Tartu University

Faculty of Science and Technology

Institute of Technology

Hans Pärtel Pani

Development of a 3D web visualization component for KuupKulgur Mission Control Interface

Master's thesis (30 EAP)

Robotics and Computer Engineering

Supervisor:

MSc Quazi Saimoon Islam

Tartu 2025

Resümee/Abstract

Development of a 3D web visualization component for KuupKulgur Mission Control Interface

This thesis presents the development and design of a web based 3D visualization component prototype for the KuupKulgur Lunar rover. Several existing robotics visualization software options were analyzed and the ideas incorporated into making this component. The software was built using well-established web development libraries like Lit and Three.js. The performance of the component was tested using several datasets of pre-collected data and tested in simulated scenarios.

Keywords: robotics, visualization, three.js, web development, ros, robot operating system

CERCS: T125 Automation, robotics, control engineering; P170 Computer science, numerical analysis, systems, control

KuupKulgur Mission Control Interface veebirakenduse jaoks 3D visualiseerimise komponendi arendus

See magistritöö käsitleb veebipõhise visualiseerimise komponendi prototüübi loomist KuupKulgur kuukulguri veebirakenduse jaoks. Analüüsiti mitmeid olemasolevaid robootika visualiseerimistarkvara lahendusi ning nende ideid kasutati komponendi loomisel. Tarkvara loodi kasutades tuntud veebiarenduse teeke nagu Lit ja Three.js. Komponendi jõudlust testiti kasutades eelnevalt kogutud andmestikke ja simuleeritud olukordades.

Märksõnad: robootika, visualiseerimine, three.js, veebiarendus, ros, robot operating system

CERCS: T125 Automatiseerimine, robootika, control engineering; P170 Arvutiteadus, arvutusmeetodid, süsteemid, juhtimine (automaatjuhtimisteooria)

Contents

List of Figures

List of Listings

Acronyms and Abbreviations

AR - augmented reality

ESA - European Space Agency

FPS - frames per second

GUI - graphical user interface

HTML - hypertext markup language

JSA - Japanese Space Agency

LIDAR - Light Detection and Ranging

LOLA - Lunar Orbiter Laser Altimeter

MCI - Mission Control Interface

NASA - National Aeronautics and Space Administration

NPM - Node Package Manager

NaN - not a number

ROS - Robot operating System

URDF - unified robot description format

VR - virtual reality

1 Introduction

Lunar exploration is seeing renewed interest around the world with plans to fully explore the moon in the near future. Nasa is planning to send astronauts back to the moon as part of the Artemis programme[1]. Meanwhile the European Space Agency (ESA) has come up with its own lunar exploration goals, the Explore_2040 programme, also called the Terra Nova[2]. It plans to bring European industry and science to the forefront of space exploration, with lunar exploration on the forefront of it. The lunar economy is estimated to grow to €40 billion in the next decade and further rise to €160 billion by 2040. As a foundational step ESA has started the Moonlight initiative to bring continuous network connectivity to the moon. There are plans to launch satellites which would provide low-cost network connectivity and positioning services on the lunar south pole by 2028, with further plans to provide full lunar coverage by 2030[3].

To discover the lunar economic and scientific potential ESA has held several calls for ideas for academics and businesses. For example, one of the calls for ideas proposed a use case scenario of a lunar power company managing their Lunar infrastructure from earth through ESA moonlight connectivity[4]. Besides governments there are a number of private companies already developing services to serve the space sector with well known examples like SpaceX, Virgin Galactic and Blue Origin innovating with providing lower cost launch systems; but also smaller companies like The Exploration Company developing space vehicles and landers[5].

In light of these developments in the near future low cost lunar exploration could be possible. Robotic lunar missions would no longer need to bring their own communications satellite or have to make use of very expensive rovers which are capable enough to communicate directly with earth from the Lunar surface. There are many low-cost rovers being developed already with one such example being the KuupKulgur rover under development in the University of Tartu at the Tartu Observatory[6].

However in order to control and monitor these low-cost lunar rovers, adequate mission control interfaces need to be developed that meet the demands of Lunar surface exploration. One key aspect of robotic control that has been evolving is the incorporation of web technologies to aid in

the development of robotics systems. This thesis focuses on the development of a prototype 3D visualization component to the existing web-based KuupKulgur mission control interface. The interface is currently used to control the rover in a Lunar analog environment on Earth, with a view to use the same approach for eventual Lunar use-case utilizing broader frameworks like ESA Moonlight.

1.1 Problem statement

KuupKulgur is a Robot Operating System (ROS) based rover being developed at the University of Tartu and it currently has a web-based interface to control it. As it is still in early stages of development, it can be controlled by a video feed and a joystick, however on the moon this might not be feasible due to the communications latency. Additionally scanned information of the lunar environment might not be possible to send back to earth at a high enough resolution and transfer speed to visualize the Lunar environment in sufficient detail. There is a need for a web-based 3D visualization component to visualize the rover and the environment around it. There are some popular existing tools, like RViz[7] which come installed with ROS, however they fall short due to not providing easy integration in a web application running in a browser. Some commercial options like Foxglove studio[8] have web based interfaces, however Foxglove is a standalone robotics debugging, visualization and control application and not a single component, which could be embedded in an existing application.

1.2 Objectives and roadmap

1. Research and analyze different robotics visualization approaches with a focus on lunar rovers
2. Design a web visualization interface
3. Identify key feature-sets needed for the web visualization interface
4. Test feature-sets using recorded data from KuupKulgur and simulated data

2 State of the Art

Human Robot interaction (HRI) is a wide field of study concerned with studying how humans perceive and interact with robots. It is closely related to other fields such as Human Computer Interaction (HCI)[9]. Much of the HRI research is focused on social robotics i.e how humans perceive and interact with robots in person[9], however it is also related to designing computer interfaces for robots. Robots occupying physical space and interacting with the environment, being directed by humans and having real consequences to their actions differentiate robotic interfaces from general computer interfaces. This embodiment and real world consequences to it require special approaches to HRI compared to HCI. Recognizing those distinctions, Murphy and Tadokoro outlined three distinct interface types for robotics: the developer interface, the end user interface and public interface; the developer interface being concerned with displaying various data to debug and develop robots, the operator interface having to simplify technical details and avoiding information overload and the public interface with providing explanations for the layman[10].

There is a lot of recent research into using virtual reality (VR) and alternate reality (AR) for interaction with robots[11], including some recent research at the University of Tartu: Reynes researched enhancing teleoperation using a VR interface[12] and Zorec investigated using VR to create a teleoperation demonstration[13]. However there seems to be little research in regards to designing non-VR/AR user interfaces, with existing tools like RViz being taken as the de facto standard[11]. With one of the few counter-examples being the Foxglove visualization suite[14].

For user interfaces in general there exist more general guidelines such as Jakob Nielsen's "10 Usability Heuristics for User Interface Design"[15] or specifically for robotics, the “Thirty-Two Guidelines for HRI User Interfaces”[10].

2.1 Development interfaces

The development interface is usually a highly customizable tool to display every type of information about the robot.

2.1.1 Visualization

RViz is a widely used 3D visualization tool by roboticists. It comes with the Robot Operating System ROS and provides visualizations for common ROS data types. Twenty three display types are built in with the possibility of developing additional custom visualizations[7]. An example of custom visualizations and tools being developed for RViz are triangle based map meshes[16]. RViz comes both as a graphical user interface (GUI) and as a library. The GUI version of RViz can be used to edit visualization scenes. The limitation of RViz however is that it requires a ROS environment to run. Both the GUI and library version of RViz require a desktop installation of ROS to run[17].

There are some projects bringing ROS capabilities to the web with the Robot Web Tools being one of the more prominent examples. It is a collection of javascript libraries bringing a subset of ROS features to the web browser. It includes a number of libraries with some examples being roslibjs which enables communicating with ROS based robots, rosbag.js to read bag files and ros3djs to visualize ROS data in the browser[18].

While ros3djs offers browser based visualization tools, its capabilities are limited to a subset of what RViz offers. It uses the roslibjs to directly integrate with a ROS installation. It uses the Three.js web graphics library which makes it possible to extend. No GUI editor is provided and the visualizations have to be created programmatically. However the ros3djs library appears to be abandoned as its list of dependencies lists outdated versions of libraries like Three.js and with many people reporting issues getting it to work or it not working with the latest version of ROS2[19].

Foxglove is a robotics visualization platform created to help visually debug robotics. The software comes both as a desktop and a web app. Users can create a dashboard of multiple visualization panels and other tools, the 3D visualization view is just one component of it. ROS bag files can be opened using the software and it can open a live connection to ROS over a websocket connection. The rosbridge suite library allows clients to connect to a ROS computer using a websocket connection over the internet[20]. Some examples of available visualization tools include 3D views, graphs and video feed displays, an example of which can be seen on

figure 2.1 [14]. Foxglove can visualize many of the same data types available in RViz[21], however based on their marketing materials and documentation page, their main target audience seems to be the automotive sector and self-driving cars, although they have some examples robotic manipulators, humanoid robots and quadrupeds[22].

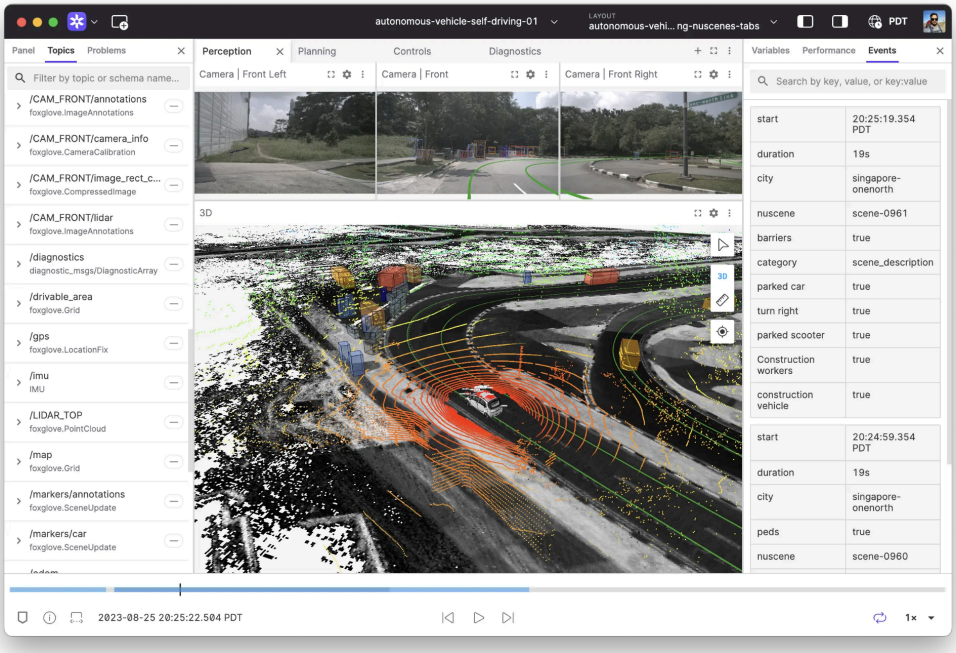


Figure 2.1 the foxglove user interface configured for autonomous driving visualization

2.2 Teleoperation interfaces

Teleoperation is the process of remotely operating a device such as a robot or other machine. It is most commonly used when a robot has to operate in a scenario hazardous or otherwise inaccessible for humans. Common scenario types include disaster zones, underwater environments, tight places and outer space[23–25]. Teleoperation interfaces are split into three different categories by how the robot receives commands from the operator. Direct teleoperation in which the operator directly controls the robot e.g with a joystick; supervisory teleoperation where the operator can only direct, but not control the robot e.g sending coordinates to move to; and multi-modal interfaces where the operators control robots through VR, AR and MR

interfaces. The interfaces are further split into at least two categories by how the operator receives their primary feedback. The first is the egocentric interface, with which the operator receives a video feed from cameras placed on the robot; and endocentric, which provides an overview of the robot in the system with a camera overlooking the robot, a map view or some other third-person perspective[24,25].

For teleoperation specifically, there doesn't seem to exist freely available, equivalent tools like RViz. While RViz itself does provide a way to send teleoperation commands like a 2D goal pose[7], it is not a specialized tool created with the requirements of teleoperation interfaces in mind. There are some commercial solutions which can be adapted to teleoperation like Foxglove, which could be used to create custom teleoperation interfaces. Foxglove allows creating custom dashboards/layouts composed of various widgets[26] and includes both a teleop[27] and map[28] panels. However I could only find one such example from the literature, where the foxglove map panel was used as the interface for an operator to generate waypoints[29]. There have been some other interface examples like 'teleop' by Freedom Robotics, however the company appears to not be active currently and all that remains are screenshots of the application and various tutorials[30]. Additionally ROS itself provides simple utilities, like teleop_twist_keyboard to control a robot with a keyboard[31] or teleop_twist_joy to control a robot with a joystick[32], however using these for teleoperation requires using RViz or a custom GUI solution. As existing tools do not often meet the requirements,custom interfaces are often developed. Figure 2.2. Illustrates some recent examples related to specifically remote teleoperation in space. The common theme among seems to be the use of a video feed with controls clustered around it, with all but one offering a map of the surroundings of the robot. In the case of (c) the Unity game engine was used, making the software look quite a bit different from other teleoperation interfaces and (d) most closely resembles that of Foxglove by the panel based nature[23,33–35].

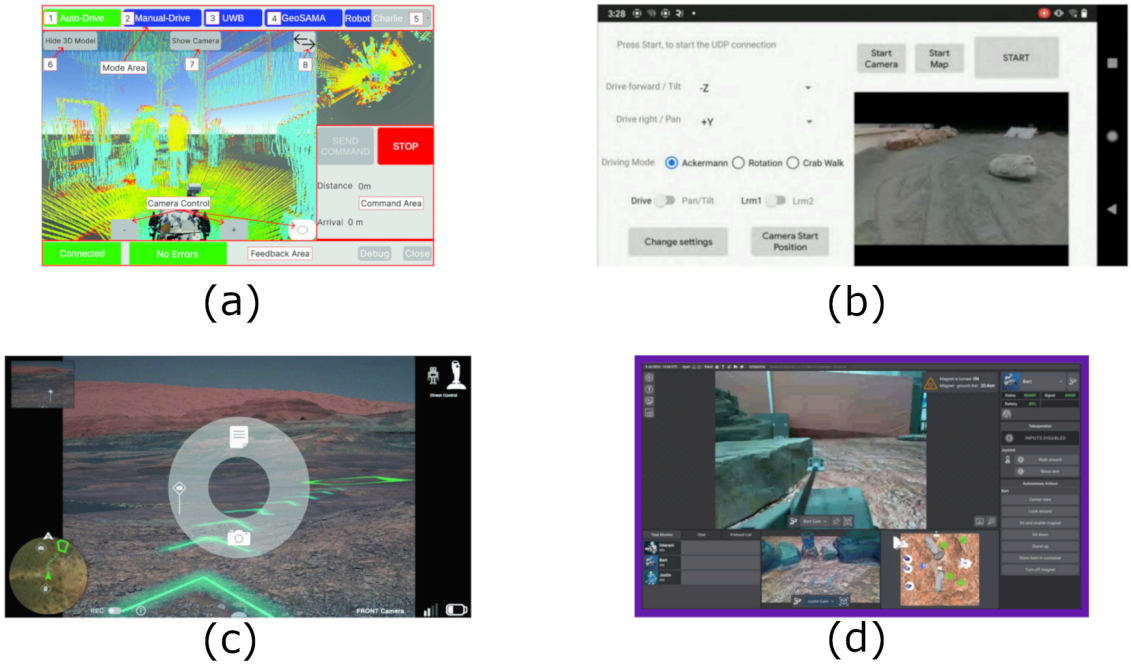


Figure 2.2 recent teleoperation interfaces (a) UI developed for astronauts to control planetary mobile robots[23], (b) smartphone based control interface for an educational lunar rover[33], (c) an augmented interface for rovers[34] and (d) an interface made for astronauts on the ISS to control a quadruped robot on earth[35].

