_granularity	int	50	The accuracy of the vision, a higher value results in more accuracy (more raycasts).
visualize_perception	bool	false	When set to true, the perception raycasts are visualized in Unity.
publish_frequency	int	5	The number of times perception should be published by Unity per second.
gyroscope_frequency	int	20	The number of times gyroscope data should be published by Unity per second.
Parameters Rover Communication			
comm_range	float	10.0	The range in which rovers can communicate when there are no obstacles in the way.
visualize_comm	bool	true	When set to true, the communication raycasts between rovers are visualized in Unity.

Table 5: The parameters that are used for the rover perception and communication

Parameters Camera			
Name	Type	Initial value	Description
near_clip_plane	float	5.0	The distance from the camera on which rendering starts in meters.
far_clip_plane	float	7000.0	The distance from the camera where the rendering stops in meters.
move_speed	float	8.0	Move speed of the FreeCamera in m/s.
zoom_speed	float	48.0	Zoom speed of the FreeCamera in m/s.
rotate_speed	float	2.0	Rotate speed of the FreeCamera in m/s.
rotate_sharpness	float	12.0	The turning angle of the rotation (higher is a sharper angle)
move_speed_mul	float	8.0	Move speed multiplier of the FreeCamera when the shift key is held.
zoom_speed_mul	float	8.0	Zoom speed multiplier of the FreeCamera when the shift key is held.
rotate_speed_mul	float	4.0	Rotate speed multiplier of the FreeCamera when the shift key is held.
Parameters Terrain			
rover_chunk_ activation_range	float	1000.0	The distance in meters from a rover to a chunk when the chunk should be activated.
rover_chunk_nav _activation_range	float	20.0	The distance in meters from a rover to a chunk when the NavMesh of the chunk should be activated.
cam_chunk_ activation_range	float	1000.0	The distance in meters from the camera to a chunk when the chunk should be activated.

Table 6: The parameters that are used for the rovers camera and the terrain

Parameters for ROS class path_planner			
Name	Type	Initial value	Description
destination_ direction	float	0.0	The direction of the destination. (The current mission goal diretcion)
new_x	float	0.0	The x coordinate of the next move.
new_z	float	0.0	The z coordinate of the next move.
new_rotate_ angle	float	0.0	The rotation angle in which the rover will rotate in the next action.
rover_radius	float	0.5	The physical radius of the rover.
max_wait_time	float	0.5	The max amount of time a rover should wait for other rovers. The rovers waits when other rovers are close and it published a cancel action.
collision_ danger_dist	float	4.0	The extra distance rovers keep from each other to prevent collisions.
gyroscope	float	0.0	The gyroscope measures the vertical angle of the rover.
distance_to_ move_action	float	10	The distance between the current position and a move action. When a rover rotates, the distance is given in rotation degrees.
continue_migration	bool	true	When set to true the rover will migrate. When set to false it will stop moving.
ready_for_next_ move_action	bool	true	When set to true the rover can calculate a new move action.
received_cancel_ result	bool	false	When set to true the rover received a message on the topic "cancel_result".
ready_to_publish_ cancel_action	bool	true	When set to true the rover can publish a cancel result.
crater_in_vision	bool	false	When set to true the rover has seen a crater.

Table 7: The parameters that are in ROS for the class PathPlanner

Parameters for ROS class Strategy			
Name	Type	Initial value	Description
distance_relative_to_path	float	0.0	The distance between the new XZ coordinate and Path. (perpendicular to Path)
distance_walked	float	0.0	The total distance walked on Path.
angle_relative_to_path	float	0.0	The relative angle of the rover (in radian). Compared with Path.
prev_dist_relative_to_path	float	0.0	The previous distance between the new XZ coordinate and Path.
prev_distance_walked	float	0.0	The previous distance the rovers walked on Path.
prev_angle_relative_to_path	float	0.0	The previous relative angle of the rover (in radian). Compared with Path.
rotate_action	float	60.0	The angle in which a rover will rotate (in degrees).
turn_step	float	4.0	The size of the rurn step (in meters)
path_area	float	2.0	Area around a path that rovers percepts as being on the Path.
scan_angle	float	10.0	The size between the scan rays
scanner_field_of_view	float	160	The field of view in which the rover scans and calculates distances.
extra_distance_between_rovers	float	2.0	The extra distance that should be kept between rovers. (roverArea * extraDistanceBetweenRovers)
min_move_size	float	2.0	The minimum size that a move should be.

Table 8: The parameters that are in ROS for the class Strategy

Parameters for ROS class mission_planner, rover_controller and communication_controller			
Name	Type	Initial value	Description
mission_planner			
destination_is_reached	bool	false	When this is true, the destination is reached.
rover_controller			
state	string	”migration”	The current state of the rover is migration.
communication_controller			
available_for_new_message	float	true	When this is true, the CommunicationController is handling a message. (The rover can only handle one at a time)
message_to_environment_received	float	false	When this is true, a message was correctly received by the rover in unity.

Table 9: The parameters that are in ROS for the classes MissionPlanning, RoverController and CommunicationController.

C.3 Profiles

The following subsections contain several profile encodings that can be imported into the Scenario Editor. These profiles were used for various tests during the project.

C.3.1 Result Gathering Scenarios

Used for running all the simulations in the results section.


```
{"landerDataObject":{"position":{"x":-560.0,"y":36.78572082519531,"z
":50.0},"rotation":{"x":0.0,"y":0.0,"z":0.0,"w":1.0}},"
roverDataObjects":[{"position":{"x":-510.0,"y":38.4853401184082,"z
":100.0},"rotation":{"x":0.0,"y":0.9999997615814209,"z":0.0,"w
":-0.0006437301635742188}},{"position":{"x":-500.0,"y
":38.49080276489258,"z":100.0},"rotation":{"x":0.0,"y
":0.9999997615814209,"z":0.0,"w":-0.0006437301635742188}},{"
position":{"x":-490.0,"y":38.71527099609375,"z":100.0},"rotation
":{"x":0.0,"y":0.9999997615814209,"z":0.0,"w
":-0.0006437301635742188}},{"position":{"x":-510.0,"y
":38.099788665771487,"z":90.0},"rotation":{"x":0.0,"y
":0.9999997615814209,"z":0.0,"w":-0.0006437301635742188}},{"
position":{"x":-500.0,"y":38.125160217285159,"z":90.0},"rotation
":{"x":0.0,"y":0.9999997615814209,"z":0.0,"w
":-0.0006437301635742188}}],"craterDataObjects":[{"position":{"x
":-590.417236328125,"y":31.903087615966798,"z
":-20.685699462890626},"radius":38.11933517456055}],"
missionDataObjects":[{"position":{"x":-592.12255859375,"y
":31.221939086914064,"z":-19.00347900390625},"index":0}]}
```


Figure 27: A scenario where a crater is observed.


```
{"landerDataObject":{"position":{"x":-580.6666259765625,"y
":29.818368911743165,"z":-153.0},"rotation":{"x":0.0,"y":0.0,"z
":0.0,"w":1.0}},"roverDataObjects":[{"position":{"x":-585.0,"y
":31.99980926513672,"z":-103.0},"rotation":{"x":0.0,"y":1.0,"z
":0.0,"w":-0.00027817487716674805}},{"position":{"x":-575.0,"y
":31.91216468811035,"z":-103.0},"rotation":{"x":0.0,"y":1.0,"z
":0.0,"w":-0.00027817487716674805}},{"position":{"x":-565.0,"y
":32.094356536865237,"z":-103.0},"rotation":{"x":0.0,"y":1.0,"z
":0.0,"w":-0.00027817487716674805}},{"position":{"x":-585.0,"y
":31.502389907836915,"z":-113.0},"rotation":{"x":0.0,"y":1.0,"z
":0.0,"w":-0.00027817487716674805}},{"position":{"x":-575.0,"y
":31.551210403442384,"z":-113.0},"rotation":{"x":0.0,"y":1.0,"z
":0.0,"w":-0.00027817487716674805}}],"craterDataObjects":[{"
position":{"x":-590.9841918945313,"y":31.449506759643556,"z
":-18.162841796875},"radius":32.778839111328128}],"
missionDataObjects":[{"position":{"x":-589.5396118164063,"y
":30.432601928710939,"z":-17.880584716796876},"index":0},{"position
":{"x":-585.3287353515625,"y":30.43699073791504,"z
":-111.85443115234375},"index":1}]}
```


