

Software Design Document

for

Lunar Rover Mapping Robot Controller

Version 2.0.0

Prepared by Matthew Kozirev
SEP UG-17

School of Computer Science,
The University of Adelaide

2017 October 30

Change History

Version	Dated	Edited By	Change Summary
1.0	03/10/2017	Matthew Kozirev	Initial Draft
1.1	19/10/2017	Benjamin Schuh	Feedback from draft submission
1.2	25/10/2017	Benjamin Schuh	Unification of document style and grammar
1.3	27/10/2017	Chau Khuc	Correction to some figures and minor modifications
2.0	30/10/2017	Benjamin Schuh	Final format and grammar check

Related Documentation

ID	Document Name	Version
1	Software Requirements Specifications	2.3.0
2	Software Project Management Plan	2.0.0
3	Testing Report	1.0.0
4	User Manual	1.0.0

Contents

1	Introduction	4
1.1	Purpose	4
1.2	Scope	4
1.3	References	4
1.4	Overview	4
1.5	Constraints	5
2	System Overview	6
3	System Architecture and Component Design	7
3.1	Architectural Description	7
3.2	Component Decomposition Description	7
3.2.1	Rover	8
3.2.2	Communications	8
3.2.3	Graphical User Interface	8
3.3	Detailed Component Design Description	8
3.3.1	Rover	8
3.3.2	Communications	9
3.3.3	User Interface	10
3.4	Architectural Alternatives	13
3.5	Hardware Alternatives	13
3.6	Design Rationale	13
4	Data Design	15
4.1	Database Description	15
4.1.1	Lunar Rover	15
4.1.2	Communications	15
4.1.3	User Interface	16
4.2	Data Structures	17
5	Design Details	18
5.1	Class Diagrams	18
5.1.1	Lunar Rover	18
5.1.2	Communications	18
5.1.3	User Interface	18
5.2	State Diagrams	23
5.2.1	Rover	23
5.2.2	User Interface	26
5.3	Interaction Diagram	29
6	Human Interface Design	31
6.1	Overview of the User Interface	31
6.2	Detailed Design of the User Interface	32
6.2.1	Rover Map	32
6.3	Mini Map	33
6.4	Legend	34
6.4.1	Main Control Panel	35

6.4.2	Toolbar	36
6.4.3	Status Bar	37
7	Resource Estimates	38
7.1	Minimum Requirements	38
7.2	Recommended Requirements	38
A	Definitions	39

1 Introduction

1.1 Purpose

The purpose of this Software Design Document (SDD) is to describe the software design of the product, the prototype of the Rover to be used to fulfil the requirements of the Google Lunar X-Prize. It will be used as a guide for team members in completing the product, as well as providing others an insight to the design decisions made in developing the product.

The intended readership of this document includes team members involved in creating this product, lecturers acting as clients, and software developers involved in maintaining the product in the future.

1.2 Scope

The product described in this document is a prototype of the Lunar Rover built with a Lego Mindstorms EV3 robot. The two main requirements to claim the Google Lunar X-Prize are:

- Travel more than 500 metres
- Transmit high-definition images and video

Our prototype shall fulfil a certain aspect of the first requirement, whereas the absence of a camera will prevent the team from achieving the second requirement.

For the first requirement, the robot will be able to autonomously travel and survey the area in an A1 sheet of paper representing a 500m box.

The autonomous survey involves detecting and recording notable elements of the map, which are coloured lines of varying thickness. Furthermore, the robot shall allow manual survey of the area, that is, the robot shall provide a user with the ability to remotely control the robot from a computer.

The objective upon delivery is to create a prototype that can be used as the basis for the design of the actual Rover to be sent to the Moon.

1.3 References

Software Requirements Specification (SRS), found at:

https://github.cs.adelaide.edu.au/a1210255/2017-S2-SEP-UG17/tree/master/Deliverable_Documents/SRS

1.4 Overview

There are eight main sections in the SDD:

1. Introduction

The introduction shall set out the intended behaviours of the product and the constraints of the product based on limitations and restrictions.

2. System Overview

The system overview shall describe the overall system architecture, in terms of the sub-systems that make up the overall system. It shall contain the expected behaviour of each system and the interactions between subsystems.

3. System Architecture and Components Design

The system architecture and components design shall describe further details of the design of each subsystem, including the components comprising the subsystems. Each description shall contain its purpose, function, subordinates, dependencies, interfaces, and internal data. Also, other designs considered during the design process shall be discussed, as well as the design rationale for the current system design.

4. Data Design

The data design shall describe the databases of the system and data structures involved in creating the databases.

5. Design Details

The design details shall describe the design of the system in relation to the requirements detailed within the SRS. Descriptions will include class diagrams, state diagrams, and interaction diagrams describing the system design.

6. Human Interface Design

The human interface design shall present a description of each implemented GUI functionality that aids in fulfilling the project requirements whilst maximising both usability and user experience.

7. Resource Estimates

The resource estimates define the minimum and recommended system requirements of a computer to run the program.

8. Definitions, Acronyms, and Abbreviations

All uncommon definitions, acronyms, and abbreviations for this document shall be defined in the corresponding parts of this section.

1.5 Constraints

The implementation of the prototype is limited by the capabilities of the sensors used. The colour sensor can detect seven different colours - black, blue, green, yellow, red, white, and brown. Therefore, colours of different shades or colour values not available will be detected as one of the seven colours or not detected at all, that is, the detected colour will be none. This limits the variety of elements that can be detected using the colour sensor.

The ultrasonic sensor has a wide beam to detect distances from objects in front of the robot and gives a distance reading from 1 cm to 250 cm. As a result of the wide beam, the robot may detect objects that are not directly in front of it, causing the robot's movement to stop when near another object.

As communications are sent through BlueTooth, the robot can only be controlled within a radius of 10 m due to the range of BlueTooth transmissions. Thus, the operator will have to control the robot from a computer that is in the same room as the robot.

2 System Overview

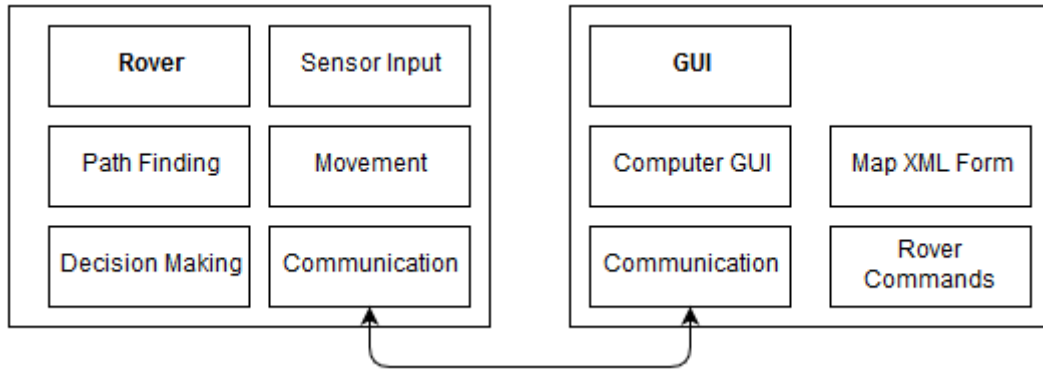


Figure 1: System Overview

The overall system design can be described by the context model above, where arrows indicate interactions between subsystems. The model can be further divided into the three main subsystems of the design: the Rover, communications, and graphical user interface (GUI) system. The back end comprises the Rover system and the communications system - the user does not have direct access to these systems. The front end only comprises the GUI system, which allows users to interact with a program to change the behaviour of the robot.

The Rover system contains all the logic for the Rover to operate autonomously and allows the user to control the robot's movement. Its subsystems include the movement, path finding, mapping, sensor, and data processing systems. The path finding system provides the user the ability to move the robot to a designated point, while also avoiding dangerous situations. This system relies on the movement system as the path finding system merely finds the shortest and safest path to travel to a point. The sensor system sends data to the GUI system so that the on-screen map may be updated and also sends data to the data processing system to then determine where the robot shall travel in autonomous mode. The mapping system reflects sensor detections in the map file that is to be exported.

All subsystems of the Rover system play a key role in determining the robot's movement such that the robot's safety is maintained and any objects remain undamaged.

The GUI system provides a visual representation of the data received from the robot's sensors and allows the user to control the robot's movement when in manual mode. The user may also influence the robot's movement by marking NGZs on the map produced by the GUI system. As the robot is designed to avoid entering NGZs, the robot shall manoeuvre around these marked areas.

The communications system ties the front-end and the back-end together by sending the commands from the user to the Rover and sending data from the Rover to the user. The data received is then handled by the appropriate functions on each end.

3 System Architecture and Component Design

3.1 Architectural Description

The Lunar Rover system can be divided into three primary subsystems, as shown below in figure 2. The three subsystems are the Rover itself, the Graphical User Interface (GUI), and the communications between the Rover and GUI. Each of these subsystems can be further divided into individual components, detailed in section 3.3. The interactions and relationships between all the subsystems and components are also shown in figure 2.