2.3 Public interfaces

Public interfaces are created to show and explain to the public what a robot is doing. However these take extra development time and often require completely different considerations as the viewers can not be assumed to have any specialist knowledge[10]. Public interfaces are often made for science projects like space missions. One such example is the eyes on the solar system app by NASA, which visualizes the solar system, various satellites and space missions by NASA and which can be accessed by anyone with an internet connection shown on the left on figure 2.3. The user can click on various objects in the scene to learn more about them and to view a high-fidelity 3D model of the object[36]. Another recent example of a public interface can be found from the live stream of the Beresheet lunar lander shown on the right on figure 2.4. It

includes a 3D model with informative arrows, showing relative direction of orbital bodies to make the scene more interesting for the viewer[37].

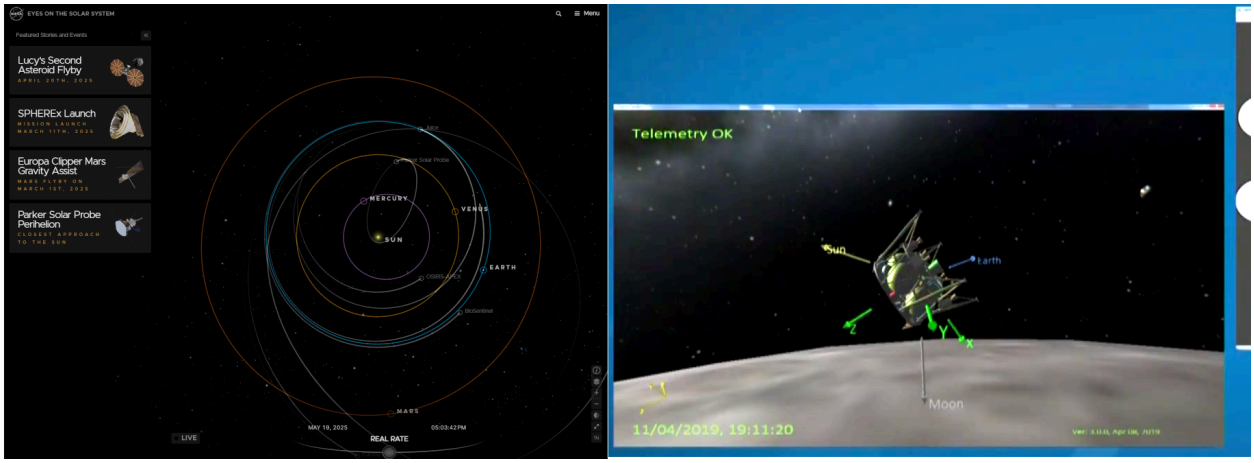


Figure 2.3 Eyes on the solar system on the left and the Beresheet lunar lander explicative interface on the right.

2.4 Analysis of existing tools

The interfaces of two existing ROS visualization tools were analyzed to draw inspiration from.

2.4.1 RViz

RViz comes with 20 different built in display types with some of them illustrated on figure 2.4. These 20 visualizations are connected to ROS topics and only appear when data is available from the topic. The specific topic can be configured from the options of the visualization[17]. ROS topics are named message buses used to send messages in a ROS system[38]. A message is a unit of data communicated over a ROS topic[39]. In addition to that RViz provides visualizations for grids and axes, which don't require a topic to display. From the display type options that RViz provides, of note for this thesis are the various shapes from the marker display type, point cloud, the robot model and uncertainty covariance[17].

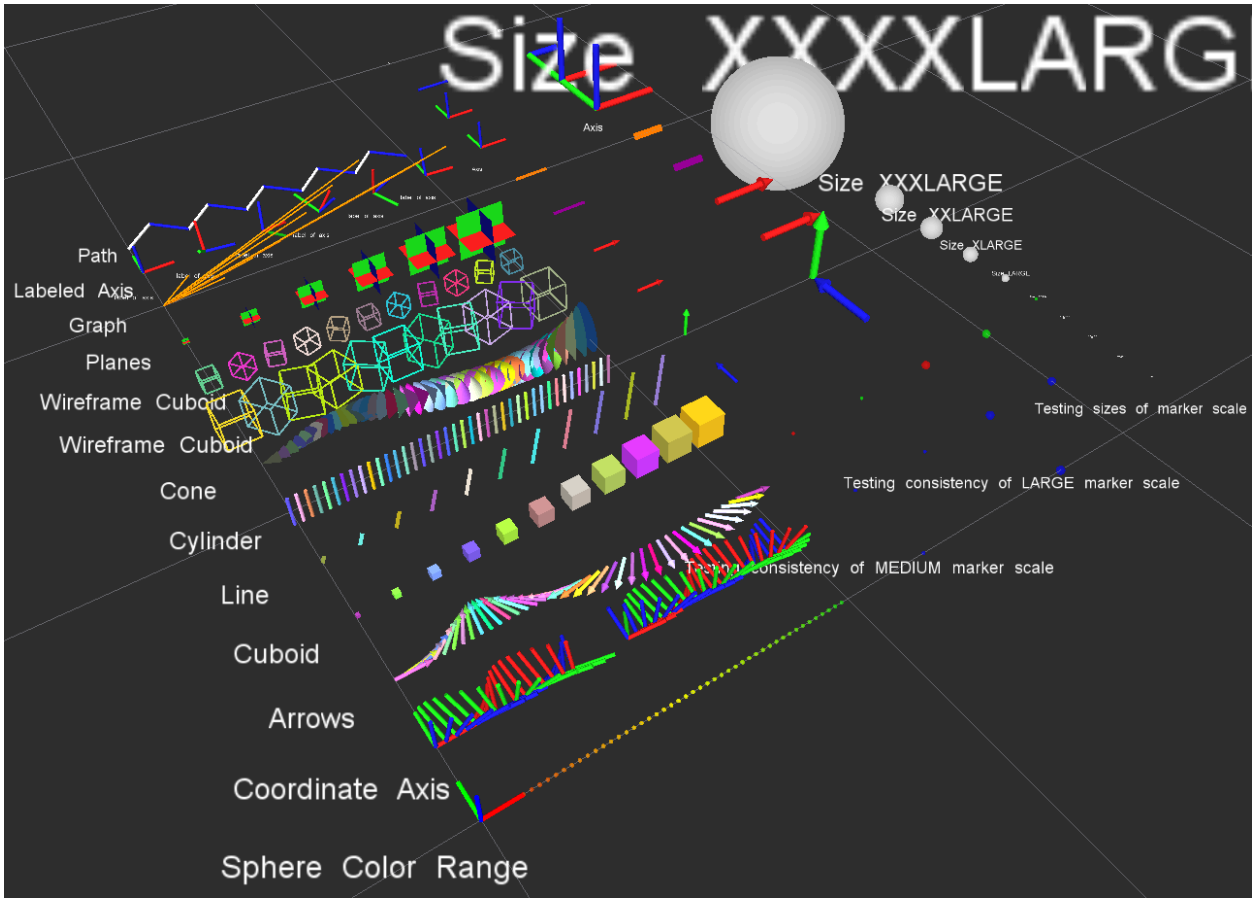


Figure 2.4 some common RViz visualization objects[40]

The user interface of RViz can be seen on figure 2.5. It consists of a 3D viewer in which the user can move the camera using the mouse. The displays panel which at a glance provides an overview of all the objects being visualized. Objects in the displays panel can be expanded from the arrow button and various properties can be edited. Below the objects panel buttons allow the user to add new visualizations, duplicate existing ones, delete existing visualizations or rename them. Besides visualizations, groups can be added to group different visualizations. A time display shows the current time in the ROS system and the wall time, as ROS allows the user to use simulated time instead of the wall time. The views panel displays information of the position of the camera and allows the user to switch between different presets. The plugins panel allows using various tools, like measuring the distance between objects in the 3D scene or sending goal poses to the robot.

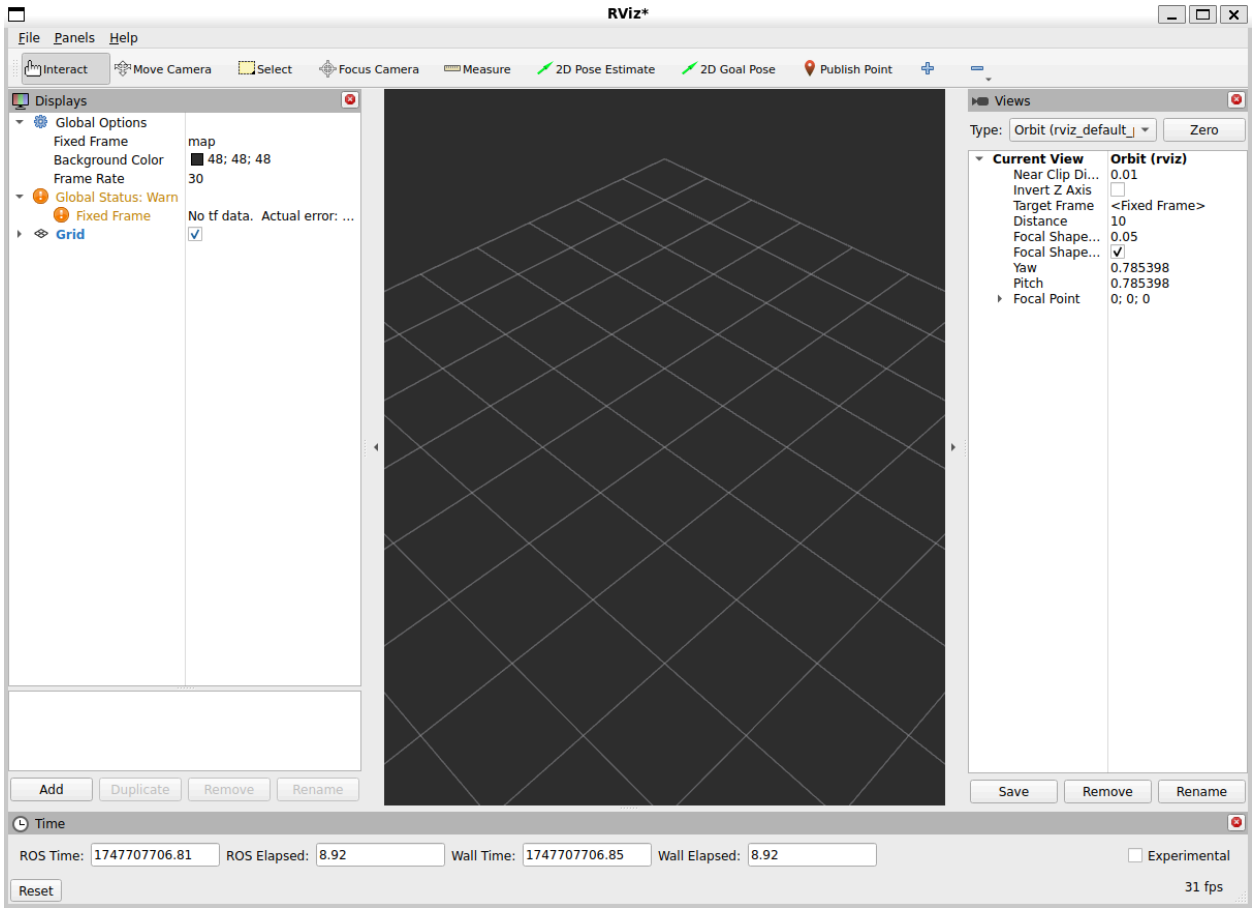


Figure 2.5 the RViz GUI editor

2.4.2 Foxglove

Foxglove has a more complex user interface which the user can fully customize seen on figure 2.6. To switch between layouts the menu bar, located at the top edge of the application, has a dropdown menu which allows the user to open and manage different layouts. Different types of panels can be added, moved around and edited within a layout. The displayed contents and other settings of a panel can be set using the panel tab. The topics tab displays all ROS topics in the visualization and the problems displays a list of errors if any appear. Similarly to RViz, the 3D view can be navigated using the mouse. A difference to RViz however is that the 3D scene has tools to focus on a single object in the scene and to display extra information about it. Similarly to RViz the user can switch between a top down 2D camera and a 3D camera, center the camera on the base link of the scene and to measure distance between two points in the scene. In the

bottom are controls to control ROS bag file playback, with extra controls that allow the user to pause and to change the speed of playback. For visualizations, Foxglove shows all topics available from a data source and allows the user to choose to hide or show specific topics per panel.

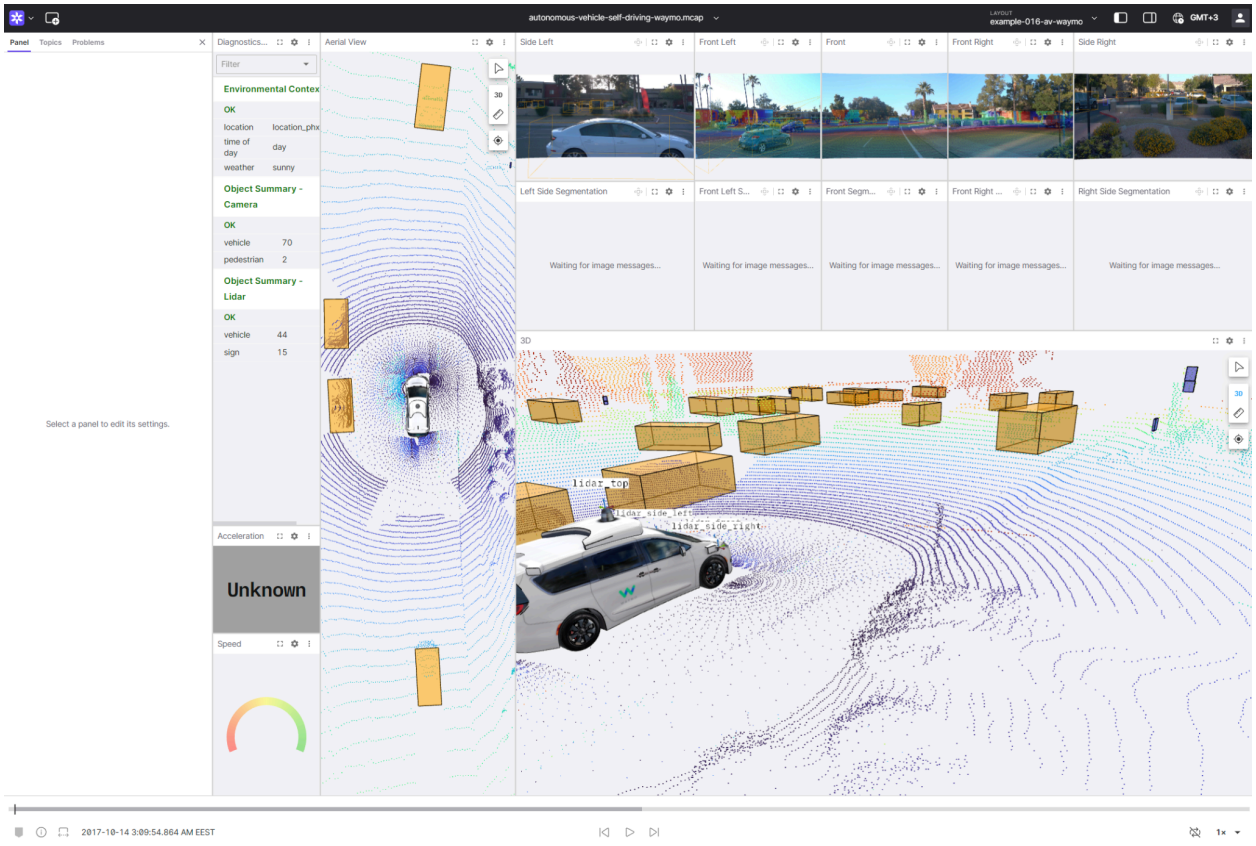


Figure 2.6 the Foxglove user interface

2.5 Analysis of similar software

As 3D editors bear some semblance to visualization tools, interfaces of two of them were used for inspiration as well.