Figure 28: A scenario where the rovers need to walk to a crater and return to the lander.

```
{"landerDataObject":{"position":{"x":-560.0,"y":36.78572082519531,"z
":50.0},"rotation":{"x":0.0,"y":0.0,"z":0.0,"w":1.0}},"
roverDataObjects":[{"position":{"x":-510.0,"y":38.4853401184082,"z
":100.0},"rotation":{"x":0.0,"y":1.0000001192092896,"z":0.0,"w
":-0.00009223818778991699}},{"position":{"x":-500.0,"y
":38.49080276489258,"z":100.0},"rotation":{"x":0.0,"y
":1.0000001192092896,"z":0.0,"w":-0.00009223818778991699}},{"
position":{"x":-490.0,"y":38.71527099609375,"z":100.0},"rotation
":{"x":0.0,"y":1.0000001192092896,"z":0.0,"w
":-0.00009223818778991699}},{"position":{"x":-510.0,"y
":38.099788665771487,"z":90.0},"rotation":{"x":0.0,"y
":1.0000001192092896,"z":0.0,"w":-0.00009223818778991699}},{"
position":{"x":-500.0,"y":38.125160217285159,"z":90.0},"rotation
":{"x":0.0,"y":1.0000001192092896,"z":0.0,"w
":-0.00009223818778991699}}],"craterDataObjects":[{"position":{"x
":-588.184326171875,"y":32.183494567871097,"z":-17.80078125},"
radius":36.85749435424805}],"missionDataObjects":[{"position":{"x
":-507.575439453125,"y":32.57106018066406,"z
":-52.761932373046878},"index":0}]}
```


Figure 29: A scenario to stress test path finding by walking through a maze.


```
{"landerDataObject":{"position":{"x":-560.0,"y":36.78572082519531,"z
":50.0},"rotation":{"x":0.0,"y":0.0,"z":0.0,"w":1.0}},"
roverDataObjects":[{"position":{"x":-510.0,"y":38.4853401184082,"z
":100.0},"rotation":{"x":0.0,"y":0.9999998807907105,"z":0.0,"w
":-0.0006398856639862061}},{"position":{"x":-500.0,"y
":38.49080276489258,"z":100.0},"rotation":{"x":0.0,"y
":0.9999998807907105,"z":0.0,"w":-0.0006398856639862061}},{"
position":{"x":-490.0,"y":38.71527099609375,"z":100.0},"rotation
":{"x":0.0,"y":0.9999998807907105,"z":0.0,"w
":-0.0006398856639862061}},{"position":{"x":-510.0,"y
":38.099788665771487,"z":90.0},"rotation":{"x":0.0,"y
":0.9999998807907105,"z":0.0,"w":-0.0006398856639862061}},{"
position":{"x":-500.0,"y":38.125160217285159,"z":90.0},"rotation
":{"x":0.0,"y":0.9999998807907105,"z":0.0,"w
":-0.0006398856639862061}}],"craterDataObjects":[{"position":{"x
":-590.5499267578125,"y":30.956329345703126,"z
":-15.790863037109375},"radius":36.22389221191406}],"
missionDataObjects":[{"position":{"x":-474.1957702636719,"y
":33.535797119140628,"z":-20.077667236328126},"index":0},{"position
":{"x":-590.9535522460938,"y":30.065885543823243,"z
":-17.18743896484375},"index":1},{"position":{"x
":-506.13714599609377,"y":37.194740295410159,"z
":96.21493530273438},"index":2}]}
```


Figure 30: A scenario where the rovers move through the obstacle maze, go to the crater and back to the lander.

C.3.2 Live Demo Profile

Used during the live demo.

```
{"landerDataObject":{"position":{"x":-610.0,"y":36.63045883178711,"z":55.0},"rotation":{"x":0.0,"y":0.0,"z":0.0,"w":1.0}},"roverDataObjects":[{"position":{"x":-600.0,"y":37.91720962524414,"z":60.0},"rotation":{"x":0.0,"y":1.0,"z":0.0,"w":-0.0005321204662322998}},{"position":{"x":-590.0,"y":37.956703186035159,"z":60.0},"rotation":{"x":0.0,"y":1.0,"z":0.0,"w":-0.0005321204662322998}},{"position":{"x":-580.0,"y":38.19357681274414,"z":60.0},"rotation":{"x":0.0,"y":1.0,"z":0.0,"w":-0.0005321204662322998}},{"position":{"x":-600.0,"y":37.6497688293457,"z":50.0},"rotation":{"x":0.0,"y":1.0,"z":0.0,"w":-0.0005321204662322998}},{"position":{"x":-590.0,"y":37.71330642700195,"z":50.0},"rotation":{"x":0.0,"y":1.0,"z":0.0,"w":-0.0005321204662322998}}],"craterDataObjects":[{"position":{"x":-588.184326171875,"y":32.183494567871097,"z":-17.80078125},"radius":36.85749435424805}],"missionDataObjects":[{"position":{"x":-506.1687927246094,"y":35.87458038330078,"z":57.16644287109375},"index":0},{"position":{"x":-494.3261413574219,"y":32.847129821777347,"z":-39.518524169921878},"index":1},{"position":{"x":-590.9503784179688,"y":30.931808471679689,"z":-19.68597412109375},"index":2}]}
```


Figure 31: The scenario used during the live demo.

C.3.3 Live Demo Import Example Profile

Used to show how the data can be imported into the simulation during the live demo.


```
{"landerDataObject":{"position":{"x":-556.4857177734375,"y
    ":36.83494567871094,"z":25.931854248046876},"rotation":{"x":0.0,"y
    ":0.0,"z":0.0,"w":1.0}},"roverDataObjects":[{"position":{"x
    ":-501.35821533203127,"y":35.50514602661133,"z
    ":11.969467163085938},"rotation":{"x":0.0,"y":0.9999997615814209,"z
    ":0.0,"w":-0.000645756721496582}},{"position":{"x
    ":-468.6780700683594,"y":34.241432189941409,"z
    ":-40.325469970703128},"rotation":{"x":0.0,"y":0.9999997615814209,"
    z":0.0,"w":-0.000645756721496582}},{"position":{"x
    ":-507.83966064453127,"y":35.748844146728519,"z
    ":12.689933776855469},"rotation":{"x":0.0,"y":0.9999997615814209,"z
    ":0.0,"w":-0.000645756721496582}},{"position":{"x
    ":-502.6518859863281,"y":35.63990783691406,"z
    ":16.346275329589845},"rotation":{"x":0.0,"y":0.9999997615814209,"z
    ":0.0,"w":-0.000645756721496582}},{"position":{"x
    ":-544.1698608398438,"y":35.708351135253909,"z":-37.4091796875},"
    rotation":{"x":0.0,"y":0.9999997615814209,"z":0.0,"w
    ":-0.000645756721496582}}],"craterDataObjects":[{"position":{"x
    ":-469.9295654296875,"y":34.036190032958987,"z
    ":-42.13372802734375},"radius":32.52561950683594},{"position":{"x
    ":-542.6070556640625,"y":35.30908966064453,"z
    ":-37.585357666015628},"radius":31.41042137145996}],"
    missionDataObjects":[{"position":{"x":-538.7380981445313,"y
    ":38.20017623901367,"z":45.19073486328125},"index":0},{"position
    ":{"x":-510.7135925292969,"y":35.84369659423828,"z
    ":54.0863037109375},"index":1},{"position":{"x
    ":-475.9683837890625,"y":35.82939147949219,"z
    ":51.279022216796878},"index":2},{"position":{"x
    ":-457.3841552734375,"y":34.97935485839844,"z
    ":28.030914306640626},"index":3}]}
```


Figure 32: The scenario used during the live demo to showcase the import functionality.

C.4 Hardware Specifications

Table 10 contains the minimal and recommended hardware specifications necessary to run the simulation. These recommendations are based on personal experiences within the team.

Processor	Graphics card	Memory	SSD	
Intel Core i7-4710MQ	Quadro K1100M	8GB	no	Minimum Hardware Specifications
Intel Core i7-6700HQ	GeForce GTX 960M	16GB	yes	Recommended Hardware Specifications

Table 10: Minimum and reccomended hardware specifications

D SIG Feedback

As part of the project the Software Improvement Group (SIG) provided feedback on the codebase. On June 1, 2019 the code was sent to SIG for a first review. On June 11, 2019 SIG sent feedback on the codebase. Based on this feedback, improvements to the code were made and the improved code was sent to SIG again on June 22, 2019 for a final round of feedback.

Note that all of the feedback received from SIG was in dutch.

Furthermore, due to concerns from the client about the trustworthiness of SIG the team was allowed to only send a third of the codebase. Specifically, the team was not allowed to send all of the simulation environment code and all of the code connecting the simulation environment to the behaviour algorithm. Thus, SIG only ever reviewed the ROS rover behaviour algorithm.

D.1 Initial SIG feedback

First some information was sent about how to interpret the feedback.

> Hierbij ontvang je onze evaluatie van de door jou opgestuurde code. De evaluatie bevat een aantal aanbevelingen die meegenomen kunnen worden tijdens het vervolg van het project. Bij de evaluatie van de tweede upload kijken we in hoeverre de onderhoudbaarheid is verbeterd, en of de feedback is geaddresseerd. Deze evaluatie heeft als doel om studenten bewuster te maken van de onderhoudbaarheid van hun code, en dient niet gebruikt te worden voor andere doeleinden.
> Let tijdens het bekijken van de feedback op het volgende:
>
> - Het is belangrijk om de feedback in de context van de huidige onderhoudbaarheid te zien. Als een project al relatief hoog scoort zijn de genoemde punten niet 'fout', maar aankopingspunten om een nog hogere score te behalen. In veel gevallen zullen dit marginale verbeteringen zijn, grote verbeteringen zijn immers moeilijk te behalen als de code al goed onderhoudbaar is.
> - Voor de herkenning van testcode maken we gebruik van geautomatiseerde detectie. Dit werkt voor de gangbare technologieën en frameworks, maar het zou kunnen dat we jullie testcode hebben gemist. Laat het in dat geval vooral weten, maar we vragen hier ook om begrip dat het voor ons niet te doen is om voor elk groepje handmatig te kijken of er nog ergens testcode zit.
> - Hetzelfde geldt voor libraries: als er voldaan wordt aan gangbare conventies worden die automatisch niet meegenomen tijdens de analyse, maar ook hier is het mogelijk dat we iets gemist hebben.
>
> Mochten er nog vragen of opmerkingen zijn dan horen we dat graag.