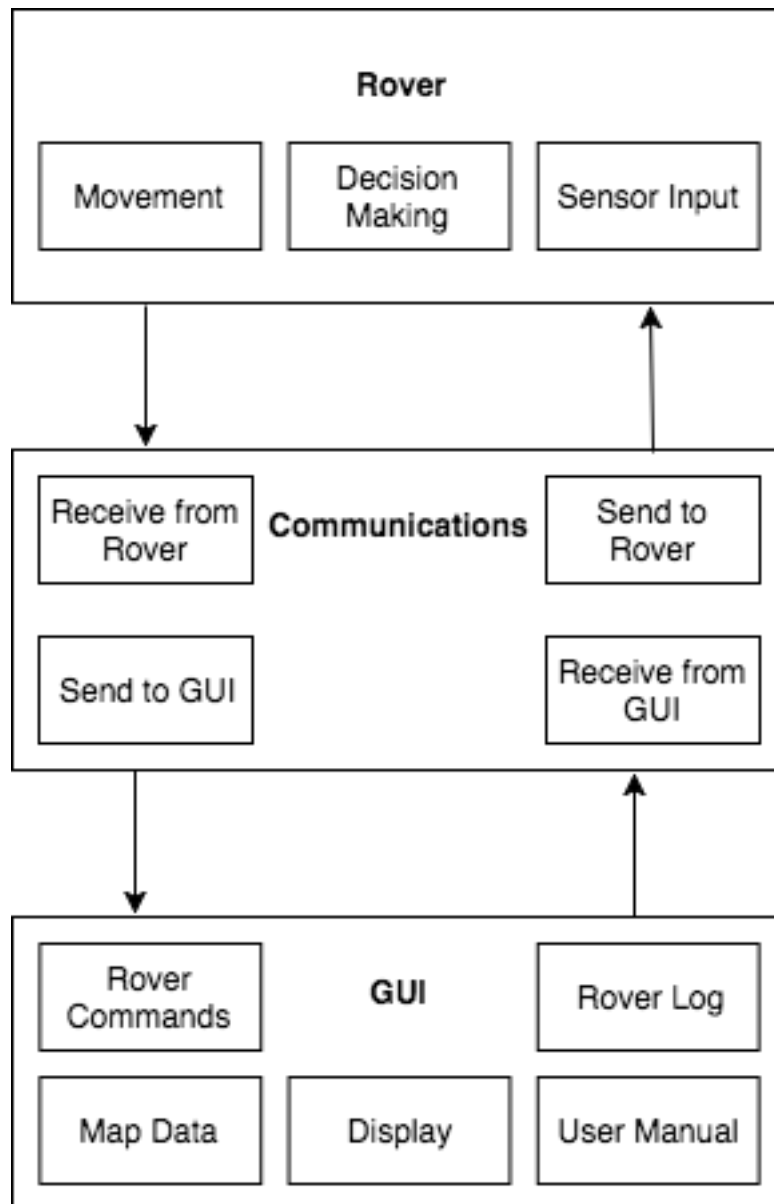


Figure 2: System Architecture

3.2 Component Decomposition Description

This sections describes the overall functions of each of the three main subsystems which work together to implement the overall system.

3.2.1 Rover

The Rover is the physical device which will land on and explore the moon for the Apollo 11 landing site. It contains all the sensors, motors, and intelligence to autonomously navigate to the landing site and locate the Apollo 11 lander, as well as identify any other objects of interest in the area around the lander, as specified in the appropriate sections of the SRS document. The individual components which form this subsystem are illustrated above in figure 2.

Since the Rover contains sensors, it is also responsible for its own safety, which is part of its internal implementation, specifically, its decision making, as seen in figure 2. The safety features of the Rover include:

- Avoiding areas marked with black lines and no-go zones (NGZs)
- Preventing contact with objects by detecting the distance between the robot and the object in front of it
- Changing direction once the robot has bumped into an object

3.2.2 Communications

The communications subsystem is client-server system which connects the Rover and GUI subsystems and allows them to communicate with each other via a Bluetooth connection in near real-time. This subsystem is not implemented on its own device, rather it is split between the Rover and GUI, hence its separation as its own subsystem. The individual components which form this subsystem are illustrated above in figure 2.

3.2.3 Graphical User Interface

The GUI is a control panel displaying the map of the area the Rover is exploring, and updating all features found by the Rover on it in real time. It has various options to direct the Rover's movement directly, such as movement keys, return to landing site button, emergency stop button, etc. It also includes options to import and export the Rover Map. The individual components which form this subsystem are illustrated above in figure 2.

3.3 Detailed Component Design Description

3.3.1 Rover

Rover Component 1: Main Driver Function

Identifier: RC01

Purpose: R0001, R0002, R0003, R0017.

Function: Integrates all the individual Rover software components to control their interactions and implement all the required Rover functionalities.

Subordinates: None.

Dependencies: RC01, RC01, RC01, CC02.

Interfaces: Receives data from the movement (RC02), sensor, navigation, and communication (CC02) components.

Data: Instanced objects of all other Components.

Rover Component 2: Movement

Identifier: RC02

Purpose: R0002, R0003, R0005.

Function: Sets up all the required leJos objects to implement Rover movement and location tracking. Includes functionality to move the Rover specified distances, rotate Rover heading by angle, update/return the Rover's current position and heading, move the Rover to specified position, unconditionally start the Rover moving in specific directions, and set the movement speeds and accelerations.

Subordinates: RC01

Dependencies: All functions must return immediately so that the main driver function (RC01) can continuously loop to check sensor data for hazards/objects of interest, and poll the communications (CC02) for new data.

Interfaces: Function parameters to specify the details of how the Rover should move/update its position. The ports the motors are connected to.

Data: LeJos objects to interact with the Rover motors and track position. Other variables required to implement required functionalities.

Rover Component 3: Sensors

Identifier: RC03

Purpose: R0001, R0002, R0003, R0017.

Function: Sets up all the required leJos objects to implement and interface with the Rover's sensors. Collects all the sensor data and converts it to a format more useful for other components.

Subordinates: RC01

Dependencies: Data types used to return the sensor values need to be appropriate for the sensor, e.g. boolean for touch sensor.

Interfaces: Return functions to give the current values seen by the sensors.

Data: LeJos objects to interact with the Rover's sensors.

Rover Component 4: Navigation

Identifier: RC04

Purpose: R0002, R0003, R0004, R0005.

Function: Sets up the leJos objects required to implement navigation and pathfinding through a specified line map. Includes functions to generate a path through the map, update the map, and follow the path to reach the desired goal position while avoiding obstacles in the map. The map represents areas to be avoided such as NGZs.

Subordinates: RC01

Dependencies: The movement handled by this component must not block the main Rover function (RC01).

Interfaces: Desired locations and the map of obstacles to avoid.

Data: Collection of waypoints comprising the path, copy of main movement component to implement movement, leJos objects required for functionality.

3.3.2 Communications

Communications Component 1: Communications Core

Identifier: CC01

Purpose: R0028 R0029 R0038 R0039

Function: Core communications functionality, which encapsulates the methods required to

implement the communications protocols used by both the Rover and GUI communications. It includes the functionality to:

- Establish and connect to BlueTooth and WiFi servers
- Send and receive data
- Configure multiple data channels for sending and receiving
- Check for available data
- Close connections

Subordinates: CC02, CC03

Dependencies: None.

Interfaces: Any data then needs to be sent between the Rover and GUI, formatted as a string. Input parameters for establishing the connections, such as IP and port.

Data: Stores data related to the connections, such as sockets, stream objects, and communications configuration variables.

Communications Component 2: Rover Communications

Identifier: CC02

Purpose: R0028 R0029

Function: Extends the communications core to implement the Rover specific communication functionalities. These functionalities mainly relate to sending updates on the Rover's status to the GUI, parsing the data received from the GUI into a format usable by the Rover, and sending map updates to the GUI. It also buffers data received from the GUI to minimise bandwidth requirements and communications overhead (only need to transmit when a change occurs). The updates on the Rover's status are automatically retrieved by this object using pointers to the appropriate component objects, which are passed on the creation of the Rover Communications component.

Subordinates: RC01.

Dependencies: CC01.

Interfaces: Return buffered data from GUI. sensor, positional, and map data.

Data: Buffered data from GUI communications, pointers to other component objects

Communications Component 3: User Interface Communications

Identifier: CC03

Purpose: R0038 R0039

Function: Extends the communications core to implement the GUI specific communication functionalities. Including sending GUI event data to the Rover to execute user commands and update Rover operation parameters such as the map. It also parses and buffers data received from the Rover for the same reasons outlined in CC02.

Subordinates: UC01, UC04, UC05, UC06.

Dependencies: CC01.

Interfaces: Return data from the Rover. GUI event occurrences and data updates.

Data: Buffered data from Rover communications.

3.3.3 User Interface

User Interface Component: Rover Map

Identifier: UC01

Purpose: R0006 R0030 R0031

Function: Displays the map surveyed by the Rover on the main map, and updating both the Rover's new positions and headings and new elements on the map in real time. With different modes toggled on the toolbar, the user can also make modifications on it directly with mouse maneuvers. It can be zoomed in with the plus key and zoomed out with the minus key on the keyboard number pad. It can also be panned with mouse drags.

Subordinates: None.

Dependencies: CC02

Interfaces: Function to read strings containing new information (new elements found by the Rover or in imported maps) and update them graphically on the map.

Data: A matrix of cells containing information of the current map without NGZs (so that when NGZs are removed the original map could be redrawn), and a matrix of cells representing the current map.

User Interface Component: Mini Map

Identifier: UC02

Purpose: R0027

Function: Displays a constantly zoomed inned map always having the Rover at its centre.

Subordinates: None.

Dependencies: CC02

Interfaces: The same interface as UC01, but without interactive functions for users.

Data: Sharing UC02's data.

User Interface Component: Legend

Identifier: UC03

Purpose: R0008 R0016

Function: Displays the legend of all elements that may appear/are appearing on the map. When each legend item is clicked with left mouse button, the user can choose a new colour among a predefined list for that element.

Subordinates: None.

Dependencies: CC03

Interfaces: Configurable element colour selected by mouse clicks.

Data: An array of elements whose colours are configurable, and an array storing all possible colours.

User Interface Component: Main Control Panel

Identifier: UC04

Purpose: R0011 R0012 R0013 R0014

Function: The buttons functions include: Arrow keys that can be controlled by keyboard arrow keys and mouse clicks. The Rover will move according to the arrow pressed in manual mode. Two options to switch between manual mode and auto mode. A STOP button that once pressed will trigger emergency mode, making the Rover stop all actions as soon as possible. The emergency mode can be disengaged by pressing the button twice within a second. An option to command the Rover to return to the initial starting point.

Subordinates: None.

Dependencies: CC03

Interfaces: Utilizing the User Interface Communications module to send commands to the Rover.

Data: Record of current mode to avoid being in multiple conflicting states, and for disabling options that are not possible under current mode.

User Interface Component: Toolbar

Identifier: UC05

Purpose: R0007 R0009 R0010 R0019 R0021 R0022

Function: The various options on the toolbar include:

- One button to import Rover Maps.
- One button to export the current Rover Map.
- One button to toggle NGZ marking mode, that will allow the user to draw NGZs on the Rover Map with mouse clicks and drags.
- One button to toggle NGZ removal mode, that will allow the user to erase NGZs on the Rover Map with mouse clicks.
- One button to toggle target setting mode, that will allow the user to designate goal points for the Rover to travel to on the Rover Map.
- One button that once clicked, will open the user manual PDF.