2.5.1 Blender

The Blender 3D creation suite was one source of user interface inspiration. It is a free and open source suite of 3D modeling tools. It features too many tools and modes to list here, but the default layout can be seen from figure 2.7. In the default layout, the central focus is on the 3D

view, with various buttons, menus and widgets surrounding it. To move around the 3D view the user uses the mouse in conjunction with various keyboard shortcuts[41]. The UI features are analyzed more in-depth due to blender having a more complex user interface compared to RViz and Foxglove.

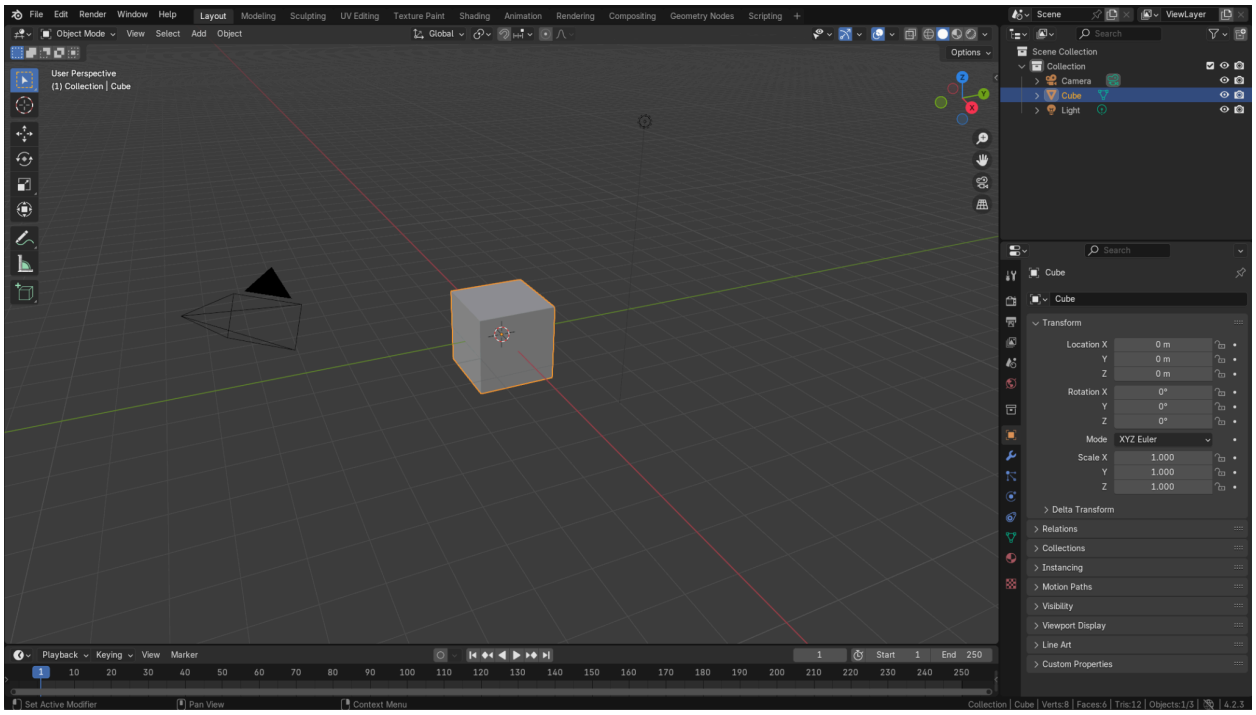


Figure 2.7 The Blender 4.2.3 LTS user interface.

To switch between modes there is a tab bar (outlined in yellow on figure 2.8) which lists the different workspaces such as Modeling, Animation etc. all with their own layouts and tools[41].

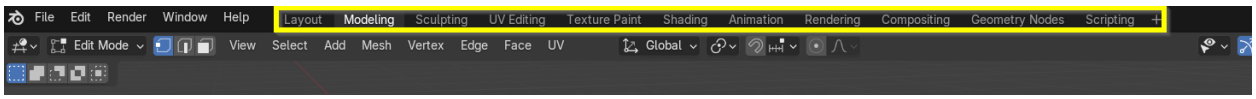


Figure 2.8 Blender layout tab with modeling selected as the active workspace.

The tool menu displayed on the left on figure 2.9 allows the user to switch between different available tools. Since not all tools are useful in every mode, the displayed tools change between the different layouts. The buttons use icons and hover popups to display the tool in case the user does not know what the icon means or what the tool can be.

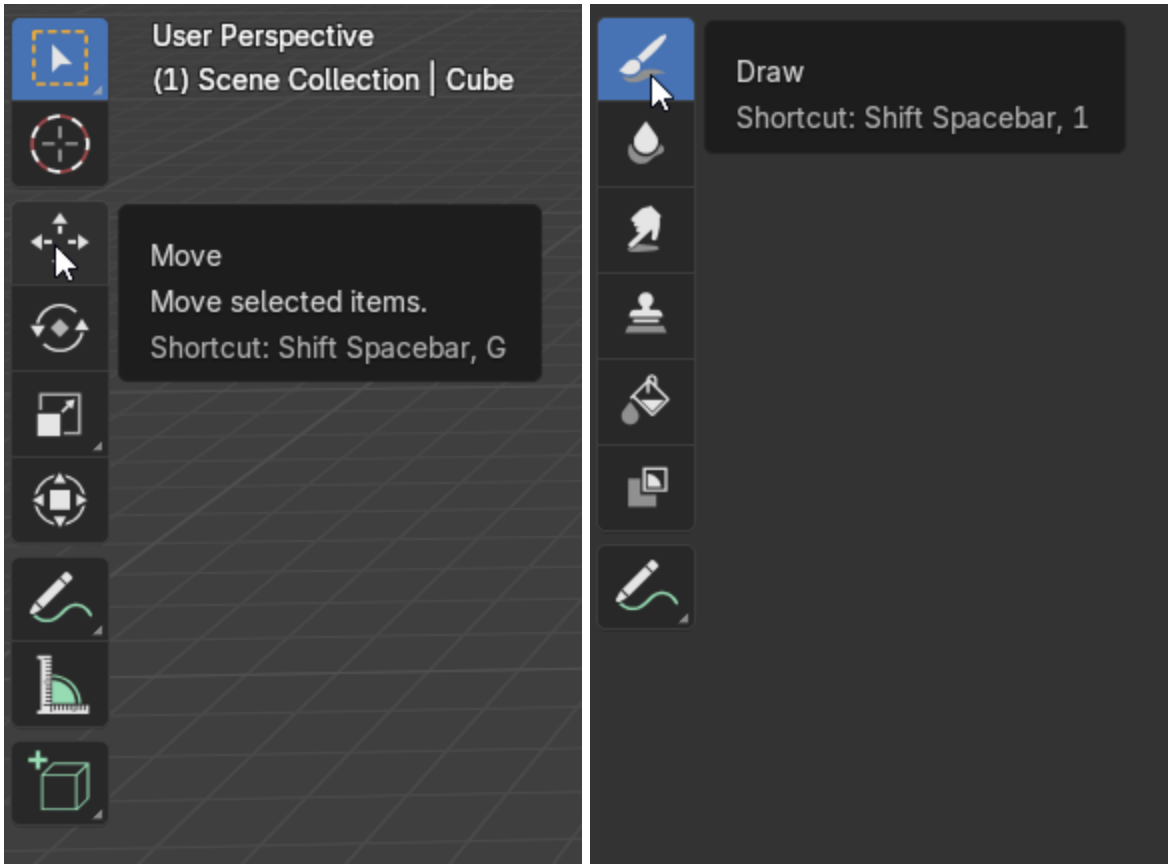


Figure 2.9 The Blender tool menu in Layout and Texture Paint modes.

The scene outline on figure 2.10 lists every object in the scene. Individual objects can be hidden using the eye icon. Objects can be dragged into collections, which allows the user to collapse parts of the outline and to show or hide the grouped objects together[41].

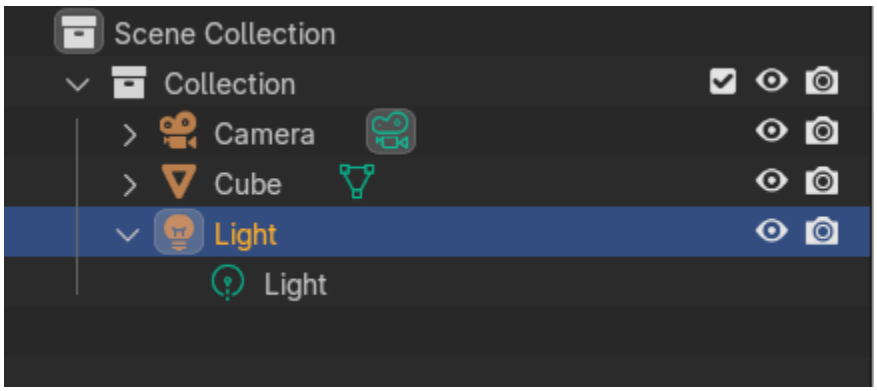


Figure 2.10 The blender scene outline.

2.5.2 Three.js editor

The Three.js editor, shown on figure 2.11 was another of the design inspirations. The UI bears semblance to Blender, however it is just a tool to try out features of the Three.js browser 3D graphics library. The user can use it to compose a simple scene consisting of various primitive shapes, lights and cameras. The scene can be viewed, rendered as an image or exported as JSON formatted data. All objects in the scene are shown in the 3D view and the scene outline. Objects can be selected by clicking on them in the 3D view or selecting them from the outline. Every object has user configurable properties, which can be edited using the properties pane. The properties of the currently selected object are displayed in the properties pane. Additionally a transformation gizmo to translate, rotate or scale the object is shown in the 3D view, centered on the currently selected object. The transformation gizmo type can be changed from the tool select buttons. The mouse can be used to move the camera around in the 3D view. A rotation gizmo can be used to rotate the camera[42].

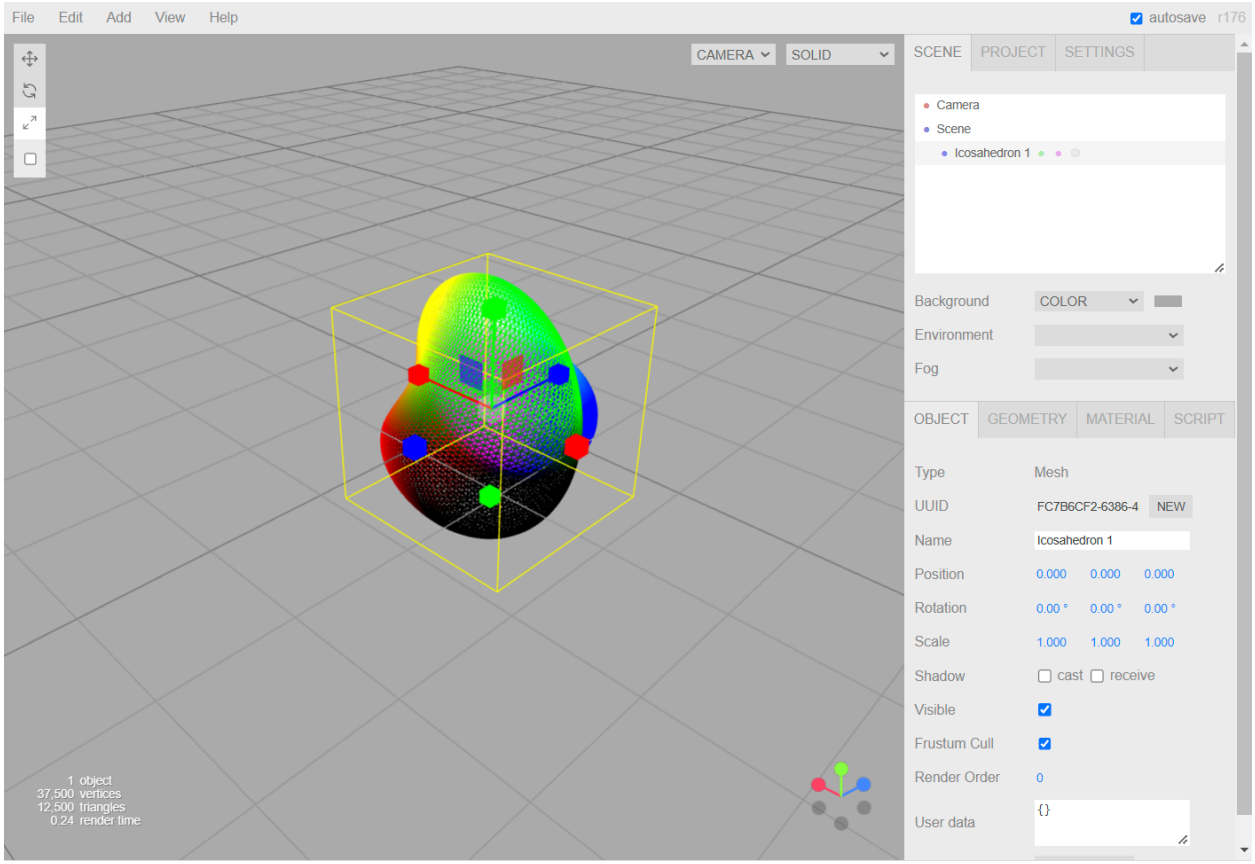


Figure 2.11 The Three.js web editor

2.6 KuupKulgur

KuupKulgur is a small lunar rover being developed as a technology demonstration at the Tartu Observatory of the University of Tartu[43]. The dimensions of the rover are 417mm x 286mm x 239mm. Its software runs on the robot operating system (ROS), either version 1 or 2 depending

on the configuration. The base configuration includes an 6-axis inertial measurement unit (IMU), a camera and wheel encoders. It includes attachment points to test various payloads such as a Light Detection and Ranging (LIDAR) scanner[44].

A web application, titled the KuupKulgur Mission Control Interface (MCI), is used to control the rover, making it possible to control the robot from a device which doesn't have ROS installed. The application incorporates responsive web design to run on smaller screens such as the steam deck. The primary means to control the rover through the MCI is direct teleoperation using the analog sticks of the steam deck and a video feed from the front camera of the rover. To provide debugging information the MCI includes graphs and readouts for the battery level and wheel encoders. Besides direct teleoperation, supervisory control is being explored with there currently being an image or a video feed used as a map from a camera in the ceiling of the Tartu Observatory Space Missions Simulation Center. The map allows clicking on it with the mouse cursor to set navigation waypoints for the rover. The MCI is currently hosted on the KuupKulgur hardware itself and there is no dedicated backend server to host it or any databases related to the web interface. This provides for easy demonstrations and fast connectivity, especially during trade shows and other events where the rover is displayed. However this limits the possible visualization software as no data can be easily persisted especially between multiple rovers. This additionally puts constraints on the visualization component as to not put additional stress on the rover with large file transfers when someone opens the web interface, which could impede teleoperation by using network bandwidth. There are future plans to have a single MCI which supports multi rover control by being hosted on dedicated hardware and not the KuupKulgur rover.

2.7 Lunar environment

The moon is the earth's largest satellite. It is on average a distance of 384 000 km from the earth and has a diameter of 3476 km[45]. Which means that there is an average communication delay of ~1.3 seconds one way and ~2.6 seconds to receive the response. The specific lunar latency of 2.6 seconds has been studied, and found to severely degrade teleoperation performance when using direct teleoperation[46].