Then the actual feedback specific to the codebase was sent.

> De code van het systeem scoort 3.4 sterren op ons onderhoudbaarheidsmodel, wat betekent dat de code marktgemiddeld onderhoudbaar is. We zien Unit Complexity en Unit Size vanwege de lagere deelscores als mogelijke verbeterpunten.
> Voor Unit Complexity wordt er gekeken naar het percentage code dat bovengemiddeld complex is. Dit betekent overigens niet noodzakelijkerwijs dat de functionaliteit zelf complex is: vaak ontstaat dit soort complexiteit per ongeluk omdat de methode te veel verantwoordelijkheden bevat, of doordat de implementatie van de logica onnodig complex is. Het opsplitsen van dit soort methodes in kleinere stukken zorgt ervoor dat elk onderdeel makkelijker te begrijpen, makkelijker te testen is, en daardoor eenvoudiger te onderhouden wordt. Door elk van de functionaliteiten onder te brengen in een aparte methode met een beschrijvende naam kan elk van de onderdelen apart getest worden, en wordt de overall flow van de methode makkelijker te begrijpen. Bij grote en complexe methodes kan dit gedaan worden door het probleem dat in de methode wordtd opgelost in deelproblemen te splitsen, en elk deelprobleem in een eigen methode onder te brengen. De oorspronkelijke methode kan vervolgens deze nieuwe methodes aanroepen, en de uitkomsten combineren tot het uiteindelijke resultaat.
> Voorbeelden in jullie project:
>
> - SimulationLauncher::checkParams(NodeHandle)
> - Strategy::isXZReachable(float)
>
> Bij Unit Size wordt er gekeken naar het percentage code dat bovengemiddeld lang is. Dit kan verschillende redenen hebben, maar de meest voorkomende is dat een methode te veel functionaliteit bevat. Vaak was de methode oorspronkelijk kleiner, maar is deze in de loop van tijd steeds verder uitgebreid. De aanwezigheid van commentaar die stukken code van elkaar scheiden is meestal een indicator dat de methode meerdere verantwoordelijkheden bevat. Het opsplitsen van dit soort methodes zorgt er voor dat elke methode een duidelijke en specifieke functionele scope heeft. Daarnaast wordt de functionaliteit op deze manier vanzelf gedocumenteerd via methodenamen.
> Voorbeelden in jullie project:
>
> - PathPlanner::launch(int, char*)
> - Strategy::calculateNewXZ()
> - CommunicationController::launch(int, char*)
>
> De aanwezigheid van testcode is in ieder geval veelbelovend. De hoeveelheid tests blijft nog wel wat achter bij de hoeveelheid productiecode, hopelijk lukt het nog om dat tijdens het vervolg van het project te laten stijgen. Op lange termijn maakt de aanwezigheid van unit tests je code flexibeler, omdat aanpassingen kunnen worden doorgevoerd zonder de stabiliteit in gevaar te brengen.
> Over het algemeen is er dus nog wat verbetering mogelijk, hopelijk lukt het om dit tijdens de rest van de ontwikkelfase te realiseren.

D.2 Improvements based on SIG feedback

Based on the feedback an effort was made to split methods into smaller units. Furthermore, more testing code was written which eventually resulted in a line coverage of 100%. Note that code related to ROS publishers and subscribers was excluded from the coverage because this was too complex to test within the scope of the project, this is also described in subsection 7.3.

Developing more tests lead to more insight into what code was easily testable. This also resulted in better production code because methods were split even further to reduce unit size and to increase testability. To further improve testability all ROS API related code was also separated from rover behaviour logic.

Finally, a simplified state machine was implemented to make the entire program flow more easy to understand and work with.

D.3 Final SIG feedback

> Hierbij ontvang je de resultaten van de hermeting van de door jou opgestuurde code. In de hermeting hebben we met name gekeken naar of/hoe de aanbevelingen van de vorige evaluatie zijn geimplementeerd. Ook deze hermeting heeft het doel om studenten bewuster te maken van de onderhoudbaarheid van hun code en dient niet gebruikt te worden voor andere doeleinden.
>
> In de tweede upload zien we dat het codevolume is gegroeid, terwijl de score voor onderhoudbaarheid is gestegen.
>
> We zien dat de verbeterpunten uit de feedback op de eerste upload zijn aangepast, en op deze gebieden is dan ook een verbetering in de deelscores te zien. Jullie hebben op allebei de genoemde metrieken een structurele en grote sprong voorwaarts gemaakt, ook in de nieuwe code.
>
> Ook is het goed om te zien dat er naast nieuwe productiecode ook nieuwe testcode is geschreven.
>
> Uit deze observaties kunnen we concluderen dat de aanbevelingen van de vorige evaluatie zijn meegenomen in het ontwikkeltraject.

E Literature Study

The following pages contain the Research Report that was written at the start of the project. It contains the results of a Literature Study that lasted two weeks. During this literature study, the team learned about space mission planning, the conditions on the moon, the capabilities of the Lunar Zebro rover and the existing research that was already done for the project.

Feasibility Study of LUFAR
Research Report

Dana van Hassel, Maxim Liefaard, Raoul Bruens, and Sterre Noorthoek

Supervisor: Dr. Thomas Abeel
Client: Maneesh Kumar Verma

May 6, 2019

Contents

1 Introduction

Measurements of cosmic noise and the unexplored lower frequency ranges remain difficult from Earth. Using LOFAR, the Low Frequency Array in the Netherlands, it became possible to perform measurements on frequencies as low as 30 MHz [1]. However, it remains impossible to perform accurate measurements below 30 MHz due to the atmosphere of the Earth and the noise on Earth. A solution to this would be to perform low frequency measurements on the far side of the Moon. On the far side of the Moon there is less noise, which allows for more accurate measurements.

The company Stellar Space Industries aims to realise a concept called Lunar Low Frequency Antennas for Radio Astronomy (LUFAR) [2]. For LUFAR, a group of small rovers need to be deployed on the surface of the Moon. Lunar Zebro is the name of the project that develops and deploys these rovers. The rovers then need to travel to a crater on the far side of the Moon where the rovers will perform measurements. These measurements can then be combined using a technique called interferometry into a single frequency measurement.

Once deployed on the Moon, the group of rovers need to explore the Moon and find a specific crater. The Lunar Zebro team is developing the rovers such that they are relatively inexpensive, thus the rovers can co-operate in larger groups with the capability of performing measurements from more locations simultaneously and covering more ground quickly. Since these rovers need to co-operate, Stellar Space Industries is especially interested in the possibilities of swarm behaviour for the Lunar Zebro project.

A mission simulation will be developed in order to research the feasibility of LUFAR and to study the possibilities of collective behaviour for the rovers. The software will simulate an environment of a specific location on the Moon in which rovers can navigate and perform actions relevant to the mission. On top of that, collective behaviour will be developed for the rovers and tested within the simulation.

This report describes the preliminary research that was done to aid in the development process of the simulation software. First, section 2 explains the various stakeholder needs. From these stakeholder needs, requirements are formulated in section 3. Then, section 4 defines the mission that is to be simulated and all of its phases. Next, section 5 describes which aspects of the mission will be simulated to what extend including software requirements. Finally, section 6 focuses on the development of strategies that will be used to design the algorithm that manages the rovers' behaviour.

2 Stakeholders

This section describes the needs of the various stakeholders involved in the project. Based on these needs the team researched relevant topics and constructed a series of requirements.

2.1 Mission Needs

Stellar Space Industries defined the overall mission goal as: *To design, develop, and demonstrate a lunar rover that can walk on a lunar surface and communicate with Earth.* The following needs for the Lunar Zebro project simulate rovers in terms of goals, capabilities, and behaviour:

- N1 A group of rovers need to navigate to the edge of a specific crater. The exact location of the crater is unknown, only the general direction towards the crater from the lander is known.
- N2 Once the rovers reach the crater, they need to explore the area around the crater.
- N3 The rovers need to always maintain communication with the lander such that they can send images or topological information while they are exploring.
- N4 The rovers need to be able to perform most phases of the mission autonomously. Efficient navigation within the swarm of robots must be autonomous.
- N5 The swarm must be able to adapt to change of the targeted crater during the mission at all times.
- N6 It should be possible to stop specific rovers such that they stop or return to the lander.
- N7 The rovers should be able to navigate in a configurable formation.
- N8 The simulated rovers could be configurable in terms of features.