Subordinates: None.

Dependencies: UC01

Interfaces: Various interfaces including:

- Interface for sending new data to the Rover Map.
- Interface for toggling different modes to call corresponding functions on mouse click and mouse drag.
- Interface that will invoke system calls to read and/or write files.

Data: Record of current mode to prevent conflicting behaviours. Record of current mouse action and status for drawing functions.

User Interface Component: Status Bar

Identifier: UC06

Purpose: R0015 R0023

Function: Displays current Rover behaviour/status, and constantly updating the information in real time. When clicked, it will open a new window showing the full log of past behaviours/status.

Subordinates: None.

Dependencies: CC03

Interfaces: Utilizing the User Interface Communications model to receive data from the Rover in real time.

Data: All past actions of the Rover are stored in strings.

3.4 Architectural Alternatives

Another architecture was considered but not chosen as the design, as the development team were not able to find any way to meet the required functionality and interactions between the Rover and User Interface. The other architecture that was considered was a system where the logic to decide the Rover's actions would be held on the pc that was running the user interface, however, it was determined that such a design would lose more time than it saved due to the need for excess communications, which can be slow with a Bluetooth connection.

Also, an architecture similar to the one currently employed in the Rover was considered that would instead utilise a Wi-Fi connection operating through a sufficient medium, as it was believed that this could improve the speed of communications, however, the communications subsystem would have needed substantial redesigning and it was deemed too difficult a task to be implemented.

A previous architectural decision for the autonomous movement of the Rover relied heavily on the leJOS navigation packages, namely the function `moveTo`. Due to the inconsistent behaviour of the Rover, as a result of using the `moveTo` function, this alternative was not viable. So, the current architecture relies less on the leJOS implementations of navigation.

3.5 Hardware Alternatives

There are very few hardware changes that can be made to the design of the project, though some minor improvements have been suggested. Firstly, if a Wi-Fi communications subsystem were to be used, then a modem or mobile phone hot spot would have to be included in the physical architecture of the design. This was part of why a Wi-Fi connection was deemed difficult to implement. As this is a prototype for the Lunar Rover the range of Bluetooth (up to 100m) is acceptable, however, in practice a different technology is required to send communications to and from the Moon.

Also, there are configuration changes that can be made to the Rover, such as rearranging the location of its various sensors to provide better information or allow for better movement. One such considered change would be to have the colour sensor closer to the front of the Rover to allow for better recognition of no-go zones. Placing the colour sensor in front of the Rover would allow the Rover to detect a no-go zone before it risks entry. Also, it is unlikely that the touch sensor will detect any objects due to the logic of the robot. Currently, the robot will detect objects a certain distance away through the ultrasonic sensor, so all obstructions will be detected before the robot collides with it.

3.6 Design Rationale

The architecture outline in section 3.1 was chosen as it encapsulated the two primary deliverables which must be given to the client upon component completion, The Lunar Rover and a ground based User Interface. The third subsection was added to encompass the communications and interactions between the two, as both primary deliverables are required to talk to each other using a standard protocol. Separating this into its own subsection allowed the protocols used to be standardised between the two primary sections in a much easier and simpler fashion. The component division was chosen as it encapsulated all the related functions under a single object, simplifying the interactions of these related methods. This did however introduce some

issue with interactions between a select few methods. Mainly due to the immediate return requirement for most methods, required to ensure the main Rover driver loop in the Rover did not get blocked.

4 Data Design

4.1 Database Description

There is no explicit database structure used to store data in the Lunar lander project, rather, an object oriented data storage structure is used. All the data required by and used by the three main sections (the Rover, User Interface, and Communications) is stored in the objects comprising the individual components within each section. If data is required by multiple components, a reference of the data or object containing the data is passed to the additional components which require access to it. Figure 3 demonstrates the way in which the objects allow each other access to necessary data.

4.1.1 Lunar Rover

The Lunar Rover features four main packages that control and constrain its behaviour. These are main, movement, navigation and sensors. Each of these packages contain classes that store and manipulate data to direct the Rover to automatically search and map the area while meeting safety and efficiency requirements. The classes are StartRover, SpecificMove, Pathfinder, Mapper, Navigator, and Sensor, and the interfaces are ChassisMovement and BaseSensor, which are implemented by the SpecificMove and Sensor classes respectively. The Rover also includes a set of classes dedicated to maintaining communications with the GUI, which are shared from the communications section.

These classes are shown in figure 3 in orange. The diagram shows that the objects made from these classes hold and maintain all the necessary data that they require for operation, and pass it to each other by reference, as is the standard in Java derived languages. Figure 3 also shows that all information stored in the Rover's core classes is passed to the user interface via the communications classes.

4.1.2 Communications

Both the GUI and the Rover share classes for communications, but these classes are part of the communications section. The communications section is comprised of four classes, three of which have interfaces. The interfaces CommunicationsAPI, RoverCommsAPI and GUICommsAPI are implemented by the classes Communications, RoverComms and GUIComms respectively. The RoverComms and GUIComms inherit from the Communications class and specialise towards the sections that they serve. The RoverComms class contains objects related to Rover functions so that it can access necessary data by reference and send this information to the GUI via a wireless connection. The GUIComms class contains float values related to position, heading and sensors so that it can receive data, store it, and then pass it to the necessary objects. The communications section also contains the class XMLParser, which stores document data related to mapping.

These classes are shown in figure 3 in green. The diagram shows that these classes contain a few references to data maintained elsewhere or to other objects, and that they serve as the pipeline for all data to travel between the Rover and the user interface.

4.1.3 User Interface

Beside the data received from the Rover through the GUIComms interface, and data to be updated to the Rover (all as strings), the GUI also stores the current map data and another version of it without NGZs in integer array format. The latter is required so that when NGZs are removed the original state of the cells before the NGZs were applied can be restored. All elements on the map are stored as cells with different identifications numbers, which can be parsed from a given map file, and exported to a file following the format specified in the DTD. There are also flags indicating the current state the GUI is in (such as auto or manual mode, NGZ drawing states, destination setting states, etc.), which will make sure conflicting commands will either be disabled or ignored. Finally, there are temporary variables to trace the mouse's trails and past actions to determine the effects of mouse press, mouse drag, and mouse release, but these will not be exported or written to any file.

The main user interface classes are shown in figure 3 in purple. Objects created from these classes also encapsulate any of their necessary data, however, since the user interface is quite complicated, there are some auxiliary classes which also provide functionality, however, there were too many such classes with too little practical purpose to include them all in the diagram. Those classes can be found in the class diagrams.

4.2 Data Structures

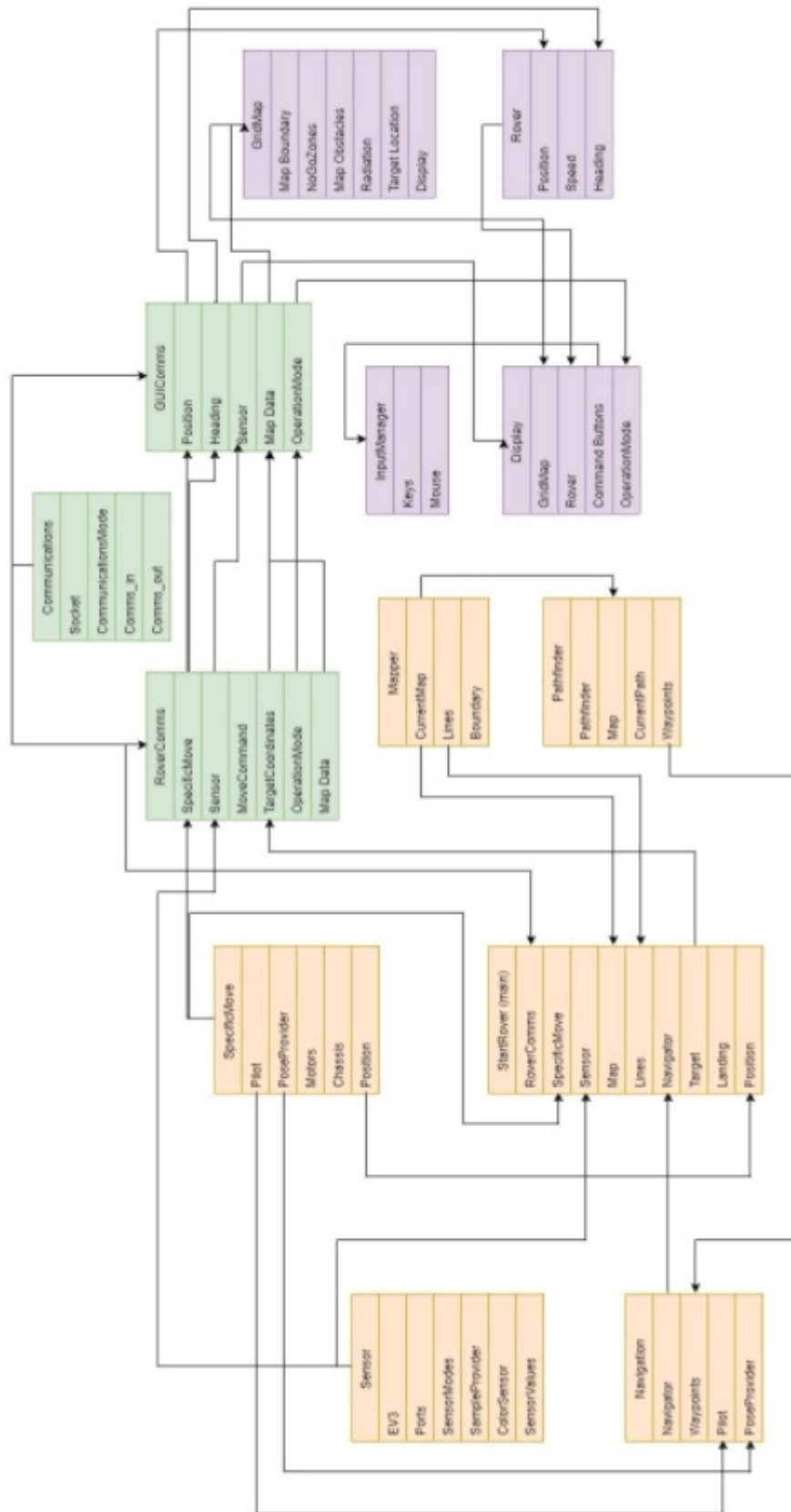


Figure 3: Data Structure Diagram

5 Design Details

5.1 Class Diagrams

5.1.1 Lunar Rover

The Rover section of the system is comprised of 6 classes which apply a variety of leJos API defined classes with some user made code to assist in their utilisation. These classes are the same as those mentioned in section 4.1.1. Figure 4 details the internal state and methods of all of these classes.