Similarly to the earth, the moon goes through cycles of days and nights, however the lunar day and night are each approximately 14 earth days long. Figure 2.12 shows the moon passing through the earth's shadow. These eclipses can last up to several hours depending on which trajectory the moon passes through the shadow, which could be a significant event for a solar powered rover[45]. The moon lacks almost any meaningful atmosphere, with an atmospheric pressure of 0.3 nPa at night[47]. In comparison the earth's atmospheric pressure at sea level is commonly defined as 101400 Pa[48] and martian atmospheric pressure at 636Pa[49]. Due to the very low atmospheric pressure on the moon, there probably won't be any flying lunar drones similar to Mars Ingenuity helicopter[50].

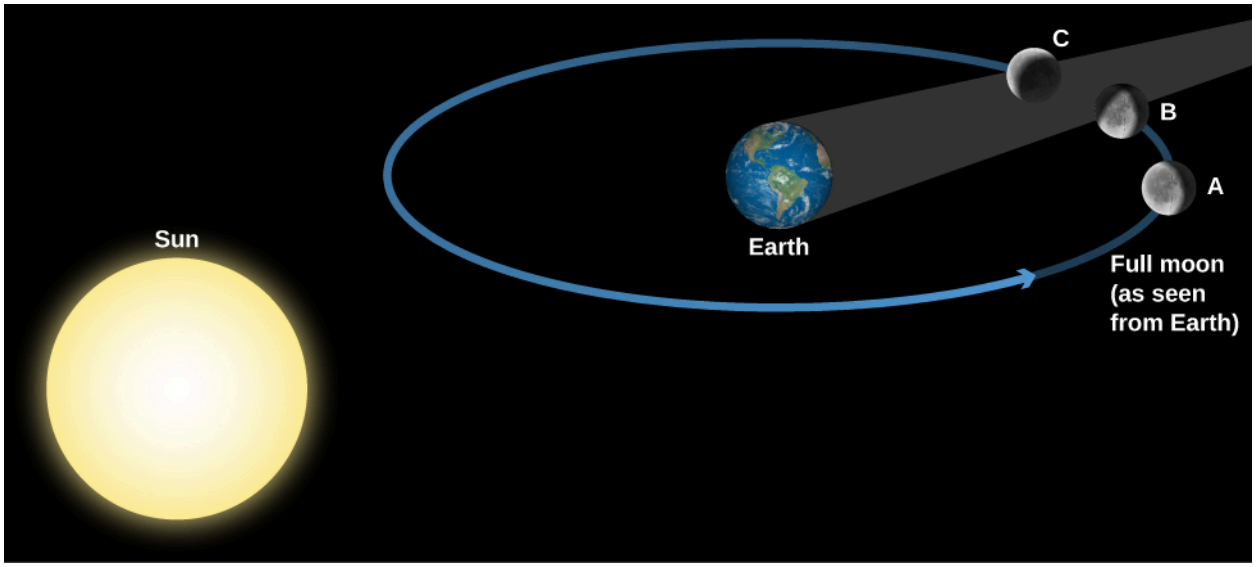


Figure 2.12 solar eclipse on the moon[45]

2.7.1 Future lunar exploration

The Artemis missions are a series of lunar missions to send humans back to the moon. Artemis II is planned for 2026 and aims to send 4 astronauts to orbit around the moon. Artemis III is the planned continuation mission aiming to send 4 astronauts to the lunar south pole on a 30 day mission[1].

Lunanet is a planned lunar internet project by the National Aeronautics and Space Administration. (NASA), European Space Agency (ESA) and Japanese Space Agency

(JSA)[51]. The latest draft version of the specification document also outlines among other things lunar positioning services similar to the global navigation satellite system (GNSS) on earth. It's expect that the lunar south pole will have lunanet service by 2028, with full service becoming available around 2030[52]. This would open up possibilities for getting the positions of and being able to visualize rovers, structures, astronauts etc. on the moon. Whether this information would be made public or not is currently not known.

One interesting feature of the lunar south pole are the geological features casting very long shadows over the surface terrain. Which can be seen from figure 2.13. The NASA scientific visualization studio website includes videos of simulations for the lightning conditions on the south pole up to 31st of December 2030[53]. This would be of interest for a small rover like KuupKulgur, which will most likely use solar panels to charge its batteries.

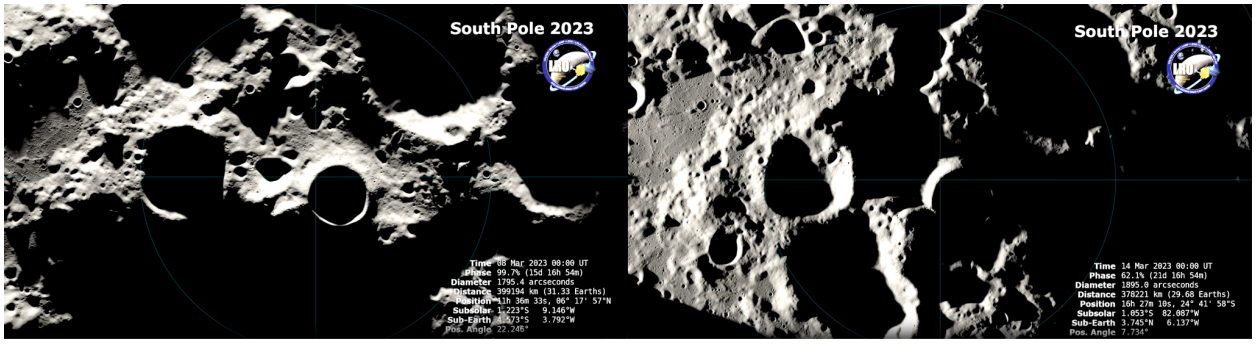


Figure 2.13 south pole illumination[53]

3 Requirements

As KuupKulgur is a ROS based robot with a control interface built using web technologies the solution must be compatible. The main requirements set for the system were.

3.1 Functional Requirements

- The system must be built using web technologies and run within a modern web browser.
- It must be a self-contained web component which can be added to an existing web application.
- It must support visualization of simulation data and lunar environments relevant to rover operations and explicative interfaces.
- The interface must implement support for a subset of ROS visualizations useful for KuupKulgur development.
- The system must expose an API that allows dynamic updates to the visualizations during operation.

3.2 Non-Functional Requirements

- The total file size of the library must be kept minimal to ensure quick loading times (G4.1 Powers of 10 in user interfaces[10,54])
- Asset loading should be optimized to prevent delays in the user seeing the visualization.
- The system must remain performant and responsive under typical operating conditions, without significant slowdowns.

4 Methodology

To meet the goals, a visualization component was developed using the Three.js library for 3D graphics and the Lit library to create various user interface elements. It comes bundled as a javascript library to allow usage in a web app such as the MCI. The component itself allows viewing, interacting with and editing visualization scenes depending on the active mode. The editor comes in two different configurations, the editor and the display. Both are usable as html elements when the library is imported, respectively named <k3d-editor> and <k3d-display>. The display component automatically resizes the Three.js renderer whenever the size of the canvas element, which Three.js is bound to is changed.

The editor component allows the user to switch between the three different modes and to save and load scenes, stored in the browser's localstorage. When the editor component is not used, the implementing application must handle mode selection and scene storage itself. Scenes can be loaded using the scene manager singleton service.

The display component allows the user to interact with the scene depending on the chosen mode. The user can move around the scene using the mouse. Left click will rotate around the current orbit point of the camera. Right click will pan the camera and the scroll wheel allows zooming the camera. The default tool is the select tool. The selected object is highlighted by the outline of a yellow box in the 3D scene and colored blue in the scene outline. Selecting the same object again will un-select it. From the tool menu the camera orbit tool visualizes the orbit point of the camera, which can also be changed in this mode by ctrl-clicking to change the orbit point of the camera.

In edit mode the user can add visualization objects to the scene using the add menu. Objects can be selected by clicking on them in the 3D scene or from the scene outline. The scene outline allows dragging and dropping objects to reorder or nest them. In the 3D view the objects can be moved around using the transform gizmo or various other properties changed using the properties menu. The active mode of the transform gizmo can be changed between translate, rotate and scale from the tool select menu. Changes to an object's properties can be undone or

redone using the edit menu and any object can be removed using the delete option in the edit menu. Figure 4.1 illustrates a cube being selected with the translation gizmo as the currently active tool.

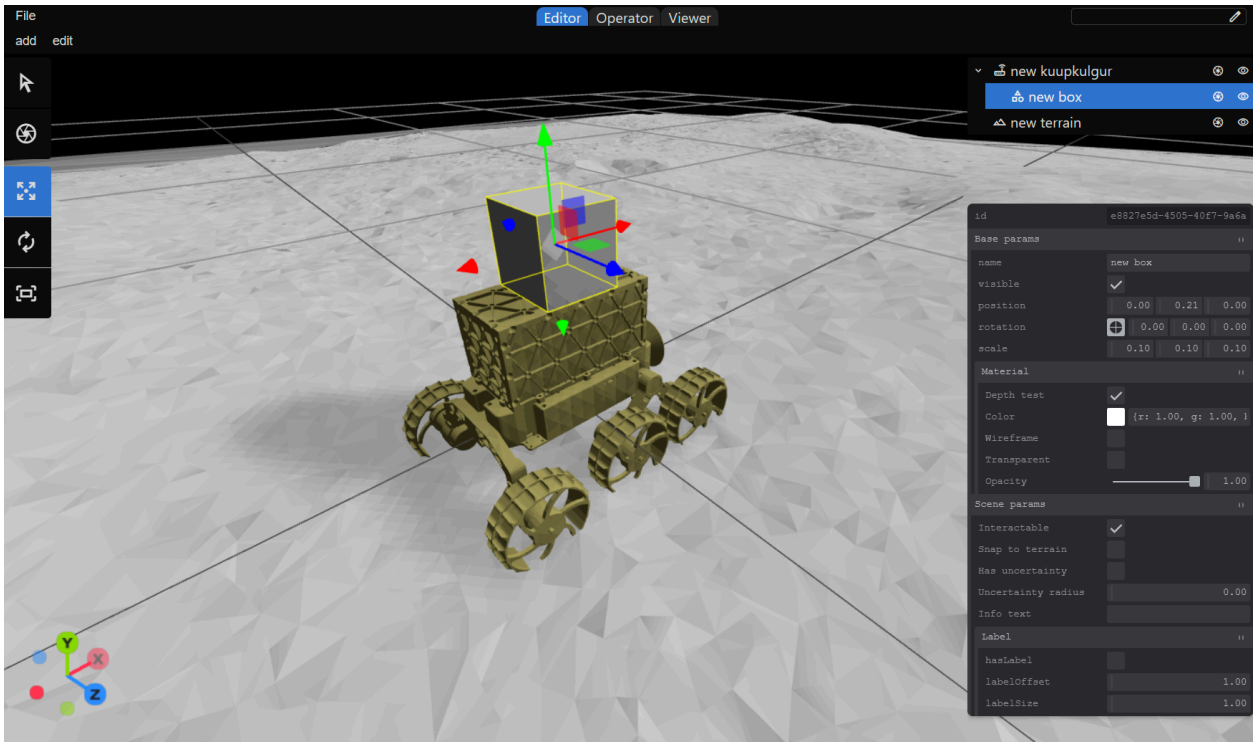
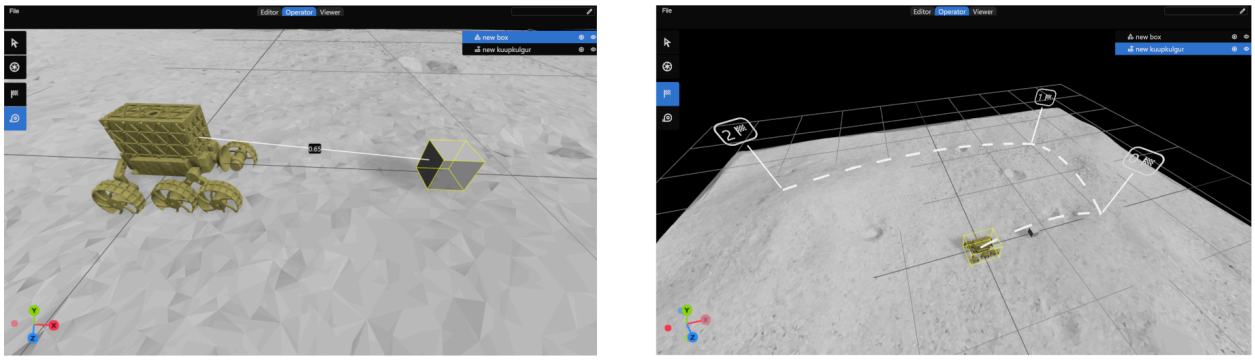


Figure 4.1 visualizer editor mode

In the operator mode the user can use the measure tool to measure the distance between two points in the scene and the goal tool to manage the goals of the currently selected object. To use the measure tool (a) on figure 4.2, the user must hold down the ctrl click while left clicking and continue to hold down the mouse button until the end of the measurement. To use the goal tool (b) on figure 4.2 an object in the scene must be selected. To add a new goal the ctrl key must be held down while clicking on a terrain object. To remove a goal, the shift key must be held down while clicking on a goal marker.



(a) (b)

Figure 4.2 operator mode tools in use.

Some aspects of the component can be configured using css variables. The configurable properties are border radius, background-color, text color and tooltip animation settings. The list of properties is provided in the implementation section below. The colors used in the thesis follow a limited color-palette, aligning with the guidelines G3.4.1 Less is more and G3.4.2 Reserve red, green for “bad” and “good” from the thirty-two guidelines for HRI[10].

The architecture divides the main components of the software into components, services and controllers. To simplify accessing data all data sources are services which are defined as singleton classes, initiated when the library is loaded. The scenes service manages loading and saving scenes from the browser’s local storage. Scene service manages all visualization specific objects in the scene and notifies any consumers about changes to the objects. For example the scene outline component listens to any changes in the scene hierarchy to update itself and notifies the user clicking on the buttons the object visibility toggle or the focus camera buttons. The scene controller handles setting up and managing user interaction and rendering for the scene. Goals and labels are managed by controller classes which handle adding, removing and updating their internal objects, separate and hidden from the overview that the user has access to.

4.1 Choice of technologies

To develop the visualization component and to test it several different technologies were used. The main technologies and the choice of which is briefly explained in this section.

4.1.1 Typescript

Typescript is a strongly typed programming language built by microsoft. It allows using all features of javascript while adding features of its own, like the type system. It was chosen due to the type safety features it has over plain javascript. Compared to javascript, typescript needs to be compiled before it can be used[55].

4.1.2 Three.js

Three.js is a lightweight 3D library which works across all major browsers. Three.js provides extensive api documentation, a lot of code examples and many features one would require to build a browser based 3D application[56]. It was chosen due to the author's previous experience with it.

4.1.3 Lit

Lit is a library implementing the web components model for creating custom html elements. It includes a number of features to simplify creating custom components. One of the features being declarative templating, which allows using html markup with additional features like loops and conditions added to the html markup. Another important feature is reactive properties, which track their values and cause the component to automatically update upon any changes[57]. The Lit library was chosen due to the aforementioned features, with the web component model being the primary decider. From the browsers and users point of view, web components are regular html elements which can be added to any html and the browser will handle instantiating them. An additional feature which web components provide is style isolation, which makes sure any style sheets from the outside do not reach inside the web component[58]. These features provide for easy integration with the MCI as no additional application frameworks need to be supported.