Next, the client defined a series of needs with respect to the environment in which the rovers are simulated:

- N9 The rovers should be affected by a simulated temperature in the simulation environment and the rovers turn off if their temperature becomes too low.
- N10 The rovers have a limited battery life that can be recharged by means of their solar panels.
- N11 The rovers can communicate with one another when they are within line of sight in their environment.

2.2 Systems Engineering

In cooperation with members from the Lunar Zebro team systems engineering is applied to help define the functionality of the rover that is to be simulated. First, a discovery tree is made to uncover all the different concepts involved in the rover simulation. The discovery tree can be found in Figure 1. Next, a functional tree is made to define what the simulated rover must be able to do in Figure 2.

2.3 Other organisations

As the project continues the rovers are developed and eventually sent to the Moon, more stakeholders will become involved.

- The International Telecommunication Union (ITU) will become involved when determining the frequency ranges used for communication between rovers the lander and Earth.
- The company that eventually provides the lander will have requirement for the specifications and functionality of the rovers.
- The company that launches the lander will have requirements for the weight of the lander and thus the rovers.
- Several space agencies like European Space Agency (ESA) and National Aeronautics and Space Administration (NASA) will become involved.
- The United Nations Office for Outer Space Affairs (UNOOSA) has published The Moon Agreement which limits what is allowed within lunar missions [3].

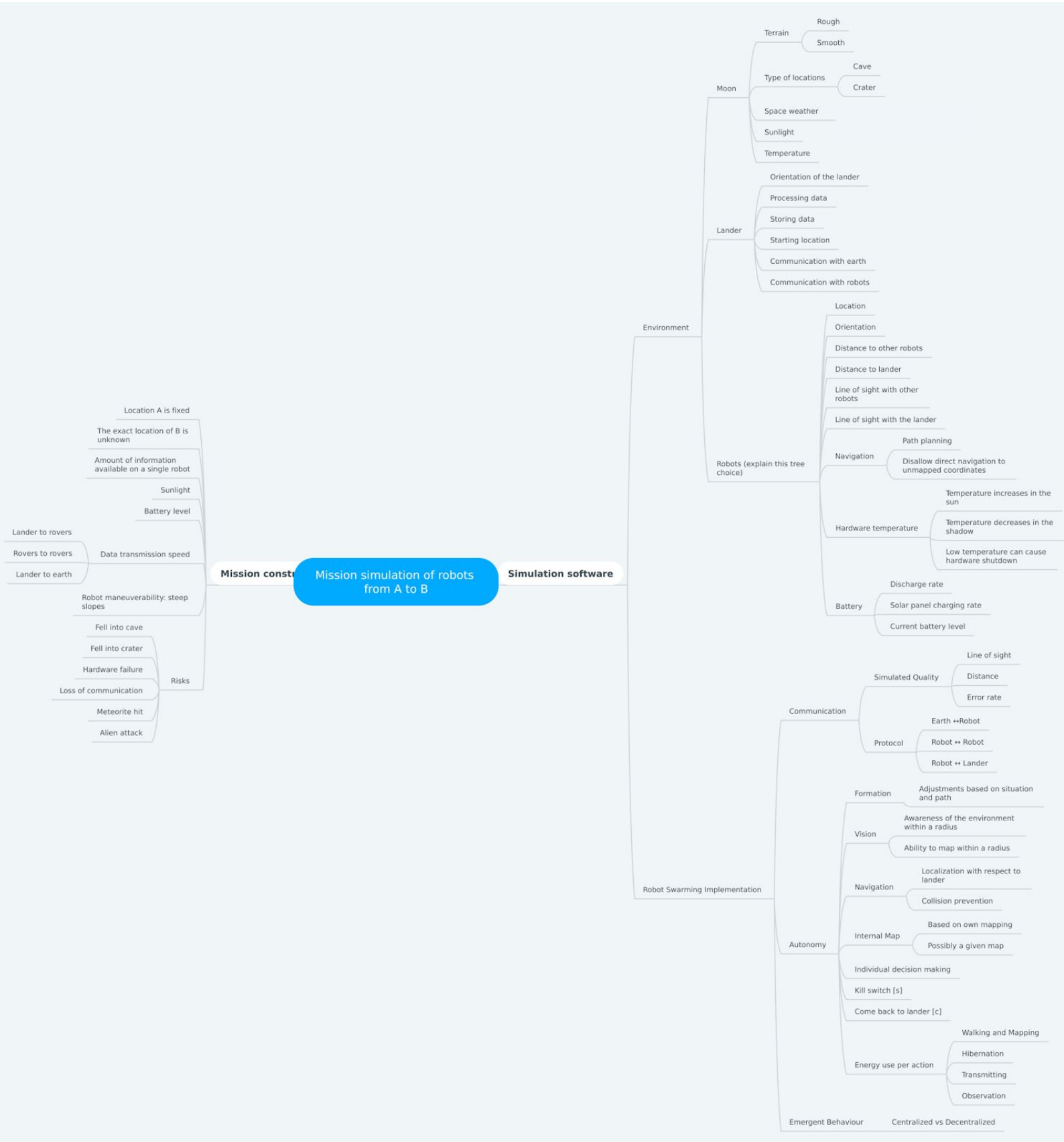


Figure 1: The discovery tree of the rover and the simulation environment. It describes the concepts involved in the project in a high-level overview. The reader is informed that this diagram cannot be properly viewed when the document is printed.

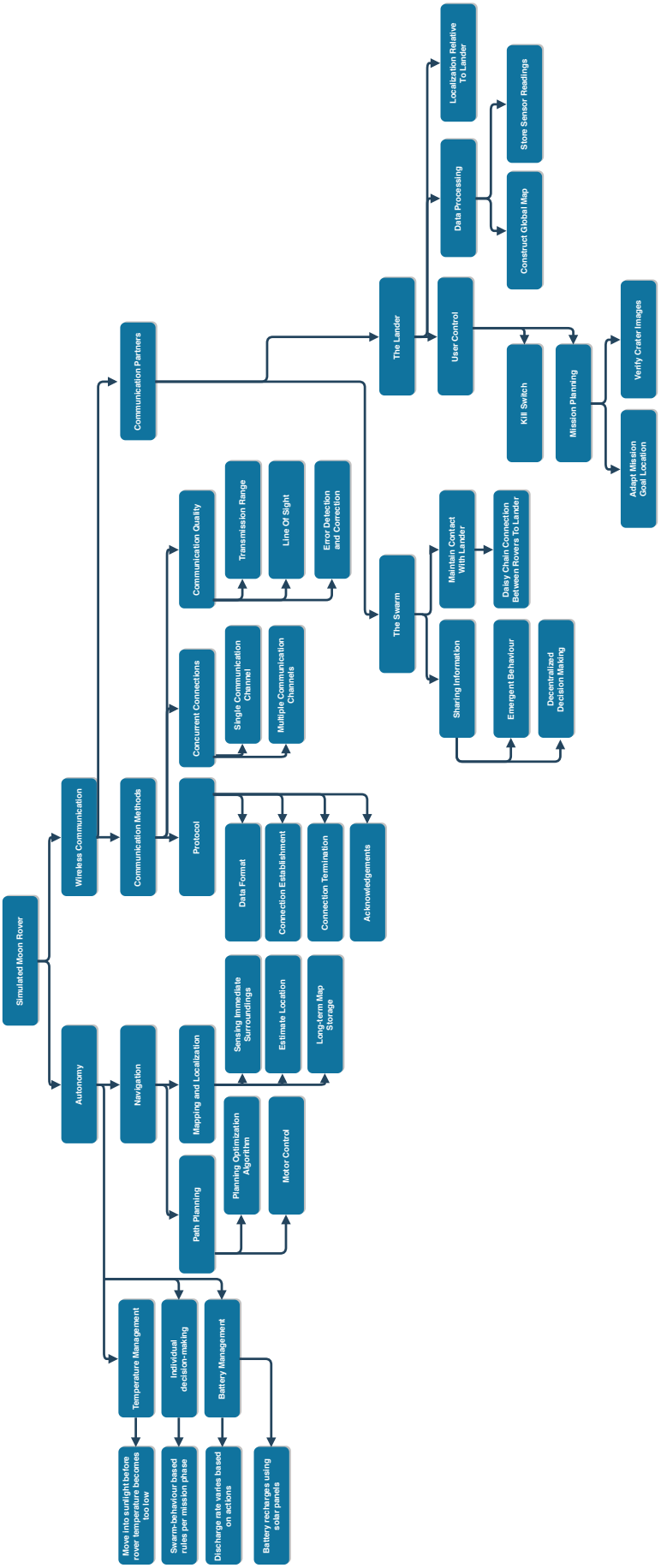


Figure 2: The functional tree of the simulated rover. It describes what the rover should be able to do but not necessarily how this is done with actual hardware. The reader is informed that this diagram cannot be properly viewed when the document is printed.

3 Requirements

Below, the project requirements based on the needs formulated in section 2 are defined. The requirements are prioritised based on the MoSCoW method. The requirement have a tag associated with it, based on the needs of Lunar Zebro (N1-N11).