5.1.2 Communications

The Communications class diagrams are included with the Rover class diagrams to keep the diagrams more concise. There is a single Common class which is used by both the Rover and GUI communications classes through inheritance. The respective Rover and GUI communications classes implement the Rover and GUI communications specific methods.

5.1.3 User Interface

To begin with, there are basic classes to load in assets for the GUI (which was aptly named Assets.java) and those to parse (import) and write (export) Rover Maps. There's also usually one class for each type of element on the map, such as the status bar, the toolbar, the legend, the arrow keys, etc. The two most important classes are Display.java and GridMap.java. The former is the interface on which all the elements of the GUI are placed, and the latter is the main Rover Map, which, besides updating the information sent from the Rover or elements specified by the user, can also store all the information necessary to export the Rover Map to a file with the same format as the input Rover Maps, but with many more details, and also supports all the possible modifications on every cell before doing so. There are also classes to dictate the whole programs running behaviours and various supplementary functions to support such transitions, which will take too much space if each were to be elaborated on.

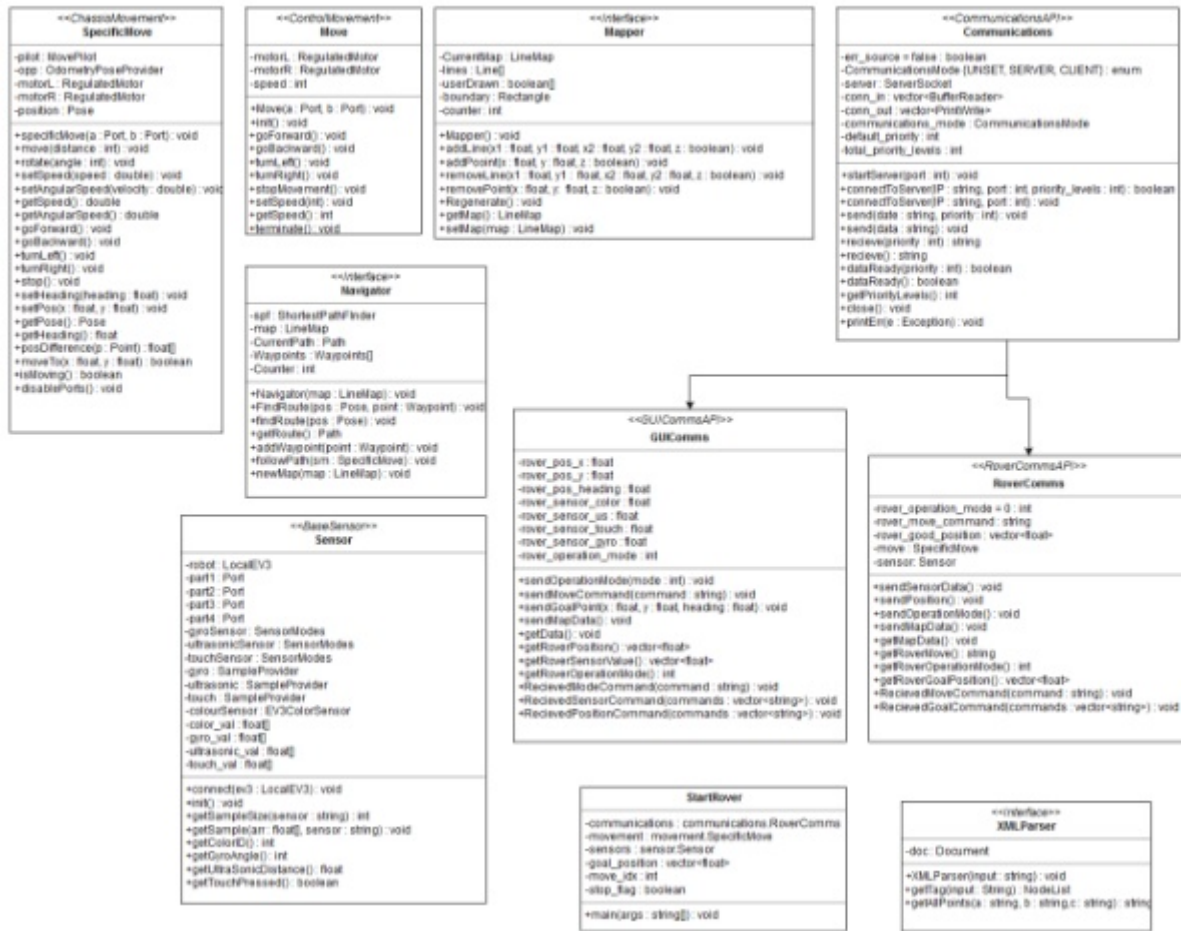


Figure 4: Rover and Communications Class Diagram

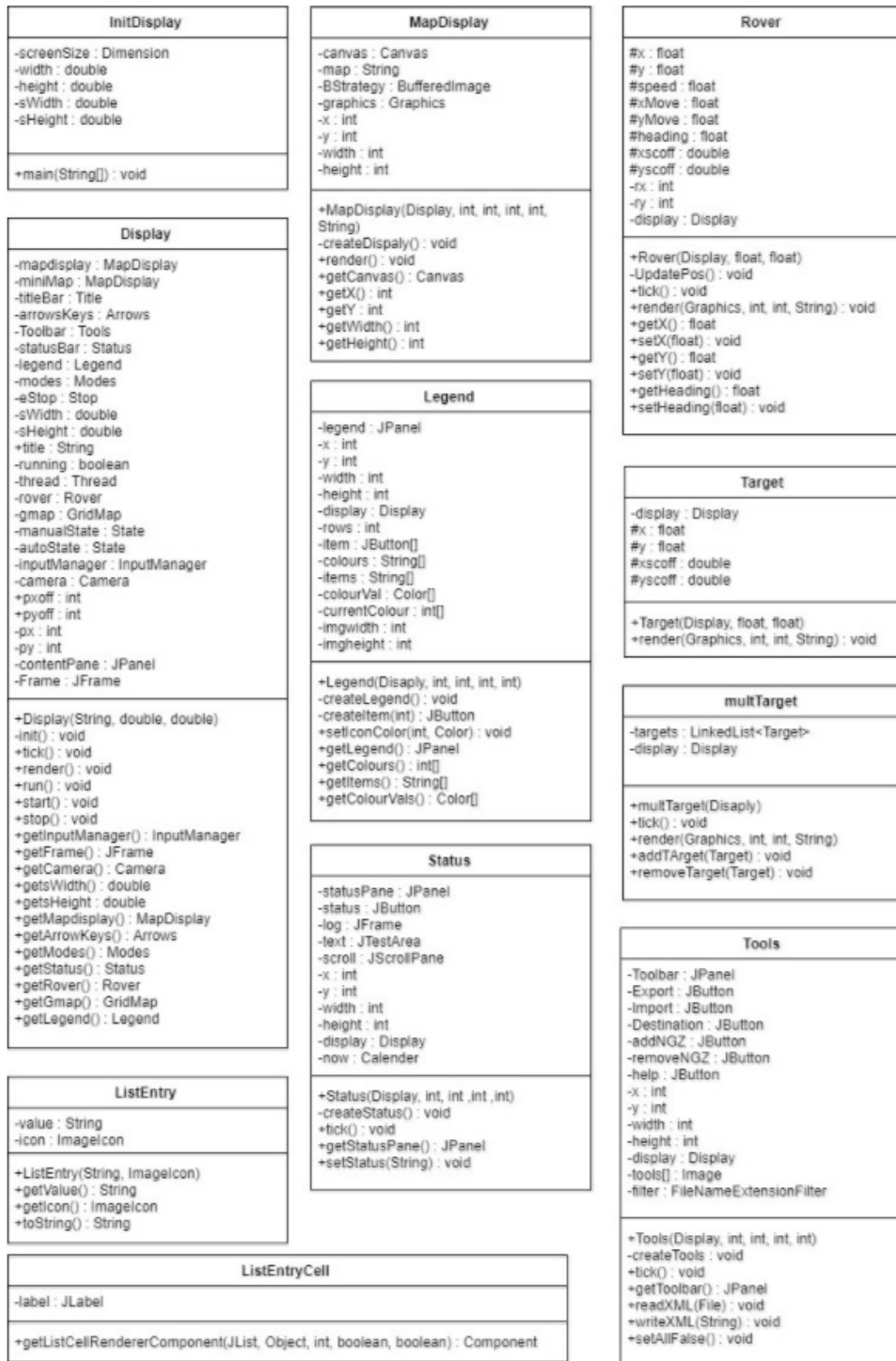


Figure 5a: GUI Class Diagram

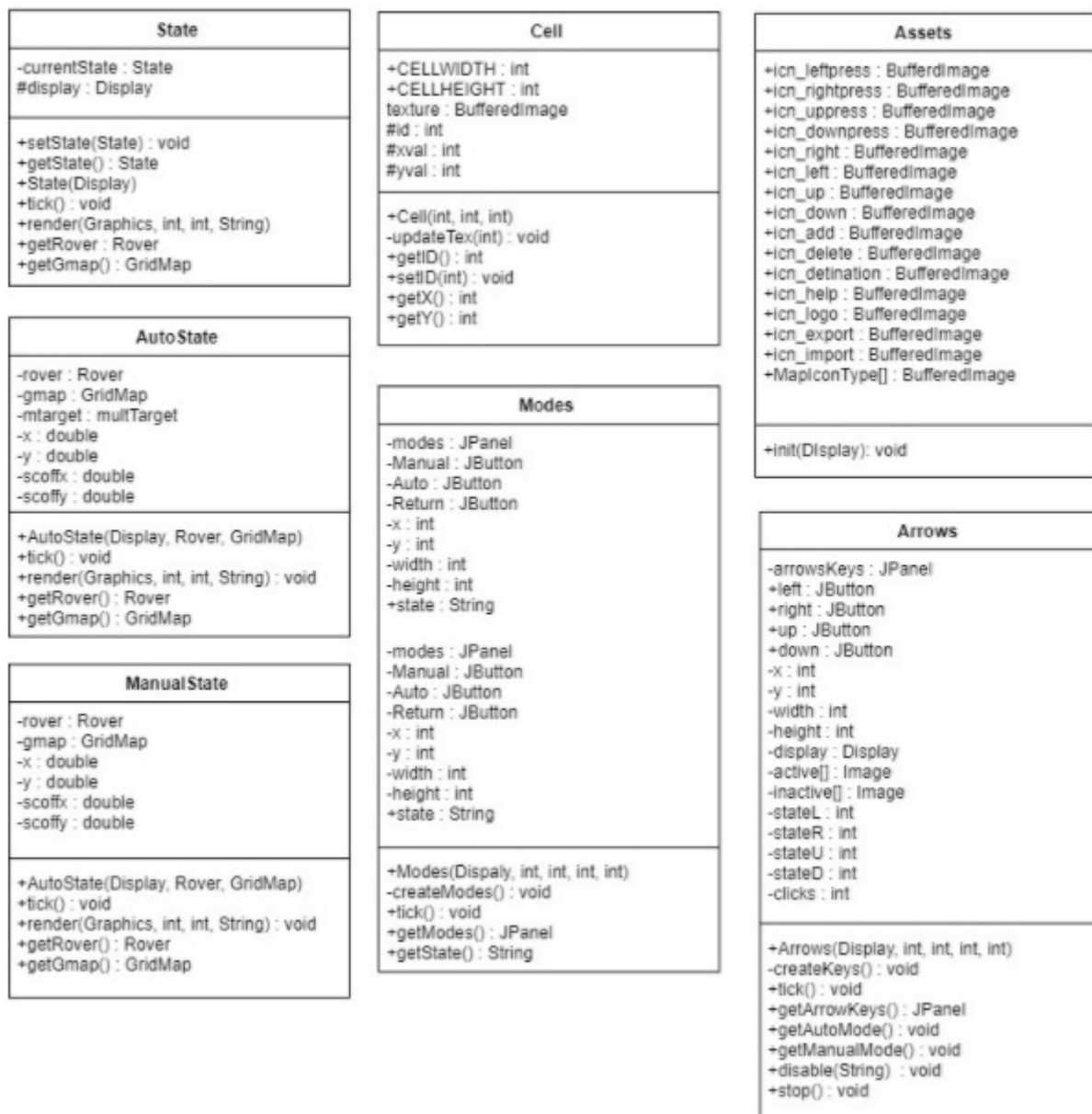


Figure 5b: GUI Class Diagram

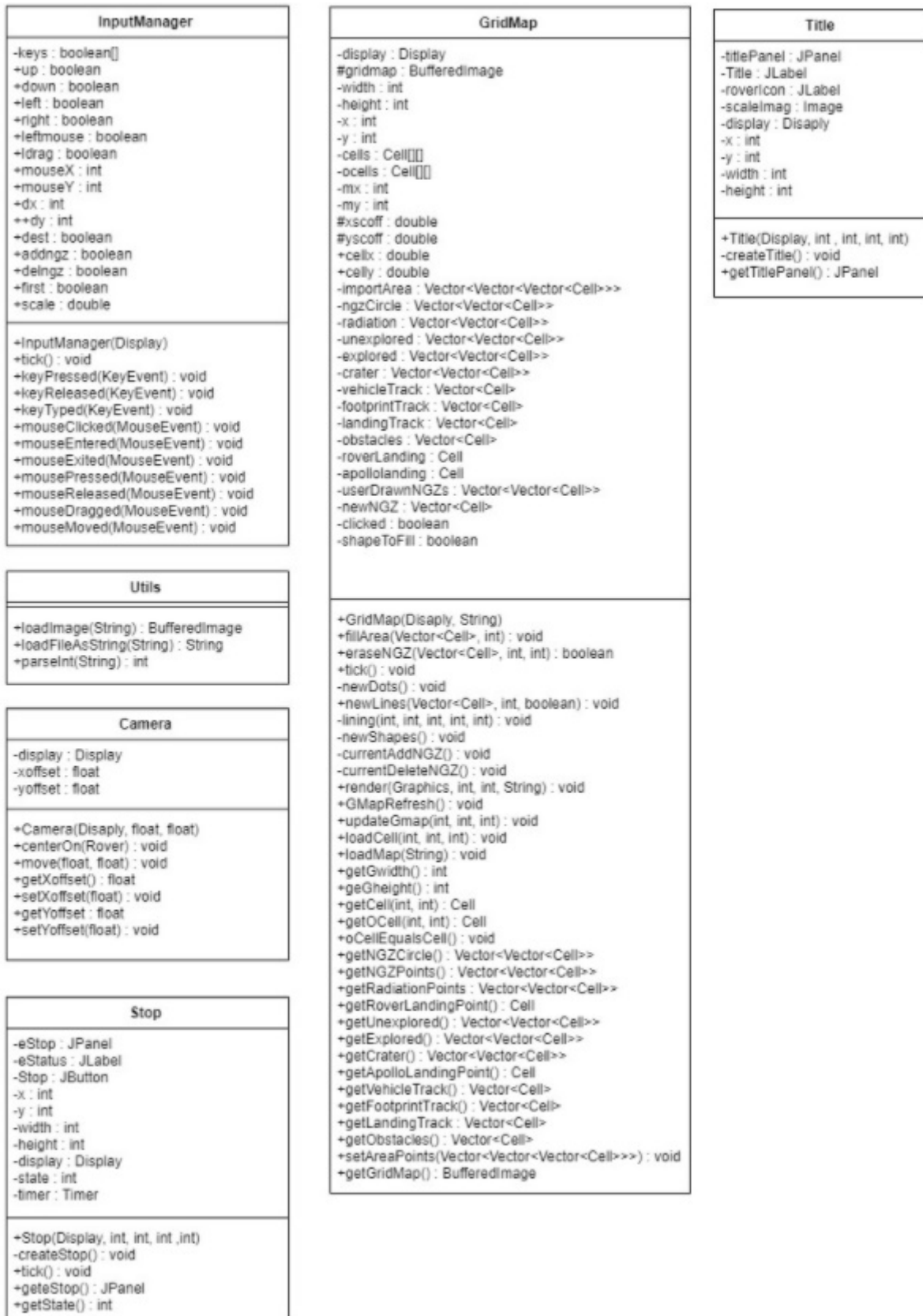


Figure 5c: GUI Class Diagram

5.2 State Diagrams

5.2.1 Rover