4.1.4 Vite

Vite is a highly configurable build tool which includes a built in dev server and typescript compilation. It was chosen due to its simple integration with typescript and its ability to be easily configured to output a javascript library. The dev server provides debugging support, automatically recompiles code on any changes to files and refreshes the browser[59].

4.1.5 QGIS

QGIS is a free and open source geographical information system software. It can parse specialized geoinformatics file formats such as geotiff, which is often used for topographical information[60]. QGIS was used to convert terrain height maps into 3D models.

4.1.6 Blender

Blender is a free and open source 3D creation suite which allows editing 3D models[41]. Blender was used to process the terrain models to use in visualization scenes.

4.2 Implemented visualizations

A subset of RViz visualizations were implemented and some custom ones were developed. RViz and ROS specific terminology was avoided where possible, to not give any false ideas about the component being an equivalent for RViz or being able to directly connect to ROS.

Several properties can be changed about every visualization object. Not every modifiable Three.js property is exposed to not over-complicate the user interface. The basic properties of name, visibility, position, rotation and scale can be changed for every object. Objects with a material allow turning off depth-test, setting color, showing the object as a wireframe or changing the opacity. Turning off the depth-dest makes the object render regardless of being occluded by any other object in the scene. This can be a useful feature to show debug info and other overlays which need to be drawn through e.g the geometry of a robot model. For points the user can set the point size and generate test points, arrows allow setting a tracking target and grid size and the

number of divisions can be configured. On the object properties pane, these properties are all under the base params folder, which can be expanded or collapsed.

The custom properties, titled ‘scene params’ allow turning off interactability, enabling snapping to terrain, and enabling and setting the uncertainty radius, including an info text and enabling the name label and setting its position and offset. Objects with interactability turned off do not appear selectable in operator and viewer modes. All nested objects inherit the interactability value of their parent.

4.2.1 Group

The group was implemented to allow grouping multiple objects. It doesn’t visualize anything on its own. It’s expected that groups are either used to group different primitives together into more complex geometries or as a ‘folder’ to group multiple moving objects to enable showing-hiding them together or to organize the scene overview.

4.2.2 Primitives

The box, sphere, cylinder and cone were chosen as these are simple shapes which can be used as placeholders or made into more complex shapes. These are also shapes that RViz can visualize using the marker display type. Besides these four primitives, Three.js provides a number of other predefined shapes like: torus, ring, knot, dodecahedron etc. which were not included to not over complicate the interface with too many choices.

4.2.3 Misc

To visualize vectors the arrow visualization was added. Motivated by the public explicative interface example of the Beresheet lunar lander, the arrow visualization can be configured to track another object in the scene as illustrated in figure 4.3. This allows creating interesting visualizations for the explicative interface. It’s possible to combine the arrow with primitives to create more advanced visualizations e.g nesting a cone inside the arrow makes it possible to visualize something like the range display type in rviz by changing the orientation and scale of the arrow.

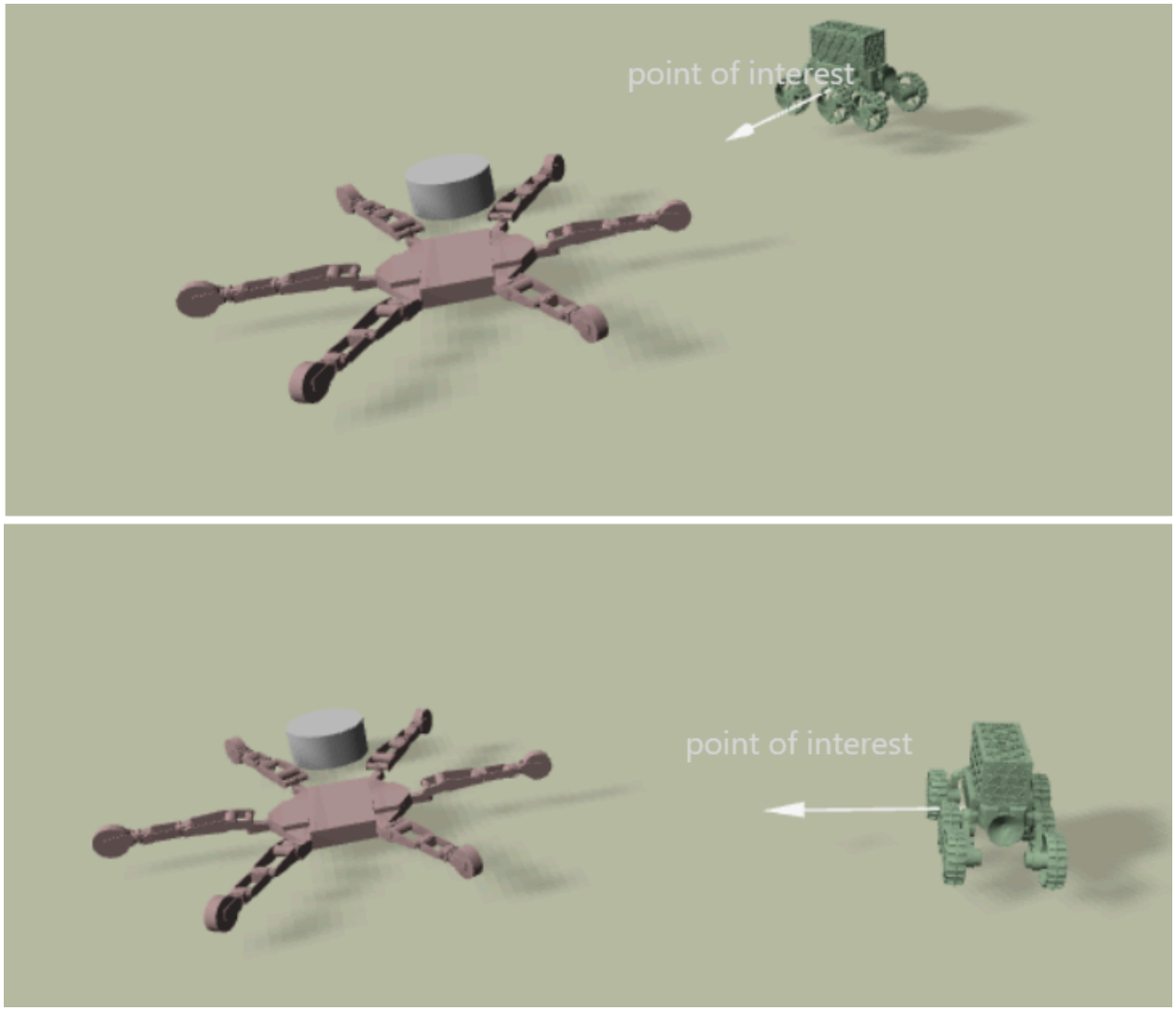


Figure 4.3 arrow visualization with an arrow pointing at the Athlete[61] robot model

Grids and axes are provided as helpers to visualize distance and orientation. For grids the number of divisions and the size of the grid can be configured.

4.2.4 Points

To visualize LIDAR or other point cloud data the point cloud visualization type was added. The size of the points and the color of the points can be set. Currently only a single color is supported for all points. Unlike other visualizations the point cloud does not support a static configuration which gets automatically loaded with the scene. The point cloud needs to be updated with point

data after loading the scene. For testing a set of randomly generated test points with the ‘generate random points’ button on the object properties pane.

4.2.5 Robot model

The robot model is a display type adapted from ROS which can display different URDF models. The urdf-loaders library is used to load the models. For development the KuupKulgur(figure 4.4 on the left) and the Athlete[61](figure 4.4 on the right) URDF models were tested, however any model could be added as long as it can be loaded using an url. The urls and assets need to be predefined in code. The user interface dynamically adds buttons for all defined assets to the editor add menu. While URDF supports textures, this isn’t used and a single color is set for the whole model instead as the KuupKulgur URDF model did not include any textures.

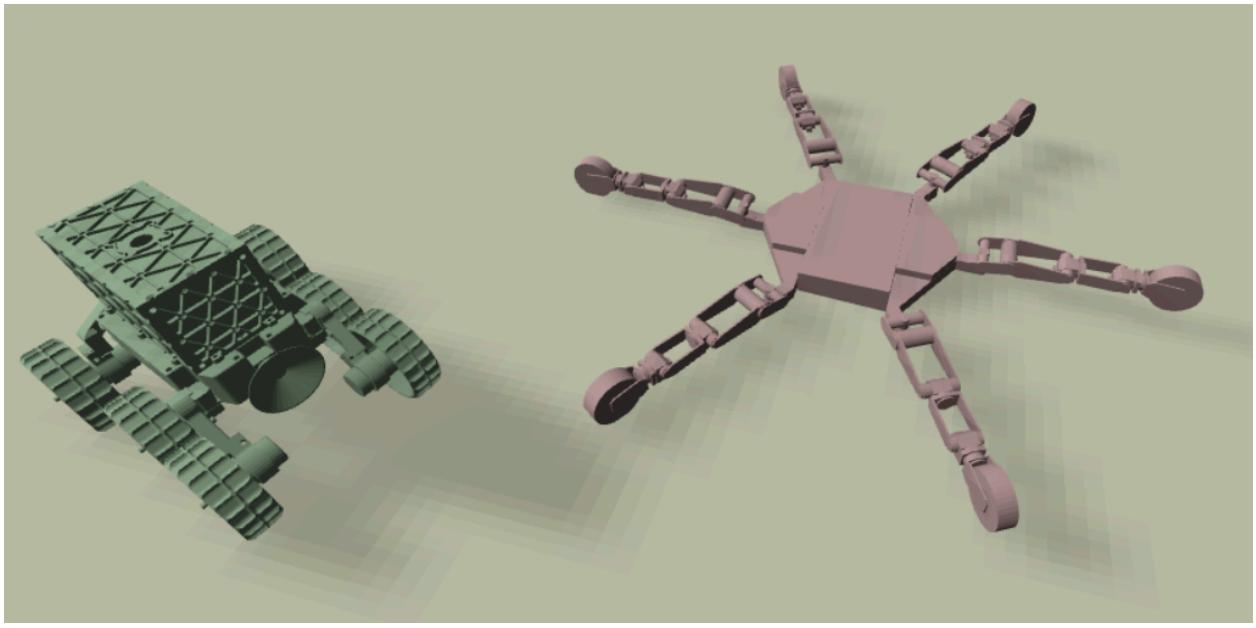


Figure 4.4 two robot models with KuupKulgur on the left and Athlete[61] on the right

4.2.6 Terrain

The terrain object is a special visualization type developed to be used with scenes created for teleoperation and the public. RViz and Foxglove visualizations often use either 2D maps or point clouds. Both of these are hard to understand for roboticists. To solve this apparent issue the

terrain visualization was created to visualize topographical info as a 3D model. As part of this thesis two different lunar terrain models were used, one of the lunar analogue environment at the Tartu Observatory and another from freely available lunar topographical scans from NASA. To enable testing all scenarios a plane is provided to be used as a featureless terrain for cases when there is no model available. Besides just visualizing terrain to avoid confusion by users, any objects in the scene can be configured to snap to the surface of the terrain when updated. This feature allows the scene to look more seamless and avoid any floating objects. When configured, a raycast is performed from above to below of the snapping object to find the intersection point with the terrain. The object's bounding box is considered to place the object on-top of the terrain. In the first implementation raycasting was found to severely reduce performance due to the large number of polygons in the terrain mesh. The three-mesh-bvh library was used to solve this problem and speed up raycasts. The library allows splitting models up into bounding volume hierarchies, which simplify searching for the intersection point of the raycast[62].

4.2.7 Position uncertainty

The position uncertainty is a custom visualization created based on the expected needs that a teleoperator might have when operating a robot on the lunar surface. Lunanet promises positioning services for rovers on the surface, however the availability and high precision are not guaranteed[52]. To account for the possibility the position uncertainty visualization was developed. In ROS uncertainty is usually visualized as an ellipse or an ellipsoid to account for differences between the axes[17]. In this case a circle was chosen to make it more easily understandable and comparable. A green circle is drawn on any terrain visualization which lies directly below the object with uncertain position. A custom shader was developed to achieve this goal. Other options like semi-transparent volumes were considered, however overlapping multiple volumes made the 3D scene very crowded and hard to read. Height info is discarded as objects flying close to the lunar surface for prolonged periods of time are not considered. In the future there might be a need for more advanced versions of this visualization to account for multi-level structures and caves on the moon. The case when two or more uncertainties overlap can be seen on figure 4.5 in which the overlapping area is drawn in pink to signal to the operator

about potential danger.

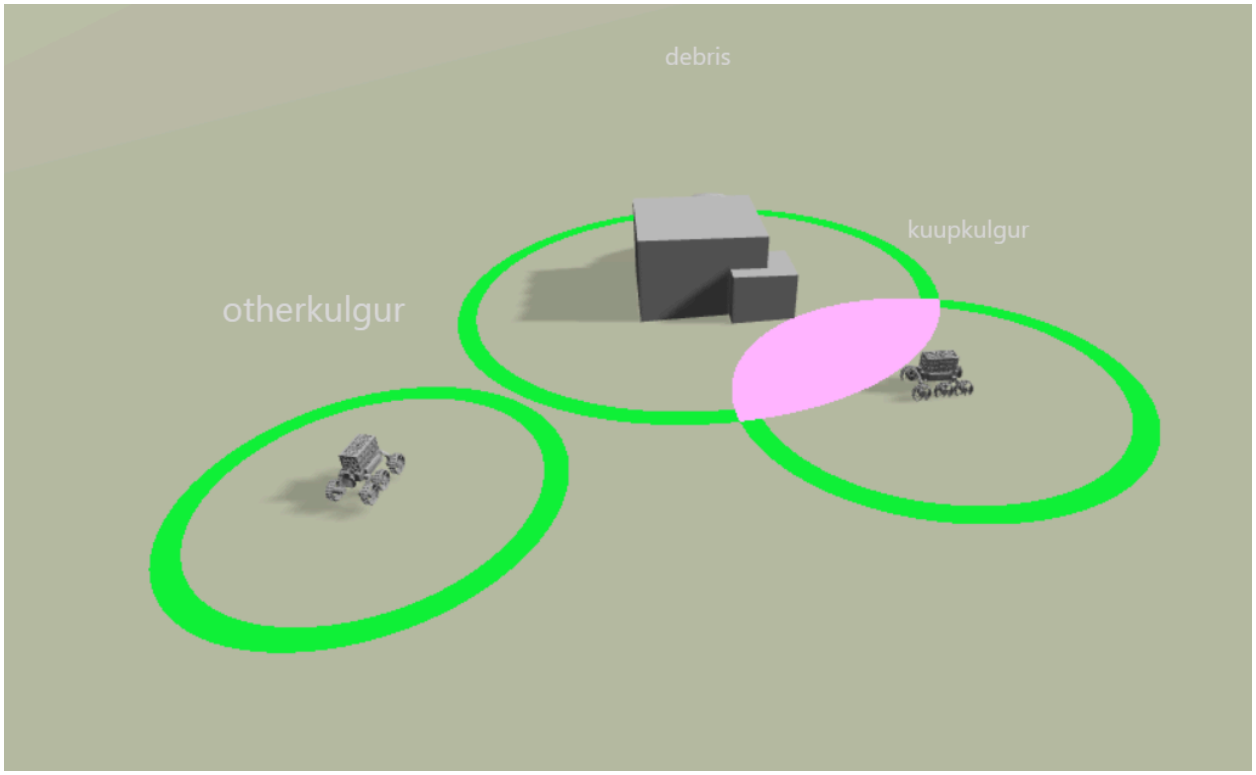


Figure 4.5 position uncertainty on terrain

4.2.8 Label

Labels can be added to any object in the scene to display the name of that object. The size and the offset of the label can be configured. The size of the text remains constant regardless how far the user is from the object as illustrated on figure 4.6.