3.1 Requirements

Must Haves

- The rovers must be capable of navigating from location A to location B. [N1]
- During navigation the rovers must avoid collision with obstacles and each other. [N1]
- During navigation the rovers must stay within communication range of at least one other rover. [N1]
- Each individual rover must maintain the communicative connection with the lander when it travels outside of the transmission range of the lander by means of daisy-chaining. [N3]
- Properties of the simulated sensors and rover components must be parameterised such that they can be configured by the ground station. [N8]

Should Haves

- The rovers must be able to communicate with each other to coordinate their actions as they move and explore. [N11]
- The ground station should be able to control and change the goal of the rover. [N5]
- The rovers should be able to change its direction of migration during exploration if they receive a new target location. [N1]
- Rovers should be capable of exploring and mapping an area around a specified location. The simulation of the mapping should not involve simulation of actual image processing. [N2]
- Sunlight should be simulated to facilitate simulation of temperature and battery recharging for each individual rover. [N9][N10]
- Rovers should stay as much within sunlight as possible to prevent their temperature from becoming too low and to recharge.
- Rovers should identify no-go (no-explore) areas (like the inside of the crater or an area without sunlight) during exploration. [N2]
- Rovers should be able to communicate images or topological data to the lander and the main mode of communication is via the lander. [N3]
- Individual navigation of a rover within the swarm should be independent. The lander can define the goal of the rover in terms of what direction to explore. However, the rover independently decides how to move in that direction and how to respond based on its visual perception. [N4]
- Users are able to change the current emergent actions of the swarm. [N5]
- Rovers should have a user-initiated individual action communicated via the lander that can make them return to the lander. [N6]

Could Haves

- The temperature of the rovers could be simulated, if the temperature becomes too low the rover shuts down. [N9]
- The battery level of the rover could be simulated, specific actions cause specific pre-defined battery drain and the battery can be recharged by means of solar panels. [N10]
- The rovers are capable of forming an arbitrary formation. [N7]
- The rovers are capable in moving in an arbitrary formation. [N7]

Won't Haves

- The rover will not be able to do exploration tasks in caves or craters.
- The rovers will not be able to perform a swarm dance by means of synchronous movement.

3.2 Software Tools

Based on the requirements, previous experience with robot simulations and a brief literature study the team decided to use the following tools for the development of the project:

- Ubuntu 16.04 as an operating system.
- Unity 2019.1.0f2 for developing the simulation environment.
- ROS Kinetic for developing the control algorithms of the simulated rovers.
- A ros_bridge server for connecting the Unity and the ROS process.
- The ROS-sharp plugin for interfacing between Unity and ROS topics.

For more details on what components of Unity are used for the simulation the reader is referred to subsection 5.3.

3.3 Team Roles

This subsection describes the individual responsibilities of all the team members.

- **Dana van Hassel:** Swarm Behaviour & Lead Testing & Lead Interaction Design
- **Maxim Liefaard:** Simulation Environment & Lead Programmer
- **Raoul Bruens:** Mission Planning & Sprint Manager
- **Sterre Noorthoek:** Swarm Behaviour & Chief Editor

3.4 Development Timeline

This subsection includes a generalized development timeline. Internally the team will use Trello for sprint organisation and tracking. Throughout the project Toggl will be used as a tool to track the time spent by each team member on the project.

As required by the TU Delft, there will be several code review throughout the project. These code reviews will be conducted by Software Improvement Group (SIG) and are noted in the timeline.

Week	What	Who
1	Initial setup	Everyone
	Plan of action	Everyone
	Literature study	Everyone
	Start mission definition	Raoul
2	Mission definition done	Raoul
	Literature study	Everyone
	Research report	Everyone
	Software requirement definition	Everyone
3	Learn ROS	Everyone
	Software repository setup	Everyone
4	Development for robot sim and environment sim	Maxim& Raoul
	Start algorithm development	Dana & Sterre
5	Further simulation improvement	Raoul & Maxim
	Further algorithm improvement	Raoul & Maxim
6	First SIG submission	Everyone
	Swarm and simulation development	Everyone
7	Implement SIG and feedback stakeholder	Everyone
	Swarming testing and development	Dana & Sterre
8	Simulation finalisation	Everyone
	Start writing report	Everyone
9	Write report	Everyone
	Second SIG submission	Everyone
	Perform and evaluate simulation test runs	Everyone
10	Final report	Everyone
11	Presentation	Everyone

4 Mission Planning

This chapter describes the scenario that will be simulated in terms of certain mission phases that are characterised by phase goals. From this, it is defined what the goal of the mission is and what challenges need to be overcome to be able to complete each phase of the mission.

4.1 Scenario Description

4.1.1 Mission Description

The scenario can be described as follows. A lander loaded with a set amount of rovers will descend upon an arbitrary location on the edge of the far side of the Moon. This is referred to as location A. From there, the goal is to have all rovers move out and explore the path to a certain location B on the far side. Subsequently, the rovers should explore the area around location B.

The rovers will be able to communicate to one another during this process. Communication to and from Earth is done through the lander. The lander also keeps track of all the data gathered by the rovers; it acts as the main computational hub of the swarm.

To this end, the scenario definition will be refined through division into four distinct phases: dispatch, calibration, migration, and observation.

4.1.2 Moon Characteristics

The landing site on which the rovers will be simulated is situated on the Moon. To this end, we will describe some characteristics of the Moon to introduce what we can expect for our rovers to endure on the Moon.

Since the Moon rotates synchronously with the Earth, also known as tidal locking, we only see one side of the Moon at all times. This part is often referred to as the near side of the Moon. Naturally, the other side is called the far side. This distinction is important, since communication links with Earth cannot be directly established when an entity is on the far side of the Moon. This

entity would have to establish a connection by other means, for example through another entity on the near side or through an orbiter.

The surface temperature on the Moon can vary significantly. The temperature when there is full Sun can reach 127 degrees Celsius, while in darkness, the temperature can reach a minimum of -173 degrees Celsius. Taking into account that the day/night cycle on the Moon lasts for 27 days, we can conclude that any entity on the Moon will be exposed to prolonged extreme temperatures. Modelling the effects of temperature on the individual electronics of the rover will be out of scope for the simulation. However, the day/night cycle will be taken into account to determine when the rover's battery is charging.

While the end goal of the simulated mission would be to obtain topological and visual data of the Shackleton crater on the lunar south pole, the simulation will aim to facilitate a configurable landing site. The Shackleton crater's rim is an interesting place as a future landing site, since its perpetual exposure to sunlight makes it ideal for reliably obtaining electricity by means of solar panels [4]. Also, there have been indications of ice inside such polar craters, although no definitive proof of this is known [5]. In future missions, like the one this project aims to simulate, more data will have to be gathered to (dis)proof this.

4.1.3 Past Rover Missions

In this section similar successful rover missions are listed with a short description on the mission execution. This should give a better view on what to expect during a mission on the lunar surface. An overview of the described missions is given in subsubsection 4.1.3

Rover	**Country**	**Launched**	**Operated for**
Lunokhod 1	Soviet Union	November 1970	321 days
Lunokhod 2	Soviet Union	January 1973	117 days
Yutu	China	2013	973 days
Yutu 2	China	2019	117+ days

Table 1: Overview of a view rover missions

Lunokhod 1 This rover would operate during lunar daytime and hibernate at night, when it would be heated by a radioactive source. While on the surface, it drove more than 10 kilometres and performed dozens of soil samples. The rover was designed to operate for three lunar days, however, it continued to operate for eleven days. It seized operation because reflected signals from its laser ranging could not be located, effectively making the rover untraceable until 2010, when it was rediscovered using NASA's Lunar Reconnaissance Orbiter. [6]

Lunokhod 2 This rover operated similar to Lunokhod 1. It broke down on its fifth lunar day as dust got into the radiators, which caused the rover to overheat. The dust got in there presumably while it was charging. Dust would have gotten onto the unfolded solar panel, which would allow dust to get in to the instrument bay once the panel was folded back.

Yutu After the rover was deployed on the Moon, it began on its initial mission: to photograph the lander from several different angles. The rover survived its first lunar night on the Moon and was able to successfully continue its mission during the day. Starting the second lunar night, the rover experienced control circuit malfunction, which prevented the rover from entering hibernation properly as it could not fold all its instruments back. On the third lunar day, connection with the unit was initially lost, but the rover unexpectedly re-established a connection after a day. Though it was immobile as a result of its control failure, its ground-penetrating radar and imaging sensors were still functioning properly. On its thirty-fifth lunar day, Yutu ceased to communicate, effectively concluding its mission.

Yutu 2 Yutu 2 will be the first rover exploring the far side. As this inhibits the rover from sending data directly to Earth, it relies on a relay satellite to communicate to Earth, which is in gravitationally stable orbit (also known as Earth-Moon L_2) such that it can establish communication links with both units on the far side and Earth.

Concluding Remarks From these past missions, we can conclude a few things. Until now, rovers that have been situated on the Moon have been brought there one at the time. Generally,

these rovers drove around and gathered data unaccompanied. However, the downside of this is that the success of a mission depended, apart from any launch or spacecraft failures, on the fitness of the rovers. Having multiple small, cheap robots would mitigate the risk of mission failure due to rover malfunction. The swarming behaviour could potentially facilitate single rover loss, as described in section 6.