The state diagrams in figures 6a to 6d below illustrate the behaviours of the Rover in the various states it operates in. Contrary to a normal state machine, the states in the Rover are in a continuous cycle. I.E. each individual state is non blocking, and will immediately move on to the next state once all actions contained within the state are completed. This behaviour is required to ensure the states related to object detection and Rover safety can be performed in quasi parallel to the other behaviours. The first figure shows the overall Rover states, with the coloured states being super states which contain additional states within the overarching super state. The details of these super states are shown in the additional state diagrams below. These super states were included to simplify the state diagrams, and combine related states under a single theme.

Manual Movement Mode super state, figure 6b, handles the Rover states when the Rover is taking manual user movement commands. This super state is fairly simple and has little complex logic in the implementation.

The React to Object super state, figure 6c, handles all the map and 3D object detection using the Rover's sensors. It then performs the appropriate reactions to move the Rover into a less hazardous state if required and updates the internal object map for the Rover. It will also interface with the navigation components to re-generate the Rover's path. This action is performed here as the Rover will be in a stationary state that allows for heavy processing tasks which block execution to be performed without endangering the Rover. This state only occurs when the Rover has detected a feature in the map.

The automatic movement super state, figure 6d, handles goal selection, path following, and pathfinding (in some rare cases). The state diagram for this super state is quite simple and only contains high level states, as much of the implementation which handles these action comes from the lejos libraries. The details of the implementations of these libraries is unknown, so the internal states used in them cannot be shown in this diagram. This is the main state of the Rover, and the one it will spend the most of its time in.