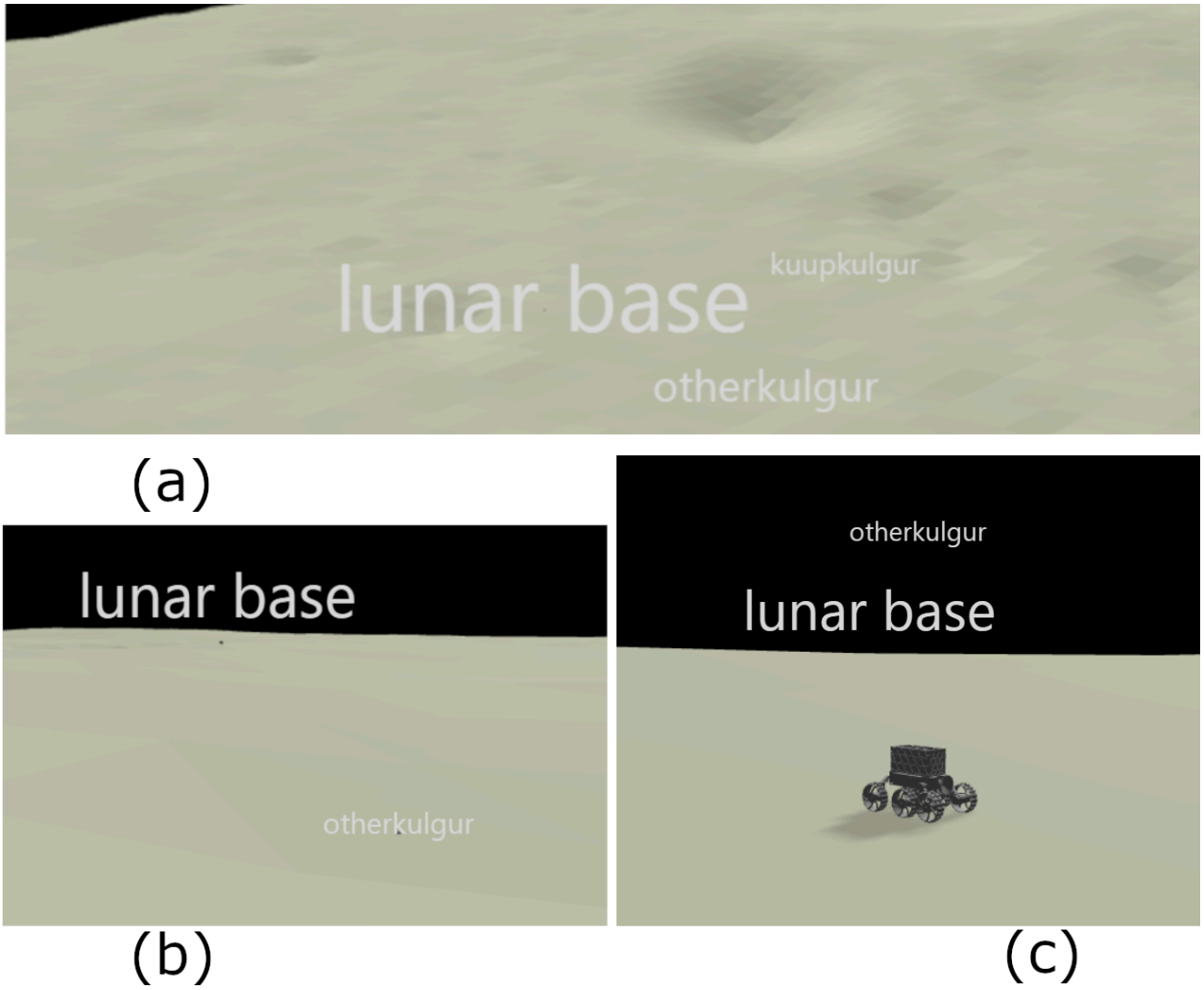


Figure 4.6 object labels at three levels of zoom (a) very zoomed out, (b) medium distance to a rover and (c) right behind a rover, with the lunar base object being hidden by distant terrain.

4.2.9 Goal

Goals are used to visualize movement waypoints. A dotted line is drawn from the object that is the 'target' of the goal and the goal itself. A dotted line is also drawn between consecutive goals. Figure 4.7 illustrates nearby and distant goals. As the labels can't be configured to disable their labels they were made to account for distance i.e unlike object name labels goal labels do not remain at constant size on the screen. The labels automatically rotate towards the user's camera to avoid the user having to rotate the camera to read the goal number. Any object in the scene can

have a single series of goals assigned to it. What the goals actually do is up to the containing application to implement.

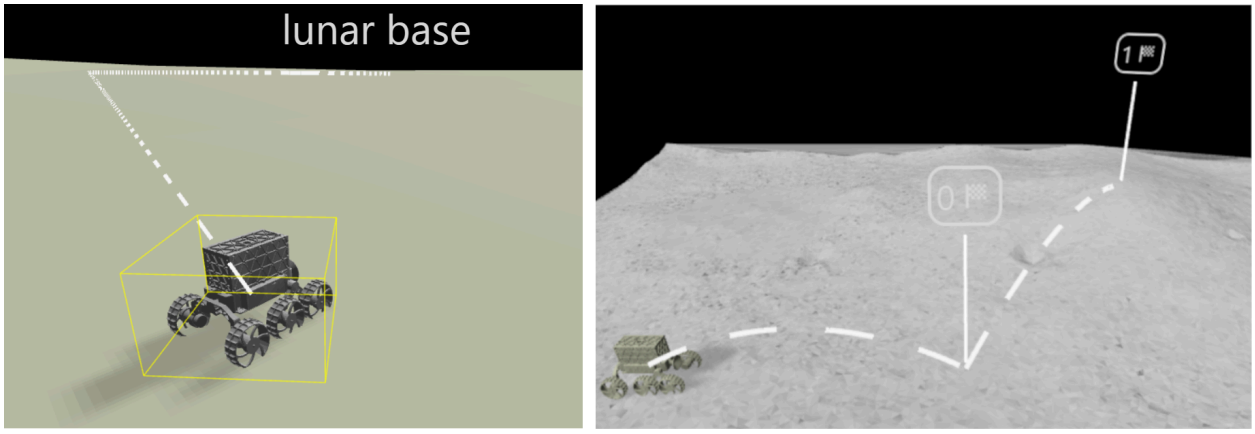


Figure 4.7 different goal scenarios in a large and in a small scene.

4.3 Lunar visualization limitations

To visualize the lunar terrain at high resolution can require very large 3D model files. It is possible to load larger model files in the browser, however the user experience would suffer. Downloading e.g a 100MB 3D model file, when opening a web app, could take considerable time depending on the connection speed and put unwanted strain on the server. The last point being especially important as the server currently runs on the rover hardware. No local caching/storage solution or use of a specialized file hosting service for the backend, for large files, was implemented as part of this thesis.

The moon goes through a day-night cycle, has long shadows cast by craters and mountains and a solar eclipse is also possible on the moon. Due to technical limitations of Three.js it is not feasible to visualize accurate shadows. First loading enough terrain geometry at high enough detail to accurately visualize the shadows is not feasible due to large 3D model file sizes required, but secondly Three.js uses the shadow mapping technique to generate shadows. Shadow mapping renders information to generate shadows onto separate texture(s). In simple terms whatever the light sees is illuminated and what it doesn't see is in the shadow. To achieve this, the scene is first rendered from the point of view of the light(s). When the user's point of view is rendered, the shadow map is sampled to find out by how much each rendered pixel is

illuminated. To cover large areas at sufficient levels of shadow detail the shadow maps need to be scaled up[63]. Using a shadow map which is too small, ends up producing very dim shadows or possibly even no shadows, as the shadow information gets averaged out. For Three.js the default suggested shadow map size is 512px by 512px[64]. Three.js comes with an add-on for cascading shadow maps (CSM) which was used in this thesis. CSM improves upon regular shadow maps by generating multiple shadow maps at different levels of detail, based on the distance to the camera. This allows displaying shadows for larger scenes, however the Three.js CSM add-on still does not provide infinite distance shadows. An additional source of shadow on the moon are lunar eclipses. The eclipse times on the moon could be calculated with the astronomy-engine library, available for javascript and typescript[65]. Based on that the light conditions could be simulated by controlling the brightness of the light source(s) in the scene, but this seems superfluous as other, more common types of shadows are not implemented. In conclusion, due to it not being feasible to load large models of lunar terrain and how shadows are implemented in Three.js light and shadow is only used to illuminate the scene and to give a sense of depth for the user. Accurate lunar light conditions are not replicated.

4.5 Integrating the component in a web app

To integrate the component, two javascript library files are available. One of the files in the optimized UMD format and the other in the ESM format. When the bundler is used, as is the case with the MCI, the UMD file is preferred as it comes minified and optimized by the vite bundler.
To use the component the editor or the viewer, the respective html element must be added to a html file or instantiated through code. The assets must be defined using the AssetService singleton. An example loading scenario can be seen in listing 1.

```
AssetService.instance.add({name: 'bunker', url: 'bunker.obj', type: 'terrain'});
AssetService.instance.add({name: 'lunar', url: 'npd.obj', type: 'terrain'});
AssetService.instance.add({name: 'kuupkulgur', url:
'/kuupkulgur_descriptions/urdf/kuupkulgur_descriptions.urdf', packages:
{"kuupkulgur_descriptions": "/kuupkulgur_descriptions/"}, type: 'urdf'});
```

Listing 1. Example instantiation of assets.
To enable styling the google icon font must be added to a root stylesheet and css variables set to style the component as described in listing 2.

```
@import
url('https://fonts.googleapis.com/css2?family=Material+Symbols+Outlined&display=swap');
```

```
:root {
  --k3d-border-radius: 4px;
  --k3d-border-radius-alt: 8px;
  --k3d-primary-bg: hsl(0 0 5%);
  --k3d-primary-bg-raised: hsl(0 0 15%);
  --k3d-primary-bg-highlight: hsl(0 0 25%);
  --k3d-primary-bg-active: hsl(214, 63%, 50%);
  --k3d-primary-bg-active-highlight: hsl(214, 63%, 60%);
  --k3d-primary-bg-active-alt: hsl(153.9, 63.1%, 50%);
  --k3d-primary-bg-active-alt-highlight: hsl(153.9, 63.1%, 60%);
  --k3d-primary-text: hsl(0 0 85%);
  --k3d-primary-text-dim: hsl(0 0 65%);
  --k3d-primary-text-disabled: hsl(0 0 45%);
  --k3d-primary-text-highlight: hsl(0 0 95%);
  --k3d-tooltip-appear-delay: 0.5s;
  --k3d-animation-duration: 150ms;
  --k3d-animation-easing: ease-in-out;
}
```

Listing 2. Example setting up the required css styles.

If the editor is not used, the sceneservice must be used to load a scene and the sceneMode parameter set for the display element, the allowed values are: 'view', 'operate' and 'edit'.

5 Testing and discussion

To evaluate whether the software performs to task, several tests were set up. The tests were conducted in a simulated environment. A pre-recorded csv file was used to test updating from non-ROS data sources. A ROS bag file containing lidar point cloud data was used to test points visualization. The turtlebot simulator, available in ROS, was used to test the operator interface.

5.1 Setting up the Lunar analogue environment visualization

The 3D model for the Lunar analogue environment terrain visualization was created from a LIDAR scan provided for this thesis. The scan covered an area approximately 8 by 8 meters. All movable objects such as rocks were not part of the scan. The original file was an obj format 3D model file with the size of 532MB. However as this file is too large to download over the network in a reliable manner it was downsized. As a test, an attempt was made to load the original file using the Three.js OBJLoader utility, however the model did not load nor produce any errors. The model was processed with the Blender 3D modelling software by setting the geometry to the origin to center of the scene and then using the decimate modifier. As the surface of the model became very noisy after decimation, the laplacian smooth modifier was also used. For the decimate modifier the active mode was collapse, symmetry was disabled, triangulation was enabled and the decimation ratio varying between the downsizing attempts. The laplacian smooth modifier was used with the following settings of repeat: 10, all axes selected, lambda factor: 0.1, lambda border: 0.1, preserve volume enabled and normalized enabled. After reducing the model with a ratio of 0.1, it was possible to load the model with Three.js. However to make loading the file even faster, it was further reduced down to 7.70MB using a decimate factor of 0.01 in Blender. A decimate factor of 0.001 was also tested however it eliminated all interesting geometry and was not selected for use.

The scene was set up with a terrain component, using the prepared 3D model; an appropriately scaled box to represent a pillar inside the simulation environment and a single URDF model of KuupKulgur.

5.2 Setting up the lunar visualization

The 3D model for the lunar terrain visualization was created from a height map retrieved from the high resolution LOLA topography for lunar south pole sites. The LOLA data includes sites believed to have high lunar exploration potential. [66,67]. The site designated NPD (Malapert crater), shown on the upper central area on figure 5.1, was chosen due to having relatively flat terrain and having a medium amount of sunlight. This could possibly be a site for KuupKulgur as the rover is small and would have difficult time traversing more sloped terrains and would need some sunlight to charge batteries using solar panels.

To generate the 3D model the NPD LDEM: NPD_final_adj_5mpp_surf.tif file. As the file is a special geotiff format it was imported into QGIS as a raster layer. The pixel size of the geotiff was found from the raster layer properties to be 4000x4000 pixels. Then a 3D map view was added. The 3D map view was configured to use DEM (Raster layer), the geotiff layer as the elevation map, vertical scale was set to 1.00, tile resolution to 4000px, skirt height to 10.0 map units and offset to 0. At first the 3D view was exported as a 3D scene with a terrain resolution of 128. The output file was 4MB in size however upon inspection the model was very low resolution. The scene was then exported at a resolution of 1024 which resulted in a higher quality 3D model with a file size of 176MB. Based on setting up the Lunar analogue environment terrain model, sizing this model was attempted however the model lost too much detail considering that the model will be a lot larger in extent than the smaller Lunar analogue environment terrain scan. Instead the model was loaded into Blender, centered in the scene, rotated to face the Z axis and scaled up 20000 times as QGIS had normalized the model. The range was retrieved from data in the geotiff file. A 2km by 2km cube was made in blender originating at the center of scene and a boolean modifier was applied to the terrain with the cube as the operand object and intersect as the selected mode. The final result was exported as an obj file and found to have the size of 2.11MB which was deemed to be appropriately small.