4.2 Mission Phases

Stellar Space Industries is already developing in the Lunar Zebro rover and created a mission phase diagram for the current design, which can be found in Figure 3.

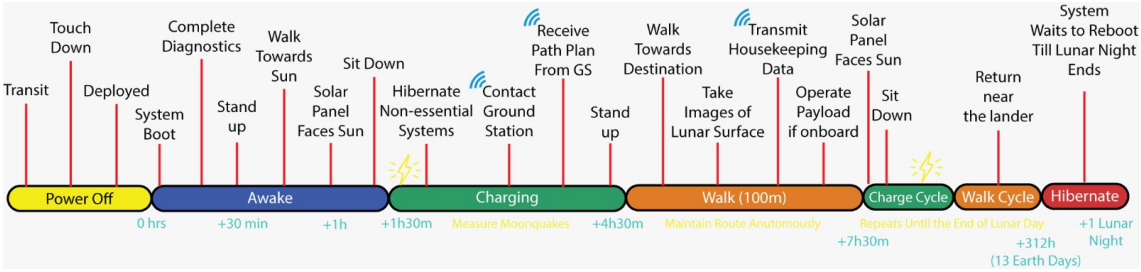


Figure 3: The phases of the current design of the Lunar Zebro project as defined by Stellar Space Industries.

For the purpose of the specific mission that is simulated the team defined simplified mission phases derived from the mission planning of the Lunar Zebro project. Below, four mission phases are described in terms of their goals. That is, the goals act as requirements for the completion of the specific phase. The phases are visualised in Figure 4.

4.2.1 Phase 1: Dispatch

- Rovers have entered the Moon's surface.
- Collisions are avoided during disembarkation.
- Rovers are spread around the lander.

4.2.2 Phase 2: Calibration

- Lander has used star navigation to determine its location and orientation.
- Rover localisation systems are calibrated through mapping of the landing site.

4.2.3 Phase 3: Migration

- Rovers are in formation before migration.
- Rovers have arrived at location B.
- The path between location A and location B is mapped.
- Communication with the lander has been maintained.

4.2.4 Phase 4: Observation

- Map of surroundings of location B have been created.
- The map of location B should be send to Earth.
- Rovers should be in hibernation mode.

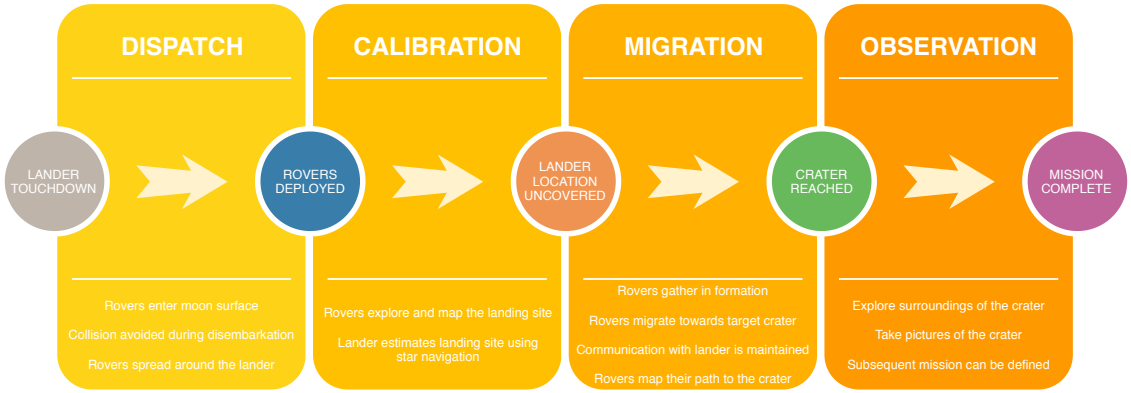


Figure 4: A diagram of the mission phases and their subgoals.

4.3 Mission Goal

The mission goal is to map a location B on the far side of the Moon, which is most likely going to be a Moon crater, such that further actions can be deduced from this map. From this, subsequent mission scenarios can ensue, like exploring the crater or cave at location B, or mapping of a new location C.

5 Simulation Environment

This section describes the various components of the simulation environment that will be developed to test different algorithms for the rovers' behaviour in the mission scenario as described in section 4.

5.1 Simulation Scope

First, the scope of the simulation is determined. Existing lunar simulations vary in complexity. Most simulations concentrate on the locomotion of rovers and the traction of their wheels [7], while other simulations are developed for navigation and path planning. [8] [9]

For the purpose of simulating behavioural algorithms it is unnecessary to simulate traction and individual rover path planning. In order to focus more on the emergent behaviour of the group of rovers the simulation should allow easy navigation such that more development time can be spent on the behavioural algorithms themselves.

The rovers also need to be able to observe their surroundings, this is often done using complex Simultaneous Localization And Mapping (SLAM) algorithms [10]. Simulating this is complex and does not help in developing behavioural behaviour. Instead it can be assumed that the rover has good vision and mapping capabilities for its direct surroundings. The simulation can provide a list of entities within a radius of the rover which are relevant for the behavioural algorithm.

The rovers' behaviour can rely heavily on communication, this is why this is important to simulate. A communication protocol and variations in communication quality can be simulated [11].

Overall, the focus of the simulation must be the testing of rovers' behaviour on a scalable number of rovers. For this reason it is important not to simulate individual sensors or low-level software subsystems like image processing and path planning. The assumption is made that the simulated rover has all the required functionality and that it can be controlled by a high-level behaviour implementation.

5.2 Simulation Features

Having declared the scope of the simulation, the following subsections focus on describing the details of the distinct features of the simulation.

5.2.1 Simulation Entities and Lunar Surface

In order to simulate rovers on the Moon it is required to have some representation of a lunar surface. On this lunar surface there should be entities that represent the rovers. Furthermore, an entity of the lander that brings the rovers to the Moon should be included. Collisions between these entities must be simulated.

5.2.2 Rover Navigation

Each rover must be able to navigate the lunar surface. The simulation of individual rover navigation should remain simple because the focus of the simulation is to test behavioural algorithms. The path planning can be done by existing algorithms while the implementation of the behavioural algorithm only specifies coordinates that the rover must travel to. Furthermore, the environment could contain obstacles that cannot be traversed and slopes which can affect the movement speed of the rover. The rover must also be able to determine its own position and orientation in the environment.

5.2.3 Rover Awareness

The rover will be equipped with sensors that allow it to map its surroundings and determine where other rovers or obstacles are. The mapping algorithms involved in this are left beyond the scope of the simulation. Instead, the simulation provides information about its surroundings within a certain radius. This information can then be used to determine the next action of the rover.

5.2.4 Communication

Rovers are able to communicate with one another and with the lander. The lander is also able to communicate with the the ground station on Earth. The simulation determines whether two entities can communicate and what the quality of the communication is. Entities cannot communicate if there is no line of sight and the quality decreases as the distance increases. Errors in the transmissions can be simulated by decreasing the transmission speed.

5.2.5 Temperature and Battery

On the Moon there is always the risk that comes with the rough terrain, such as falling into a cave. There are also two additional factors that can cause a rover to become unavailable. The first is when the temperature becomes too low. The second is when the battery of the rover is drained, which can be recharged with its solar panels. Thus, simulating sunlight becomes important. From the group perspective it is important that individual rovers are regularly located in direct sunlight to recharge and to raise their temperature. The battery and temperature of each individual rover also has to be simulated for this feature.

5.3 Software Architecture

Developing a 3D simulation for the environment with all these required features is a challenge within the scope of the project. This is why the project will be based on the Unity3D engine, which has a large portion of the required features already implemented:

- Simulation entities can be defined as Unity GameObjects [12].
- Positioning and orientation of entities can be tracked using the Unity Transform. Component [13]
- For a lunar surface, 3D models provided by NASA can be used and imported into Unity. [14]
- Collision between entities and the lunar surface can be done using the Unity Physics Engine. [15]
- Pathfinding can be done by processing the environment model as a NavMesh and developing the Rover Entities as NavMesh Agents. [16]
- Light simulation can be done using the Unity Lighting System. [17]

The behaviour control of the rovers will be developed using Robot Operating System (ROS). Control algorithms for robotics applications are often developed in ROS and there are many open-source projects that can be used as a reference. Furthermore, there exist ROS packages that aid in implementing behavioural behaviour.

The connection between the ROS control algorithms and the Unity simulation environment is established using a ros_bridge server and the ROS-sharp plugin for Unity. A diagram of the software architecture design can be found in Figure 5.