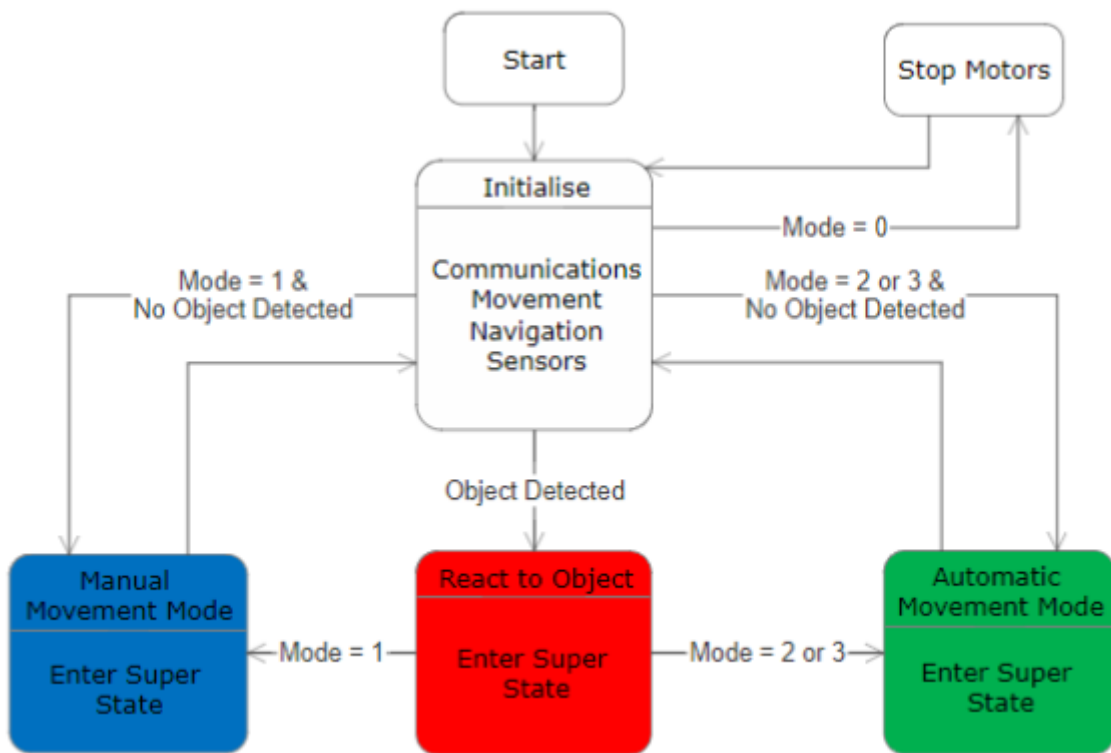


Figure 6a: Rover State Diagram

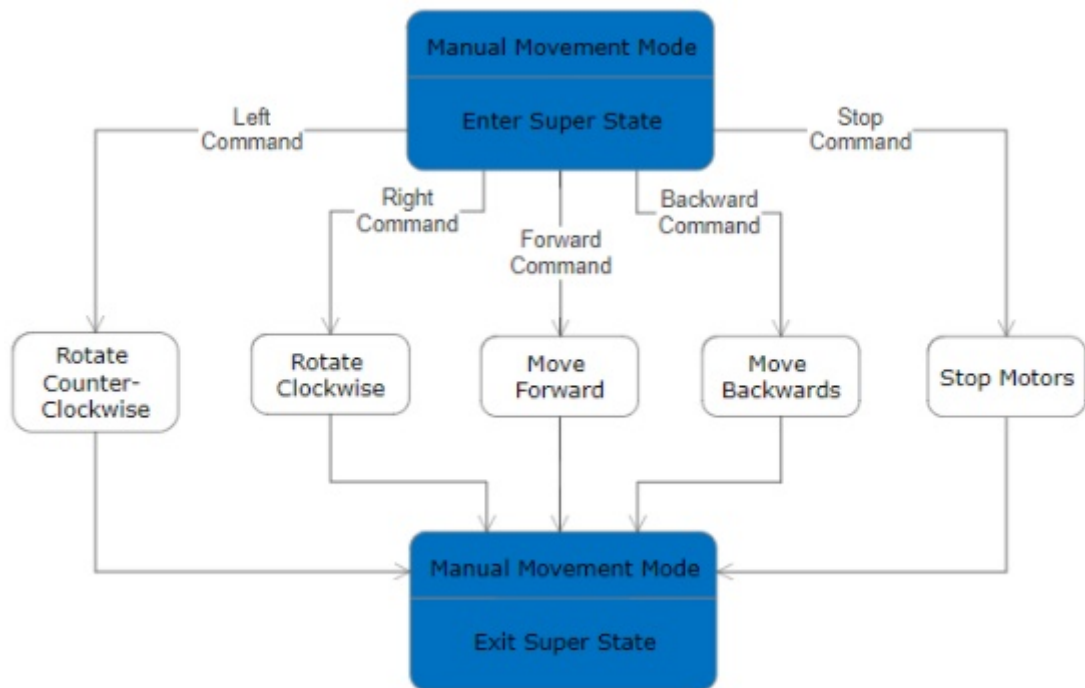


Figure 6b: Rover State Diagram

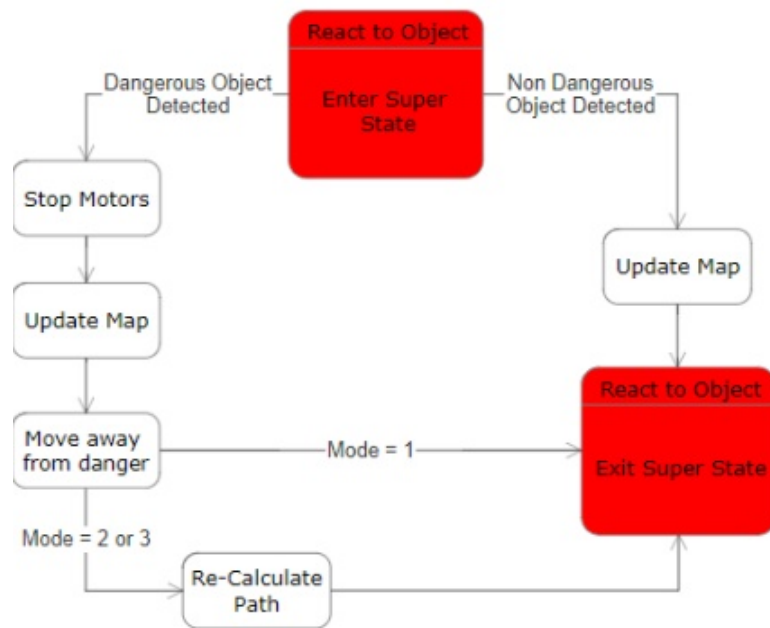


Figure 6c: Rover State Diagram

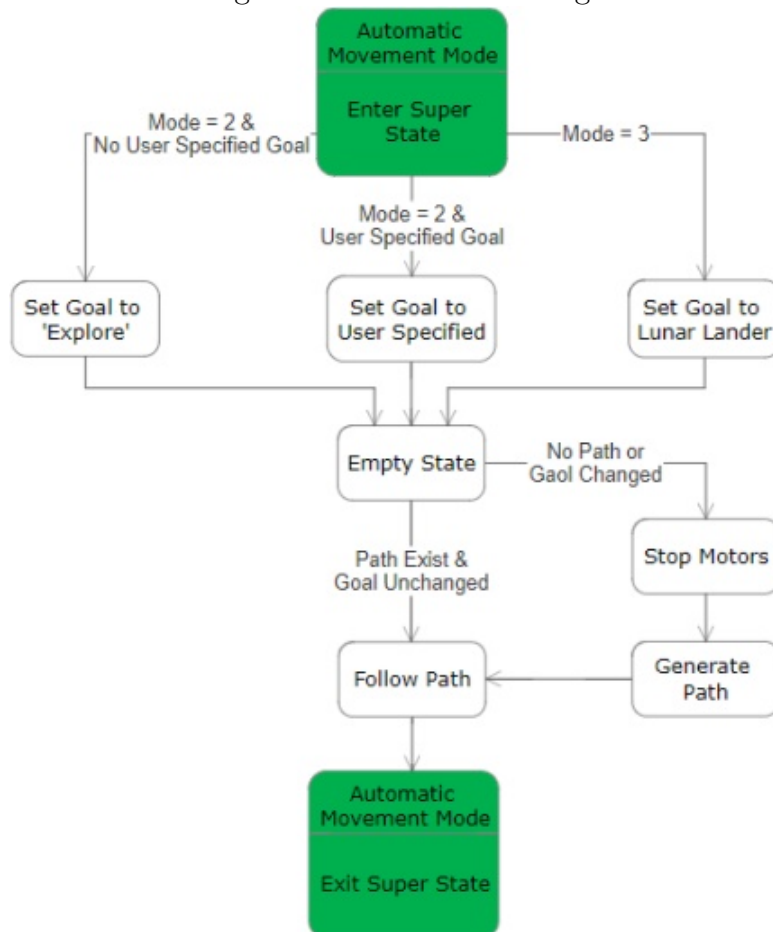


Figure 6d: Rover State Diagram

5.2.2 User Interface

The Normal State is the default state when none of the toolbar options are triggered (import, export, drawing NGZs, removing NGZs, setting destinations, help, status bar, legend options). In this mode, all control panel actions are available, and the user can pan (click and drag) the main Rover Map with a mouse.

The import and export states are triggered when the user clicks on the import or export buttons. A file explorer will be opened, and none of the GUI features will be available before the file explorer is closed by choosing a valid destination, closing the explore window by the top right close button, or clicking cancel.

The NGZ drawing mode is triggered when the corresponding button is pressed. In this mode, a mouse click (without releasing it) on the main Rover Map will allow the user to draw NGZs with mouse drags. And on mouse release the initial pixel clicked will be connected with the last to form an enclosed shape, and allow pixels within to be converted to NGZ cells.

The NGZ removing mode is quite similar to the NGZ drawing mode, while the user can remove user drawn NGZs with single clicks.

Destination setting mode will allow the user to set goals for the Rover to reach on the Rover Map with mouse clicks. There can be three destination points at most. Any extra clicks will remove the earliest drawn goal before getting a new one when there are already three. Finally, whenever the Rover reaches one destination, its corresponding mark will be removed from the Rover Map.

The help button, when clicked, will open the User Manual PDF. It is not necessarily a status, as the PDF will be opened with another application, but it will pause the Rover's current actions by switching to manual mode automatically. The status bar, when clicked, will also bring up a pop-up window so the user's behaviours are also limited this way.

The emergency mode has three states, rather than two to most users understanding. The first is the off mode, where the Rover can move freely. After a click on the emergency button, the emergency mode will be triggered, where the Rover will not be able to perform any action at all. Then in emergency mode, on a single mouse click, a counter will start. If a next click registers within 1.5 seconds, then the emergency mode will be disengaged. Else, when 1.5 seconds passed without receiving another click, it will stay in emergency mode and will require another click to start the counter again. This is how the double-click is implemented.