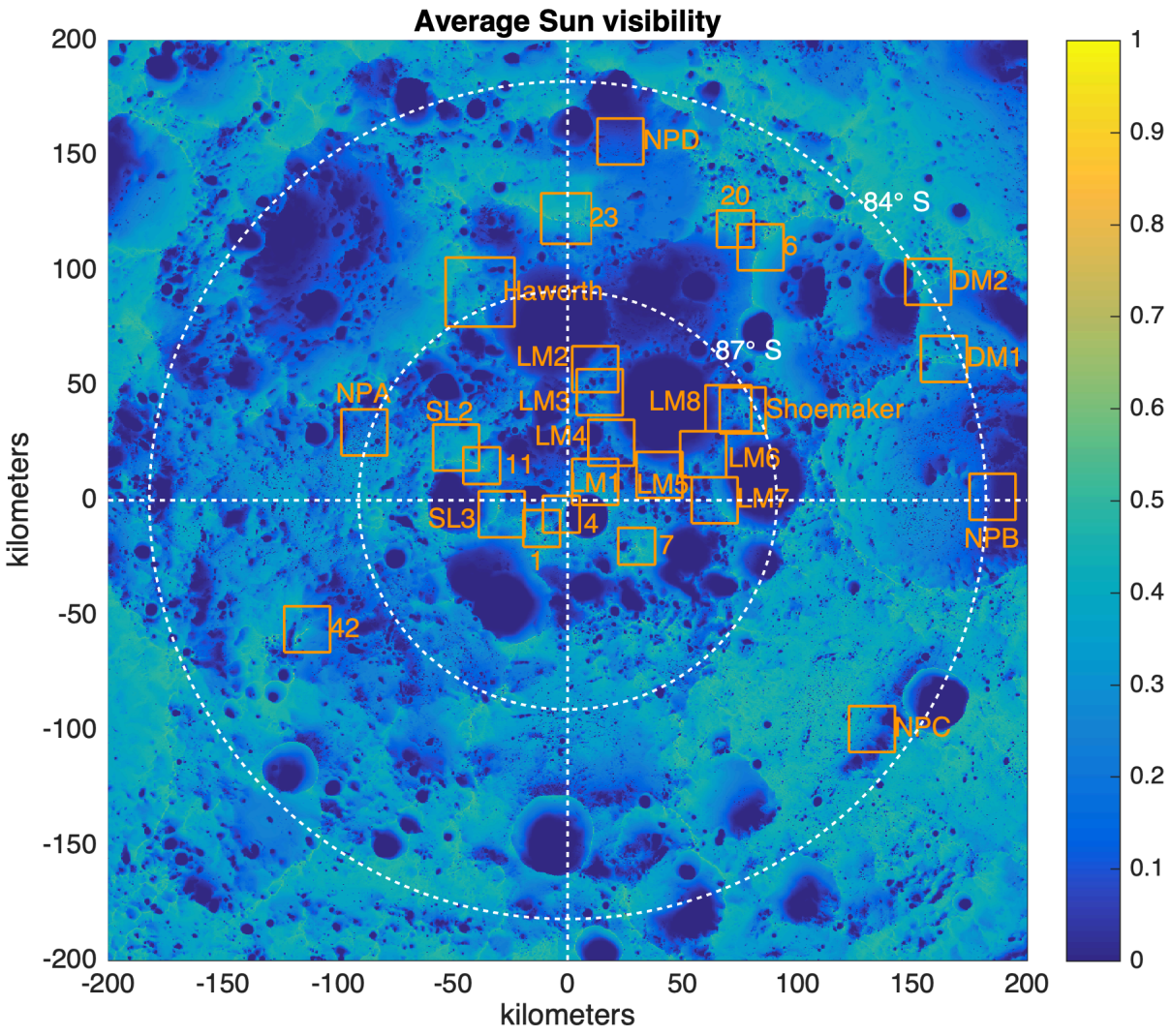


Figure 5.1 Available LOLA topographies[67].

5.3 Tests

The tests were conducted using the Microsoft Edge browser, Version 136.0.3240.76 (Official build) (64-bit) on a computer running the Windows 11 Pro operating system (Version 10.0.22631 Build 22631). The specifications of the computer were cpu: i7-13700K, video card: RTX-4070 TI, installed memory: 64GB. The used display was configured to use 144Hz refresh rate. To run the tests the project was set up and a vite dev server was started using the `npx vite` command. The localhost url reported in the terminal was opened in the browser. The size of the canvas element was fixed at 1920 x 1024 pixels.

To connect the component to ROS during tests, roslibjs[68] was used in the browser and rosbridge_suite[20] was used on the ROS side to allow websocket connections.

5.3.1 KuupKulgur square SLAM recording

To test updating data from a non ROS source a CSV file containing Simultaneous Localization and Mapping (SLAM) timestamps, positions and rotations was provided. The file was loaded and parsed. Several rows of zeroed out data preceded actual data which were removed. The RxJS[69] library was used to simulate emitting the data in a continuous stream with delays between emissions. The SLAM data points were found to have been recorded with approximately 10 millisecond delays between them. The HTML standard enforces at least a 4ms delay between timer callbacks after the first 5 consecutive callbacks[70]. The timer callback time is also not guaranteed to be accurate as there are other things running on the page e.g the rendering loop for the 3D scene. For the test, all of the points were played back and the rover moved as although slightly floating above the terrain, as the terrain had been changed since the laser scan.

5.3.2 Testing the point cloud

To test the point cloud feature a bag file containing a point cloud recording by KuupKulgur in the lunar analogue environment was used. A robot model and a point cloud were added to the scene, the point cloud data being shown is illustrated on figure 5.2. The bag file was first inspected using the ros2 bag info command to find the relevant topic and then played with the ros2 bag play command. The messages were parsed and processed, since the point cloud points data in the message is stored as a binary blob and not an easily accessible array containing point data. The binary data string was first converted from the Base64 encoding and converted into a uint8 buffer. The points data was read using the field definitions provided by the pointCloud2 message. During playback the frames per second in the scene were constant between 143-144 however some moments of stuttering or freezing were noticed in the points of the cloud.

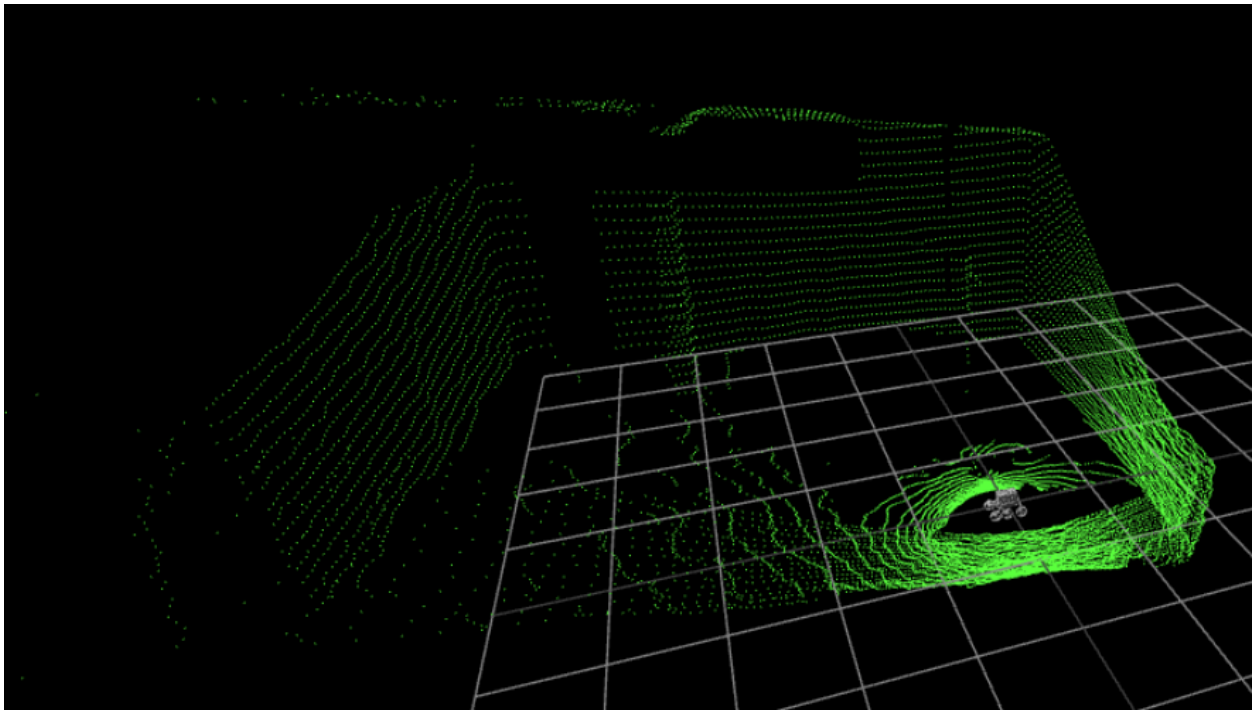


Figure 5.2 point cloud testing with recorded LIDAR output from a bag file

5.3.3 Rover simulation testing on lunar terrain

A scene was created with the NPD lunar terrain model and KuupKulgur URDF rovers added. The cases of 1, 50 and 100 rovers were tested. Labels and position uncertainties were enabled. First a static scene with the rovers doing nothing was tested. Then the rovers were connected to turtlesim. The frontend code was sending twist messages at 1 second intervals to turtlesim to move the turtles and was listening on the turtle pose topic for position info with message throttling configured at 100ms to simulate full two-sided communication. The measurements were performed for 60 seconds. The first 5 seconds of which are shown on the graph in figure 5.3. During the tests with 50 and 100 moving rovers it was found that roslibjs was unable to handle such a large number of connections and most connections got dropped or never managed to connect. Around 15-20 simultaneous connections seemed to be the maximum over several tests.

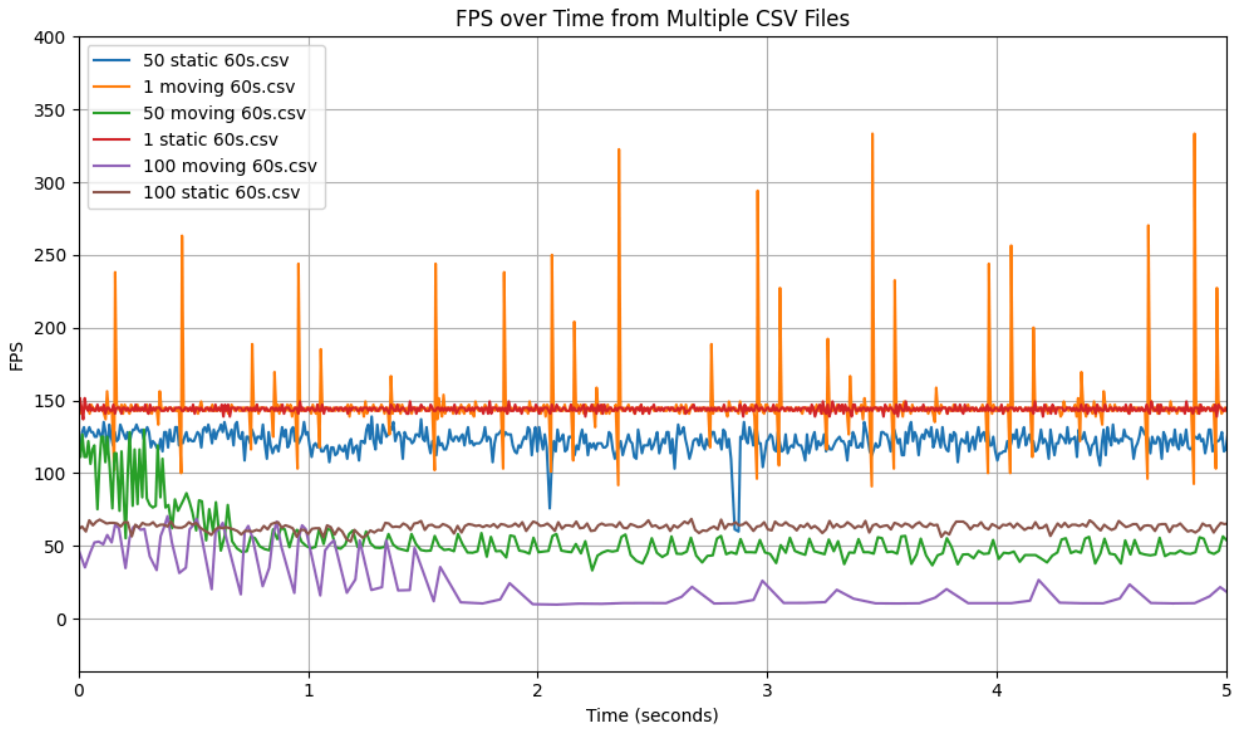


Figure 5.3 performance testing

5.4 Integration of the library

The library was built using the npx vite build command in the project directory and added to the KuupKulgur MCI as a NPM package. The package was added as a file system link as it was not uploaded to npm registries such as the public npmjs registry. The npm install command was used. Afterwards the <k3d-editor> html element was added to the html file of the 3d sub-page of the application, the required styles were added to the stylesheet and assets were defined in a script on the page. The component loaded and it was possible to create, edit and view scenes, the figure 5.4 shows the component being used.

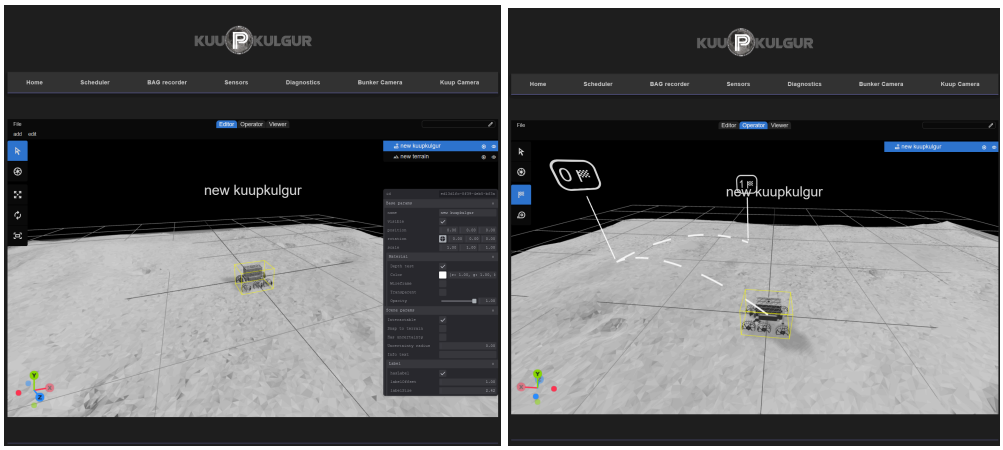


Figure 5.4 library integrated in the MCI

5.5 Discussion

Some stuttering was noticed when testing the point cloud which was found to be caused by NaN values in the point cloud point data. In the context of ROS NaN values represent invalid readings of a sensor[71]. The stuttering disappeared when the NaN values were replaced with zeroes. The erroneous values could also just be dropped instead of being zeroed out; however this requires disposing of old and allocating a new buffer as Three.js stores the point cloud points data in a fixed-length vertex buffer[64]. Another option would be to order the vertices so that the NaN values end up at one of the ends and the updating draw range property of the buffer geometry object to skip those values. A third possible option would be to reduce the point cloud point count by a certain percentage which is higher than the erroneous value count.

The results confirm that data related to KuupKulgur e.g a point cloud can be visualized and that several rovers can be visualized in a lunar environment without any performance problems. Only at the extreme end of 100 rovers being on screen does performance start to severely degrade. From the performance test chart in figure 5.3 a pattern emerges where the messages being received disrupt the rendering. To alleviate the issue it might be possible to space out the received location messages by delaying the topic subscription during a turtlesim test scenario. For the future special considerations might be needed as, roslibjs provides per topic throttling but doesn't

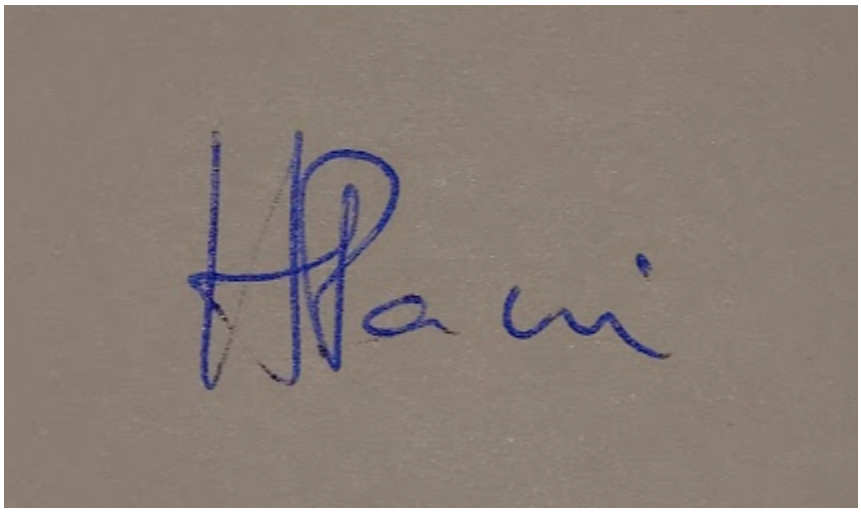
provide any features to throttle out multiple separate topics as a whole, which would be an issue with 100 or even 50 rovers being visualized.