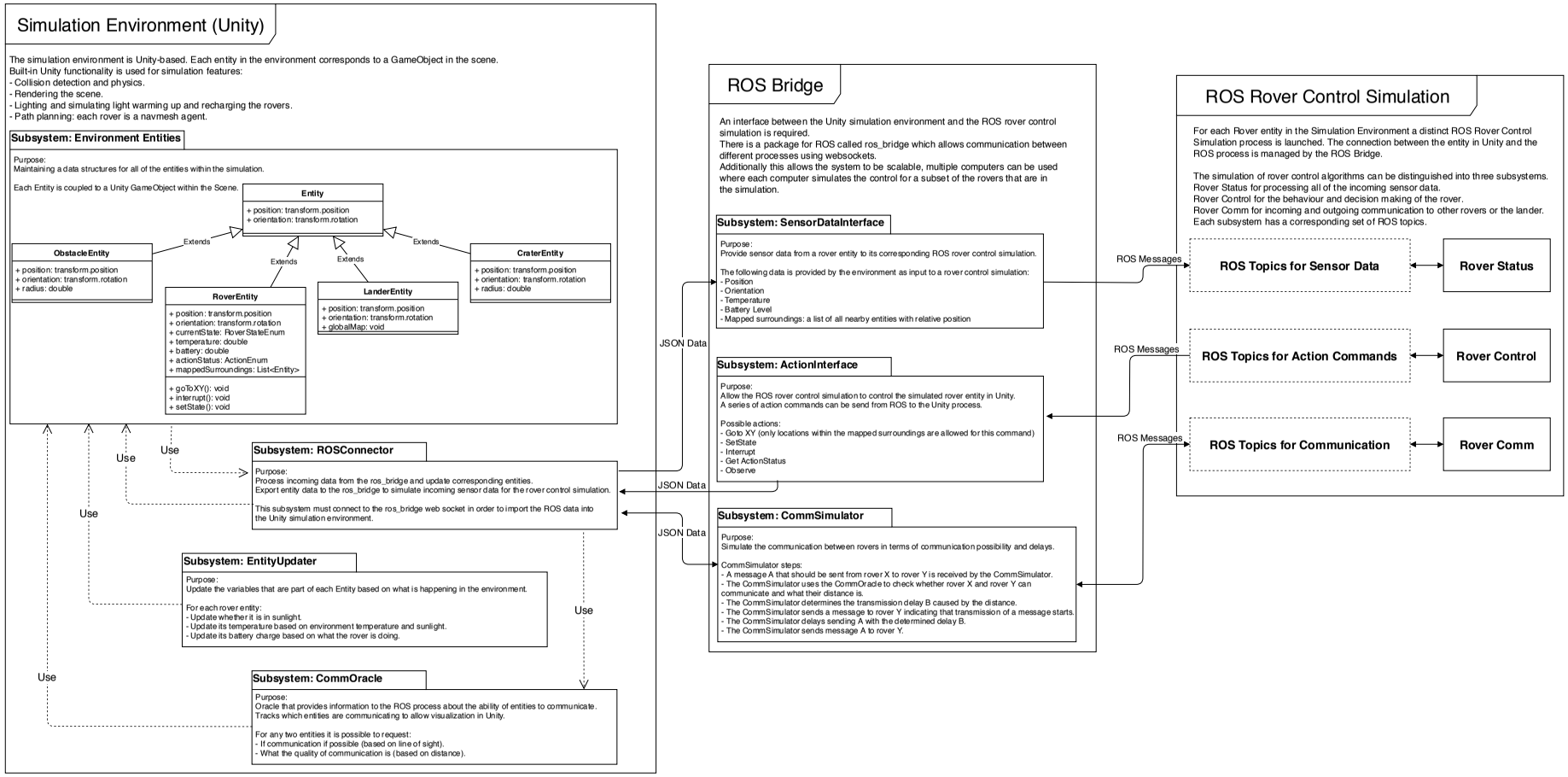


Figure 5: Architecture Design of the Unity simulation environment and its connection to the ROS. The reader is informed that this diagram cannot be properly viewed when the document is printed.

6 Rover Behaviour

This section focuses on the behavioural algorithms applied to the rovers to complete the mission. Two approaches for programming robot behaviour are looked at. Firstly, swarming is discussed. In this section, general characteristics of swarms are described, a couple of swarming algorithms inspired by nature are briefly explained, and the usefulness to the mission is argued. Secondly, multi-agent systems are discussed. This section discusses characteristics and usefulness of MAS to the mission. Thirdly, the development strategies for the rovers' behaviour are discussed. In this section, the objective is defined and strategies are defined to complete the mission.

6.1 Swarming

Swarming is a phenomenon in nature where large groups of the same species, for example bee colonies, ant colonies, bird flocks, and schools of fish, work together. Working as a team in large groups has shown to be an effective strategy to survive in nature and swarming has evolved over many years of evolution. Individual members of a swarm cannot achieve much, but when they work together they can form a collective intelligence.

6.1.1 Characteristics of Swarms

Three characteristics of swarms are identified: how the individuals make decisions (decentralised vs (strictly) centralised), how the swarm is composed (homogeneous vs heterogeneous), and the complexity of the individuals (low vs high complexity).

The decisions made in a swarm are of great importance of the survival of the swarm. Decision-making can be split into three categories: decentralised, centralised, and strictly centralised. In decentralised decision-making individual decisions are made individually and group decisions are made indirectly through all individual decisions. In centralised decision-making the group decisions are made the by the leader(s). In strictly centralised decision-making both the individual and group decisions are made by the leader(s). Strictly centralised decision-making appears in robotics but not in nature, because it is impossible for the leader to control the exact movements of all individuals.

When comparing decentralised and centralised decision-making in computer simulations of swarms, both have certain advantages and disadvantages. Advantages of decentralised decision-making are that such swarms are robust to failing individuals since they are not dependent on a leader, that individual decisions can be made quickly, and that the size of the swarm is scalable. The advantages of (strictly) centralised decision-making are that group decisions can be made quickly and that the behaviour of the swarm is more predictable.

Swarms can be made up of individuals that are identical and have similar characteristics and capabilities (homogeneous) or they are made up of individuals that are different and also have different characteristics and capabilities (heterogeneous). An advantage of a homogeneous robot swarm is that the robots can be very simple and are easily replaceable if one breaks down. For example, the kilobots developed at Harvard Univeristy. [18] A heterogeneous robot swarm, on the other hand, can be used to solve multiple different tasks. There exists, for instance, a robot swarm that can do obstacle avoidance, human rescue operations, wall painting and mapping. [19]

Individuals in swarms can have a low or high complexity. In decentralised swarms, individuals do not need to have a high complexity. The intelligence comes from the swarm rather than from the individuals. The individuals simply need to follow basic rules in order to achieve this. In robotics, this low complexity in a decentralised swarm has a couple of advantages. The swarm is less likely to run into failures since a lower complexity is less likely to have contain bugs. Secondly, the production costs lie lower. For centralised robot swarm, a leader is responsible for group decisions. The leader has a higher complexity than the rest in order to make these decisions.

6.1.2 Swarming Algorithms

Swarming in nature has been an inspiration for designing algorithms. Two well known swarming algorithms are discussed. The ACO algorithm is used to find an optimal path in a graph or maze. It is inspired by the way ants communicate with one another. When an ant finds food it takes a part of it and brings it back to its nest. On the way back ants leave behind even more pheromones to

communicate the food source with others. If more ants follow the trail the scent becomes stronger. This is also the reason why ants walk in lines. The ACO algorithm simulates ants walking through a maze. Each ant leaves a pheromone trail on the path it has walked on. The algorithm gives shorter paths a stronger pheromone scent, as those paths are favoured. Ants are more likely to choose a path with a stronger scent. Evaporation of scent is mimicked by lowering all pheromones at each iteration. This decreases the likelihood of the ants getting stuck in a local optimum.

The PSO algorithm is a fast but computationally intensive algorithm that is inspired by flocking birds and used to efficiently search in a virtual space. [20] The PSO algorithm models a search in a virtual world and it difficult to apply is to the real world. The PSO algorithm assumes that particles are infinitely small, that particles can teleport at each iteration, and that there is no built-in collision mechanism.

Plugh and Martinoli [21] use the PSO algorithm to create a model, where each robot is modeled separately. Each particle of the PSO algorithm represents a single robot. It is assumed that the robots can communicate with one another and that they know where they are in the environment. The model adds a couple of additional properties to the robots such as continuously updating their location, not being able to teleport, having a build in collision mechanism, and having a limitation on how fast they can move and accelerate.

6.1.3 Usefulness of Swarming for the Mission

This section discusses in what way swarming is relevant and if it can be used as an inspiration for the mission. The section before looked into three characteristics of swarms in nature and robotics. Since swarming is a very broad concept which makes it difficult to make general statements about it, the definition is narrowed down. In this research report a swarm is defined as: a homogeneous and decentralised group of at least one hundred individuals (where an individual can be a (simulated) human, animal, or robot) that can perform a maximum of twenty instructions.

The Ant Colony Optimization (ACO) algorithm and the Particle Swarm Optimisation (PSO) algorithm are both not a good fit for the mission as they have a different purpose and use a different environment. Both algorithms are optimisation algorithms. The Lunar Zebro rovers simply need to find a path to location B. They do not need to repeat finding different paths to location B in order to find the shortest one as is done in the ACO algorithm. Secondly, the algorithms would have to be extensively adjusted since they are made for virtual spaces and the nodes do not have the same properties as objects in real life.

This report's definition of swarming includes a large group of individuals with a low complexity that perform simple tasks. However, the mission of this project is more complex and exists of multiple different tasks that cannot just be performed by a single swarming algorithm. The Lunar Zebro rovers have a higher computational power and should have to use this in order to be prepared for different tasks of the mission.