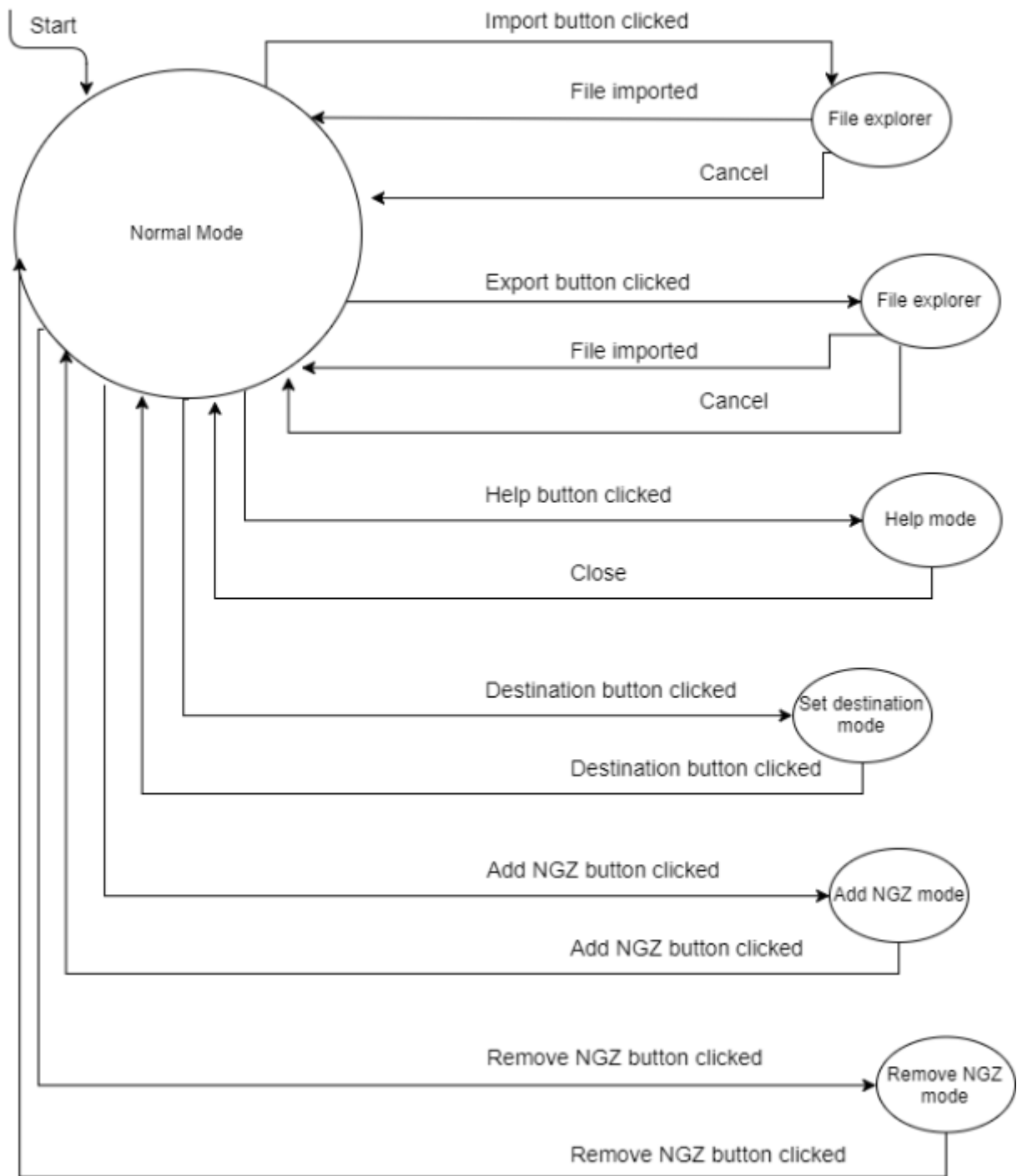


Figure 7a: GUI State Diagram

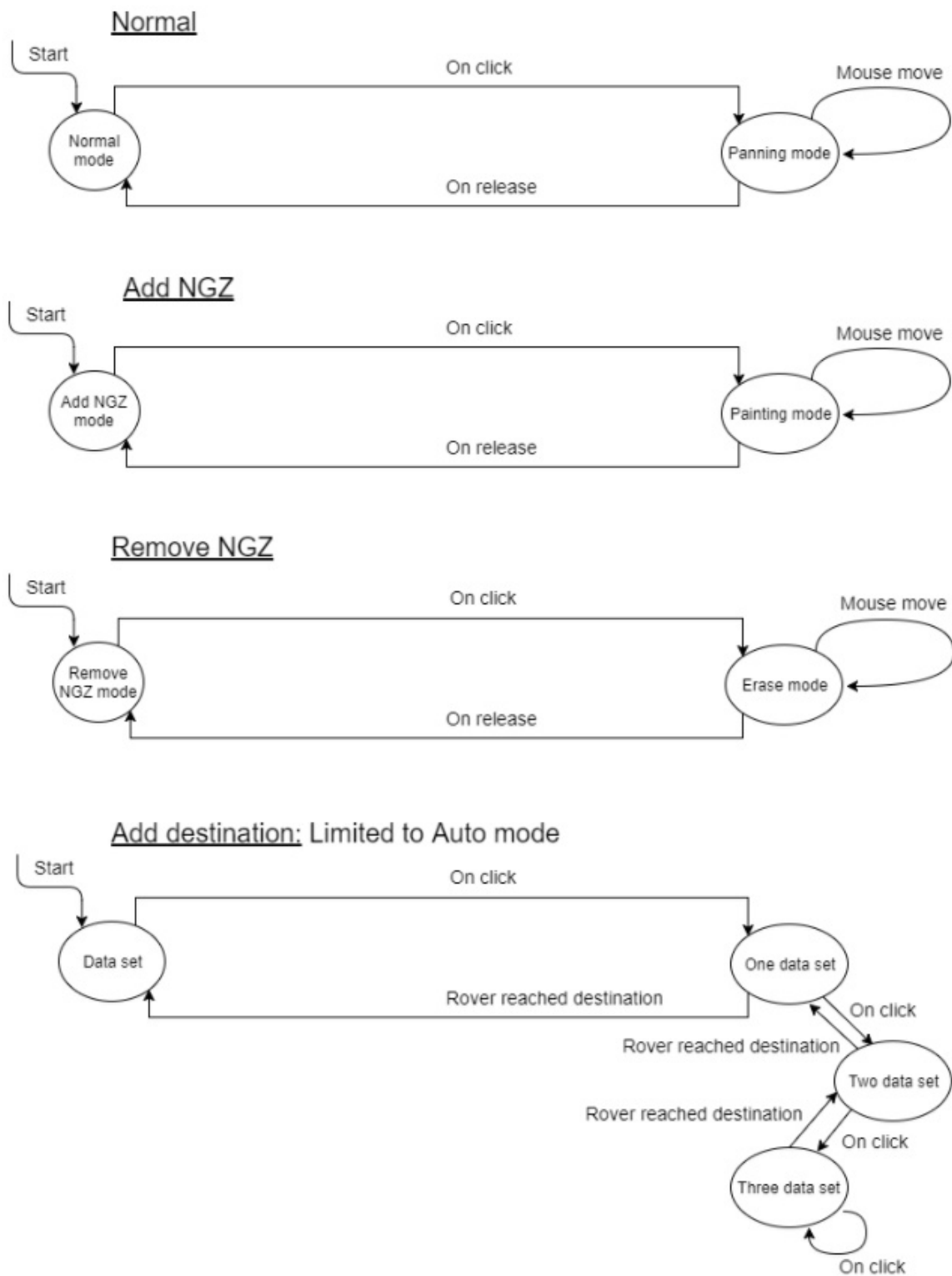


Figure 7b: GUI State Diagram

Control panel state diagram

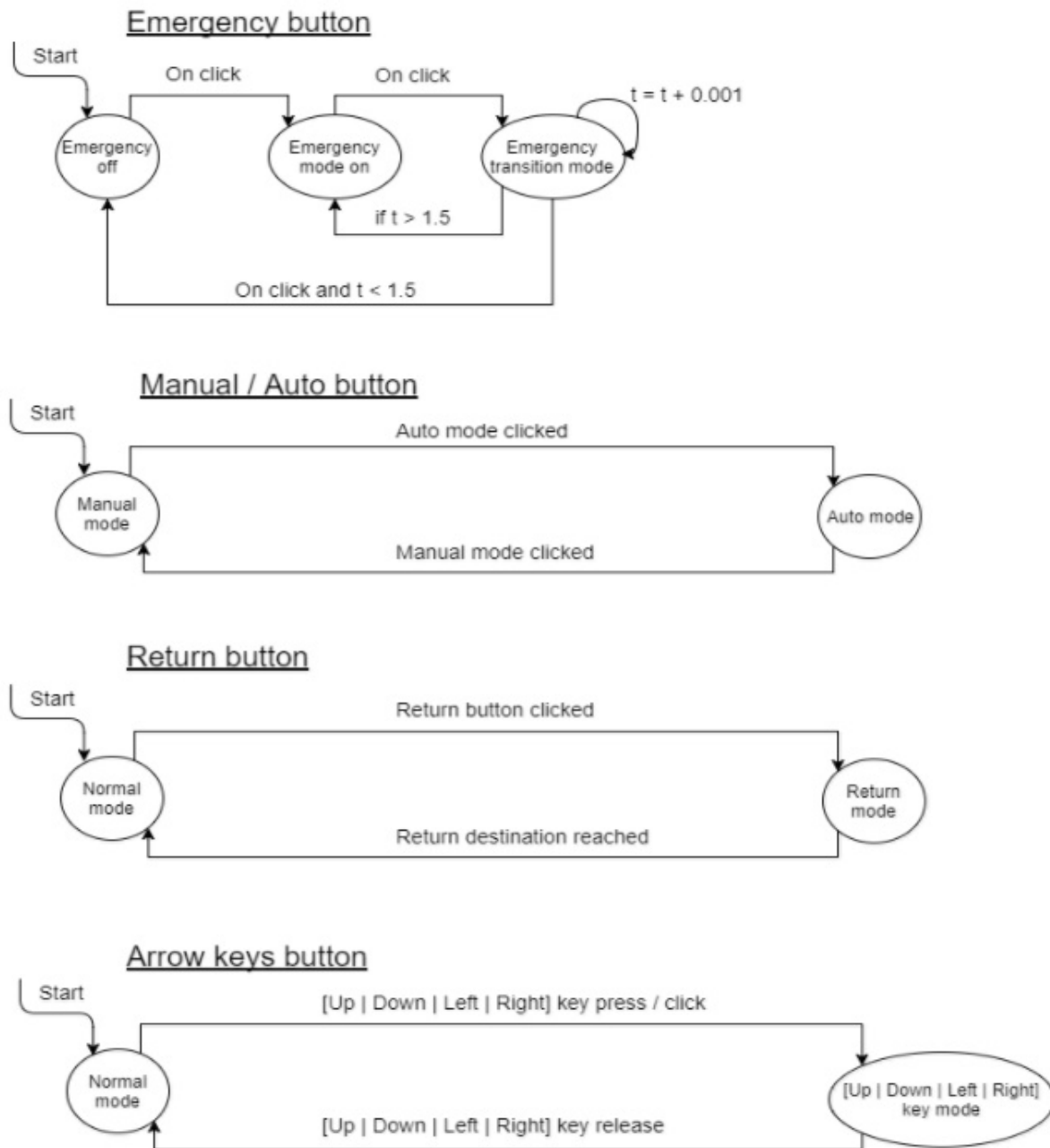


Figure 7c: GUI State Diagram

5.3 Interaction Diagram

The diagram below in figure 8 shows the data flow throughout each iteration of the main loop in the program. Each labelled arrowed line represents a data flow between individual classes in the Rover. The vertical positioning represents the timing of the flow, with higher flows occurring earlier in the loop. There are also loop-back flows which represent key data flow between sub components within each class.