6 Conclusion

The goal of this thesis was to investigate the requirements for and to develop a web-based 3D visualization component for the KuupKulgur Mission Control Interface. Different telerobotics interface types were analyzed and ideas gathered implemented in this thesis. Several user interfaces were examined in closer detail and used as a basis for the development of the 3D visualization component. Additionally a vision of the lunar environment in the near future was analyzed to find and implement what might be possible to visualize for the operator or the public. Although the visualization component was developed specifically for Kuupkulgur, its availability as a stand-alone library and adaptable design makes it suitable for use in other mission control systems or space-related applications.

Acknowledgements

I would like to thank my supervisor, Quazi Saimoon Islam, who gave me valuable feedback during critical moments of writing this thesis.



Bibliography

[1] Artemis - NASA [Internet]. [accessed 2025 May 19]. Available from: https://www.nasa.gov/humans-in-space/artemis/

[2] Terrae Novae: Europe's exploration vision [Internet]. [accessed 2025 May 20]. Available from: https://www.esa.int/Science_Exploration/Human_and_Robotic_Exploration/Exploration/Terrae_Novae_Europe_s_exploration_vision

[3] ESA Moonlight [Internet]. [accessed 2025 May 20]. Available from: https://connectivity.esa.int/esa-moonlight

[4] Wanted: bright ideas to develop the lunar economy [Internet]. [accessed 2025 May 20]. Available from: https://www.esa.int/Applications/Connectivity_and_Secure_Communications/Wanted_bright_ideas_to_develop_the_lunar_economy?utm_source=chatgpt.com

[5] Nyx | The Exploration Company [Internet]. [accessed 2025 May 20]. Available from: https://www.exploration.space/nyx

[6] KuupKulgur – Tartu Observatory Space Exploration Group [Internet]. [accessed 2025 May 20]. Available from: https://tospexgroup.space/projects/kuupkulgur/

[7] rviz [Internet]. Available from: https://github.com/ros-visualization/rviz

[8] Foxglove - Visualization and observability for robotics developers. [Internet]. [accessed 2025 May 19]. Available from: https://foxglove.dev/

[9] Bartneck C. Human-Robot Interaction. New York: Cambridge University Press; 2024. 324 p.

[10] Murphy RR, Tadokoro S. User Interfaces for Human-Robot Interaction in Field Robotics. In: Tadokoro S, editor. Disaster Robotics: Results from the ImPACT Tough Robotics Challenge [Internet]. Cham: Springer International Publishing; 2019. p. 507–28. Available from: https://doi.org/10.1007/978-3-030-05321-5_11

[11] Ikeda B, Szafir D. Advancing the Design of Visual Debugging Tools for Roboticists. In: 2022 17th ACM/IEEE International Conference on Human-Robot Interaction (HRI). 2022. p. 195–204.

[12] Reynes G. VR-Enhanced Remote Inspection Framework for Semi-Autonomous Robot Fleet [Internet]. Tartu Ülikool; 2024 [accessed 2025 May 16]. Available from: https://hdl.handle.net/10062/107729

[13] Zorec MB. XR Teleoperation Demo Development [Internet]. Tartu Ülikool; 2023 [accessed 2025 May 16]. Available from: https://hdl.handle.net/10062/93449

[14] Why robotics companies choose Foxglove [Internet]. [accessed 2025 May 10]. Available from: https://foxglove.dev/why-foxglove

[15] Nielsen Norman Group [Internet]. [accessed 2025 May 16]. 10 Usability Heuristics for User Interface Design. Available from: https://www.nngroup.com/articles/ten-usability-heuristics/

[16] Pütz S, Wiemann T, Hertzberg J. Tools for Visualizing, Annotating and Storing Triangle Meshes in ROS and RViz. In: 2019 European Conference on Mobile Robots (ECMR). 2019. p. 1–6.

[17] rviz - ROS Wiki [Internet]. [accessed 2025 May 10]. Available from: https://wiki.ros.org/rviz

[18] Robot Web Tools [Internet]. [accessed 2025 May 16]. Available from: https://robotwebtools.github.io/

[19] RobotWebTools/ros3djs [Internet]. Robot Web Tools; 2025 [accessed 2025 May 10]. Available from: https://github.com/RobotWebTools/ros3djs

[20] rosbridge_suite - ROS Wiki [Internet]. [accessed 2025 May 20]. Available from: https://wiki.ros.org/rosbridge_suite

[21] 3D | Foxglove Docs [Internet]. [accessed 2025 May 16]. Available from: https://docs.foxglove.dev/docs/visualization/panels/3d

[22] Foxglove Docs | Foxglove Docs [Internet]. [accessed 2025 May 10]. Available from: https://docs.foxglove.dev/docs

[23] Arzberger J, Zevering J, Arzberger F, Nüchter A. Design and evaluation of an UI for astronauts to control mobile robots on planetary surfaces. In: 2024 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC). 2024. p. 74–81.

[24] Moniruzzaman MD, Rassau A, Chai D, Islam SMS. Teleoperation methods and enhancement techniques for mobile robots: A comprehensive survey. Robotics and Autonomous Systems. 2022;150:103973.

[25] Rea DJ, Seo SH. Still Not Solved: A Call for Renewed Focus on User-Centered Teleoperation Interfaces. Frontiers in Robotics and AI [Internet]. 2022;Volume 9-2022. Available from: https://www.frontiersin.org/journals/robotics-and-ai/articles/10.3389/frobt.2022.704225

[26] Layouts | Foxglove Docs [Internet]. [accessed 2025 May 16]. Available from: https://docs.foxglove.dev/docs/visualization/layouts

[27] Teleop | Foxglove Docs [Internet]. [accessed 2025 May 16]. Available from:

https://docs.foxglove.dev/docs/visualization/panels/teleop

[28] Map | Foxglove Docs [Internet]. [accessed 2025 May 16]. Available from: https://docs.foxglove.dev/docs/visualization/panels/map

[29] Gontscharow M, Doll J, Schotschneider A, Bogdoll D, Orf S, Jestram J, et al. Scalable Remote Operation for Autonomous Vehicles: Integration of Cooperative Perception and Open Source Communication. In: 2024 IEEE Intelligent Vehicles Symposium (IV). 2024. p. 43–50.

[30] ROS Discourse [Internet]. 2020 [accessed 2025 May 16]. Introducing Teleop for ROS - General. Available from: https://discourse.ros.org/t/introducing-teleop-for-ros/15041

[31] teleop_twist_keyboard - ROS Wiki [Internet]. [accessed 2025 May 16]. Available from: https://wiki.ros.org/teleop_twist_keyboard

[32] teleop_twist_joy - ROS Wiki [Internet]. [accessed 2025 May 16]. Available from: https://wiki.ros.org/teleop_twist_joy

[33] Bekkers S, Bettendorf MT, Reill J, Giubilato R, Wedler A. The Lunar Rover Mini: towards a Versatile, Open-Source Mobile Robotic Platform for Educational and Experimental Purposes. In: 17th Symposium on Advanced Space Technologies in Robotics and Automation [Internet]. Leiden, Netherlands; 2023 [accessed 2025 May 17]. Available from: https://elib.dlr.de/202313/

[34] Luz R, Coelho G, Campos M, Abrantes R, Querido R, Pereira M, et al. Remote Operations and Streamlining Communication in Mars Analog Missions: Robot-Agnostic Augmented Interface for Live Annotations During Teleoperated Exploration Tasks. In: 2024 International Conference on Space Robotics (iSpaRo) [Internet]. 2024 [accessed 2025 May 17]. p. 342–8. Available from: https://ieeexplore.ieee.org/abstract/document/10688246

[35] Seidel D, Schmidt A, Luo X, Raffin A, Mayershofer L, Ehlert T, et al. Toward Space Exploration on Legs: ISS-to-Earth Teleoperation Experiments with a Quadruped Robot. In: 2024 IEEE Conference on Telepresence [Internet]. 2024 [accessed 2025 May 17]. p. 10–5. Available from: https://ieeexplore.ieee.org/abstract/document/10841792

[36] Eyes on the Solar System - NASA/JPL [Internet]. [accessed 2025 May 17]. Eyes on the Solar System - NASA/JPL. Available from: https://eyes.nasa.gov/apps/solar-system

[37] LIVE broadcast - Beresheet lands on the Moon Fasten your seatbelts, we are about to land. [Internet]. 2019 [accessed 2025 May 17]. Available from: https://www.youtube.com/watch?v=HMdUcchBYRA

[38] Topics - ROS Wiki [Internet]. [accessed 2025 May 19]. Available from: https://wiki.ros.org/Topics

[39] Messages - ROS Wiki [Internet]. [accessed 2025 May 19]. Available from:

https://wiki.ros.org/Messages

[40] PickNikRobotics/rviz_visual_tools [Internet]. PickNik Robotics; 2025 [accessed 2025 May 19]. Available from: https://github.com/PickNikRobotics/rviz_visual_tools

[41] Foundation B. Blender [Internet]. [accessed 2025 May 11]. Available from: https://www.blender.org/

[42] three.js editor [Internet]. [accessed 2025 May 11]. Available from: https://threejs.org/editor/

[43] KuupKulgur [Internet]. [accessed 2025 May 19]. About. Available from: https://kuupkulgur.space/about

[44] KuupKulgur [Internet]. [accessed 2025 May 19]. Innovative Payload Testing for Lunar Rover Missions. Available from: https://kuupkulgur.space/payload-demonstrator

[45] OpenStax. Astronomy 2e by OpenStax. Houston, Texas: XanEdu Publishing Inc; 2022. 1208 p.

[46] Burns JO, Mellinkoff B, Spydell M, Fong T, Kring DA, Pratt WD, et al. Science on the lunar surface facilitated by low latency telerobotics from a Lunar Orbital Platform - Gateway. Acta Astronautica. 2019;154:195–203.

[47] Moon Fact Sheet [Internet]. [accessed 2025 May 19]. Available from: https://nssdc.gsfc.nasa.gov/planetary/factsheet/moonfact.html

[48] Earth Fact Sheet [Internet]. [accessed 2025 May 19]. Available from: https://nssdc.gsfc.nasa.gov/planetary/factsheet/earthfact.html

[49] Mars Fact Sheet [Internet]. [accessed 2025 May 19]. Available from: https://nssdc.gsfc.nasa.gov/planetary/factsheet/marsfact.html

[50] Balaram B, Canham T, Duncan C, Grip HF, Johnson W, Maki J, et al. Mars Helicopter Technology Demonstrator. In: 2018 AIAA Atmospheric Flight Mechanics Conference [Internet]. Available from: https://arc.aiaa.org/doi/abs/10.2514/6.2018-0023

[51] LunaNet: Empowering Artemis with Communications and Navigation Interoperability - NASA [Internet]. 2021 [accessed 2025 May 19]. Available from: https://www.nasa.gov/humans-in-space/lunanet-empowering-artemis-with-communications-and-navigation-interoperability/

[52] LunaNet Interoperability Specification Document Version 5 [Internet]. Available from: https://www.nasa.gov/wp-content/uploads/2025/02/lunanet-interoperability-specification-v5-baseline.pdf

[53] Wright E. NASA Scientific Visualization Studio. 2022 [accessed 2025 May 19]. NASA Scientific Visualization Studio | Illumination at the Moon's South Pole, 2023 to 2030.

Available from: https://svs.gsfc.nasa.gov/4930

[54] Nielsen Norman Group [Internet]. [accessed 2025 May 20]. Powers of 10: Time Scales in User Experience. Available from: https://www.nngroup.com/articles/powers-of-10-time-scales-in-ux/

[55] Typescript [Internet]. [accessed 2025 May 11]. Available from: https://www.typescriptlang.org/

[56] mrdoob. mrdoob/three.js [Internet]. 2025 [accessed 2025 May 11]. Available from: https://github.com/mrdoob/three.js

[57] lit.dev [Internet]. [accessed 2025 May 11]. Lit. Available from: https://lit.dev/

[58] Web Components - Web APIs | MDN [Internet]. 2025 [accessed 2025 May 18]. Available from: https://developer.mozilla.org/en-US/docs/Web/API/Web_components

[59] vitejs/vite [Internet]. Vite; 2025 [accessed 2025 May 18]. Available from: https://github.com/vitejs/vite

[60] qgis/QGIS [Internet]. QGIS; 2025 [accessed 2025 May 13]. Available from: https://github.com/qgis/QGIS

[61] Johnson G. gkjohnson/urdf-loaders [Internet]. 2025 [accessed 2025 May 20]. Available from: https://github.com/gkjohnson/urdf-loaders

[62] gkjohnson/three-mesh-bvh: A BVH implementation to speed up raycasting and enable spatial queries against three.js meshes. [Internet]. [accessed 2025 May 18]. Available from: https://github.com/gkjohnson/three-mesh-bvh

[63] Akenine-Mo¨ller T, Haines E, Hoffman N. Real-Time Rendering, Fourth Edition. Boca Raton London New York: A K Peters/CRC Press; 2019. 1178 p.

[64] three.js docs [Internet]. [accessed 2025 May 19]. Available from: https://threejs.org/docs/

[65] Cross D. cosinekitty/astronomy [Internet]. 2025 [accessed 2025 May 18]. Available from: https://github.com/cosinekitty/astronomy

[66] Barker MK, Mazarico E, Neumann GA, Smith DE, Zuber MT, Head JW. Improved LOLA elevation maps for south pole landing sites: Error estimates and their impact on illumination conditions. Planetary and Space Science. 2021 Sep 1;203:105119.

[67] PGDA - High-Resolution LOLA Topography for Lunar South Pole Sites [Internet]. [accessed 2025 May 18]. Available from: https://pgda.gsfc.nasa.gov/products/78

[68] RobotWebTools/roslibjs [Internet]. Robot Web Tools; 2025 [accessed 2025 May 16]. Available from: https://github.com/RobotWebTools/roslibjs

[69] RxJS [Internet]. [accessed 2025 May 19]. Available from: https://rxjs.dev/

[70] HTML Standard [Internet]. [accessed 2025 May 19]. Available from: https://html.spec.whatwg.org/multipage/timers-and-user-prompts.html#timers

[71] REP 117 -- Informational Distance Measurements (ROS.org) [Internet]. [accessed 2025 May 20]. Available from: https://www.ros.org/reps/rep-0117.html

Appendix

The thesis comes with a zip archive of all the source code. The public folder inside the archive includes the 3d model files, KuupKulgur URDF files and the Lidar SLAM dataset. The measurements zip file provides the performance testing measurement datasets and the jupyter notebook used to create the graph.

Non-exclusive licence to reproduce thesis and make thesis public

I, Hans Pärtel Pani ,
(author´s name)

1. grant the University of Tartu a free permit (non-exclusive licence) to

   reproduce, for the purpose of preservation, including for adding to the digital archives of the University of Tartu until the expiry of the term of copyright, my thesis

   Development of a 3D web visualization component for KuupKulgur Mission Control Interface ,
   (title of thesis)

   supervised by Quazi Saimoon Islam ;
   (supervisor´s name)

2. grant the University of Tartu a permit to make the thesis specified in point 1 available to the public via the web environment of the University of Tartu, including via the digital archives, under the Creative Commons licence CC BY NC ND 4.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright;

3. am aware of the fact that the author retains the rights specified in points 1 and 2;

4. confirm that granting the non-exclusive licence does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Hans Pärtel Pani
20/05/2025