6.2 Multi-Agent Systems (MAS)

A multi-agent system is a system that is self-organised and consists of multiple intelligent agents. These agents can be humans, programs, or robots. This section focuses on multi-agent systems for robots.

6.2.1 Characteristics of Multi-Agent Systems

Most multi-agent systems share similar characteristics: agents are autonomous, only have local views, and are decentralised as they cannot control the others. A characteristic similar to swarms is how the system is composed (homogeneous vs heterogeneous). Other characteristics of multi-agent systems that can vary describe how the agents are co-existing and how they communicate.

They way that agents are co-existing can be categorized in: merely coexisting, loosely couple, and tightly coupled. In merely coexisting the agents treat one another as obstacles. In loosely coupled the agents can communicate but do not depend on one another. In tightly coupled, the agents are actively working together. The way agents communicate can be divided in direct and indirect communication. Indirect communication takes place when an agent makes a choice based on the behaviour of another agent.

6.2.2 Usefulness of Multi-Agent Systems for the Mission

This section takes a look at in what way multi-agent systems are relevant and if they can be used as an inspiration for the mission.

Swarming and multi-agent systems share a couple of similar advantages: both are scalable and robust to failing individuals. A difference between swarming and multi-agent systems is that multi-agent systems allow for a higher complexity of individuals. This allows agents in MAS to have different tasks. In the mission this can be used during the migration phase, where some rovers will walk and some rovers will become part of the daisy chain.

6.3 Development of Strategies for the Mission

This section defines the goal of the mission, limiting factors of the rovers, and evaluation of the simulation. It then focuses on the development of a strategy to eventually design an algorithm that completes the goal of the space mission of the Lunar Zebro rovers.

6.3.1 Goal of the Mission

The goal of the swarm is to move from point A to point B, where point A (the lander) is known but with an error rate and only the direction to point B (the Shackleton crater) is known.

6.3.2 Limiting Factors of the Rovers

The limiting factors for the rovers that need to be taken into account in order to complete the mission are:

- The availability of energy sources.
- The rough terrain on the Moon
- The unknown exact locations of A and B (there is no GPS)
- The extreme fluctuating temperatures.
- The rover's intelligence to recognise location B

6.3.3 Evaluation of the Simulation

The measurement of fitness evaluates the effectiveness of an algorithm in a certain environment. For this mission, it measures several aspects. The most important factor is power budget efficiency. Efficiency is defined as the rate between the current situation to the optimal situation. After that (in this order of importance), the other aspects are used to rate an algorithm:

- Power budget efficiency
- The time it takes the swarm to reach location B
- Path efficiency
- How many rovers break down (defined as permanently unavailable)
- How many rovers are stuck (defined as temporarily unavailable)
- The amount of times that the connection with the lander is lost
- The total duration of connection loss between the swarm and the lander
- The number of rovers in the communication daisy chain to the lander (lower is better)
- How many rovers collide with one another or with obstacles in the environment

6.3.4 Identify Strategies

It is not possible to come up with a single swarming algorithm that solves the goal at once. Therefore, the mission is split into phases as described as in section 4. Each phase is a sub task of the general objective, with the last phase "Explore surroundings of B" as an additional sub task. For each phase a strategy is identified. A simple rule based approach is chosen where possible. For rather difficult tasks, nature and existing approaches are used as an inspiration.

Phase 1: Dispatch
A simple rule-based approach will be used, so that the rovers leave the lander one by one.

Phase 2: Calibration
Initially, the rovers need to map the surrounding of the lander to correct the error of the lander's location. The rovers will form a circle around the lander and they will take images of both the environment and the lander. This phase is also used as a test to see if the rovers and the communication works well.

Phase 3: Migration
This phase consists of three sub tasks: create the initial formation, migrate, use a daisy chain, and confirm arrival at location B.

The initial formation consists of two simple rules about their distance to one another. The rovers walk in a formation where they are close enough to communicate, but far enough to prevent collisions. The sub task of migration is based on three simple rules: walk in the direction of location B, avoid collisions with neighbours and objects, and remain within neighbors communication range.

While migrating a daisy chain of rovers is needed to maintain communication with the lander. This is a type of network topology that directs how network nodes are linked, these are typically computers but can also be rovers. A node is attached to the next one in a line or chain. The rule-based approach that will be applied is: do migration until connection with lander is lost, stop migration, let one rover walk back until connection is restored, set this rover up as daisy chain node, and continue migration.

For the final sub task we use a simple rule-based approach. Once a rover thinks it might have arrived at location B, it makes images of location B and sends these to the lander for confirmation. If location B has indeed been reached, the rover communicates to the other rovers that they should also come to that location.

Phase 4: Observation
In the observation phase the rovers will take images of the environment and send these to the lander.

References

[1] M. á. van Haarlem, M. Wise, A. Gunst, G. Heald, J. McKean, J. Hessels, A. De Bruyn, R. Nijboer, J. Swinbank, R. Fallows, *et al.*, "Lofar: The low-frequency array," *Astronomy & astrophysics*, vol. 556, p. A2, 2013.

[2] M. K. Verma, C. Verhoeven, and R. T. Rajan, "Mission concept for lunar low frequency antennas for radio astronomy (lufar)," *70th International Astronautical Congress*, 2019.

[3] C. Q. Christol, "The moon treaty enters into force," *American Journal of International Law*, vol. 79, no. 1, pp. 163–168, 1985.

[4] K. Fristad, D. Bussey, M. Robinson, and P. Spudis, "Ideal landing sites near the lunar poles," in *Lunar and Planetary Science Conference*, vol. 35, 2004.

[5] L. Qiao, Z. Ling, J. W. Head, M. A. Ivanov, and B. Liu, "Analyses of lunar orbiter laser altimeter 1064 nm albedo in permanently shadowed regions of polar crater flat floors: Implications for surface water ice occurrence and future in-situ exploration," *Earth and Space Science*, 2019.

[6] Kim McDonald, "Locate long lost soviet reflector on moon." https://ucsdnews.ucsd.edu/archive/newsrel/science/04-26SovietReflector.asp. Online; accessed May 2, 2019.

[7] A. Azimi, J. Kövecses, and J. Angeles, "Wheel-soil interaction model for rover simulation based on plasticity theory," in *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 280–285, IEEE, 2011.

[8] H. Gao, Z. Deng, L. Ding, and M. Wang, "Virtual simulation system with path-following control for lunar rovers moving on rough terrain," *Chinese Journal of Mechanical Engineering*, vol. 25, no. 1, pp. 38–46, 2012.

[9] W. Li, Y. Huang, Y. Cui, S. Dong, and J. Wang, "Trafficability analysis of lunar mare terrain by means of the discrete element method for wheeled rover locomotion," *Journal of Terramechanics*, vol. 47, no. 3, pp. 161–172, 2010.

[10] A. Kim and R. M. Eustice, "Perception-driven navigation: Active visual slam for robotic area coverage," in *2013 IEEE International Conference on Robotics and Automation*, pp. 3196–3203, IEEE, 2013.

[11] R. Doriya, S. Mishra, and S. Gupta, "A brief survey and analysis of multi-robot communication and coordination," in *International Conference on Computing, Communication & Automation*, pp. 1014–1021, IEEE, 2015.

[12] Unity Technologies, "Unity - manual: Gameobject." https://docs.unity3d.com/Manual/class-GameObject.html. Online; accessed May 2, 2019.

[13] Unity Technologies, "Unity - manual: Transform." https://docs.unity3d.com/Manual/class-Transform.html. Online; accessed May 2, 2019.

[14] NASA, "Nasa 3d resources." https://nasa3d.arc.nasa.gov/search/lunar. Online; accessed May 2, 2019.

[15] Unity Technologies, "Unity - manual: Physics." https://docs.unity3d.com/Manual/PhysicsSection.html. Online; accessed May 2, 2019.

[16] Unity Technologies, "Unity - manual: Navigation system in unity." https://docs.unity3d.com/Manual/nav-NavigationSystem.html. Online; accessed May 2, 2019.

[17] Unity Technologies, "Unity - manual: Lighting overview." https://docs.unity3d.com/Manual/LightingInUnity.html. Online; accessed May 2, 2019.

[18] M. Rubenstein, C. Ahler, N. Hoff, A. Cabrera, and R. Nagpal, "Kilobot: A low cost robot with scalable operations designed for collective behaviors," *Robotics and Autonomous Systems*, vol. 62, no. 7, pp. 966–975, 2014.

[19] M. D. Patil and T. Abukhalil, "Design and implementation of heterogeneous robot swarm," in *ASEE 2014 Zone I Conference*, pp. 3–5, 2014.

[20] R. Eberhart and J. Kennedy, "Particle swarm optimization," in *Proceedings of the IEEE international conference on neural networks*, vol. 4, pp. 1942–1948, Citeseer, 1995.

[21] J. Pugh and A. Martinoli, "Inspiring and modeling multi-robot search with particle swarm optimization," in *2007 IEEE Swarm Intelligence Symposium*, pp. 332–339, IEEE, 2007.