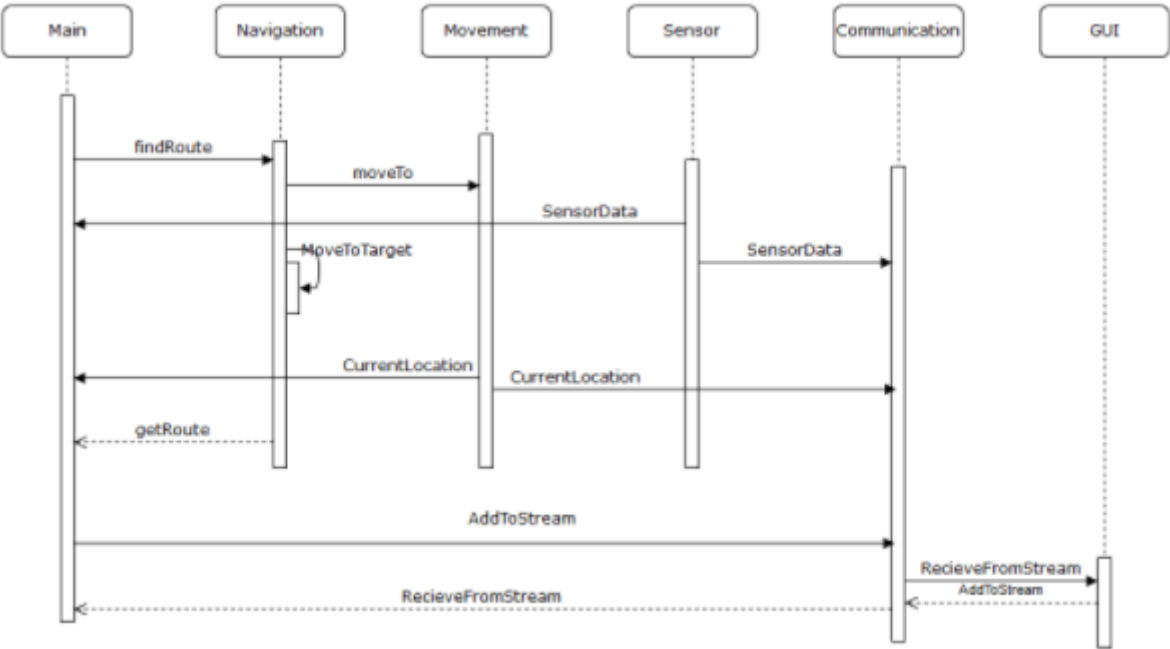


Figure 8: Data Flow and Interaction Diagram

6 Human Interface Design

6.1 Overview of the User Interface

The user interface for the Rover is a remote control panel that is run on computers with monitors. Users can interact with it by a standard keyboard and mouse set.

The user interface's main components are:

(Refer to the next section for detailed explanations of the elements on the user interface)

A. **Rover Map**

A 2-dimensional representation of the real world area that the Rover surveys. It will be updated in real time (though with some inevitable delay), and it is the major means for the user to keep track of the Rover's mapping progress.

There are various operations associated with this Rover Map, such as designating a point for the Rover to go to, defining NGZs for the Rover, zooming in and out of the map, panning, etc.

B. **Mini Map**

A radar map very similar to the Rover Map, but always zoomed in, and the Rover is always fixed at the centre. It allows the user to always be aware of the Rover's immediate surroundings, so that the user won't miss any items of interest that the Rover passes by.

C. **Legends**

Key entries for elements on the Rover Map and Mini Map. Users can define how colours are interpreted by the Rover by changing these legends.

D. **Main control panel**

This is where the user issues commands that directly affect the Rover's behaviour. The user can let the Rover survey the area by itself with predefined algorithms in auto mode, or switch to manual mode to control the Rover directly with the arrow keys. There's also an emergency button to freeze the Rover in case its self protection program didn't suffice.

E. **Toolbar**

Users can choose to import and export maps, toggle the mode to designate a point on the Rover Map for the Rover to travel to, toggle the mode to draw NGZs, toggle the mode to remove NGZs, and open the user manual.

F. **Status Bar**

A status bar to display the Rover's current behaviour/result. When it's clicked, a window containing the full log of past behaviours/results.

6.2 Detailed Design of the User Interface

6.2.1 Rover Map

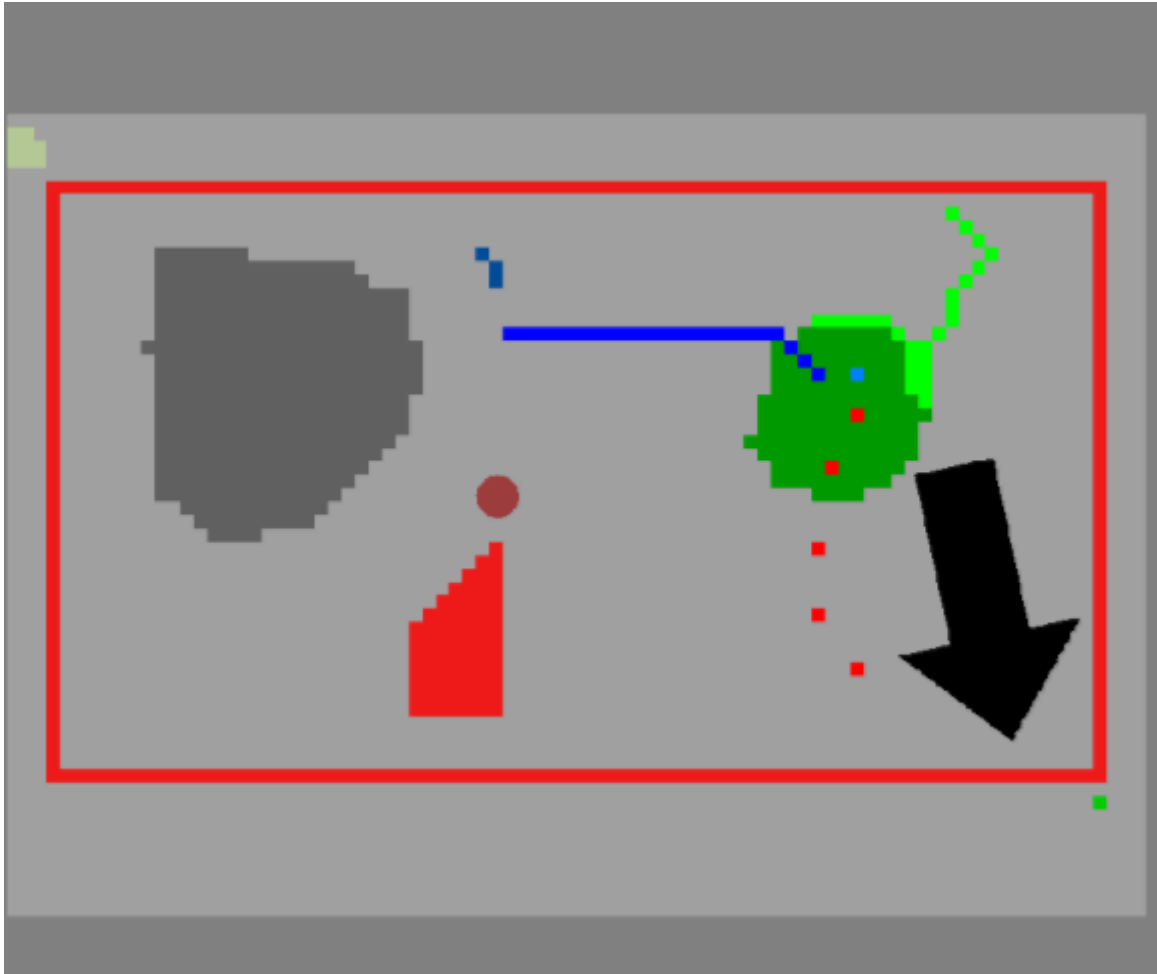


Figure 9a: GUI Map

The Rover Map is a graphical representation of the terrain around the physical Rover system based on the sensor data transmitted by the Rover. When initialized, an arrow icon is drawn representing the Rover in the centre, the Rover Map also allows the user to utilise various functions such as:

- **Setting Destination** - The function to set the destination of the Rover is enabled from the Toolbar when the Rover is in auto mode. When it's enabled, the user is able to designate up to three points on the Rover Map. Designated points are targets which the user wishes the Rover to move towards and override the Rover's self-exploration protocol.
- **Add/Remove NGZ** - Functions to add and remove NGZ is enabled from the Toolbar. When add NGZ is enabled, the user is able to draw NGZ areas into the map using the mouse. This changes the cell colour on the map to one which represents the NGZ. Enabling remove NGZ function will allow the user to use the mouse to revert any user drawn NGZ cells back to what they were before the changes.

- **Zooming** - The Rover Map has a zoom features which allows the user to resize the map to what they like. The zooming is controlled using the mouse wheel or the plus and minus keys on the number pad.
- **Panning** - The user is able to pan the map by clicking the left mouse button and drag it in the desired direction. Panning is only possible when the destination and NGZ functions are disabled. There will be an option on the toolbar to centre the view back to the Rover.

6.3 Mini Map

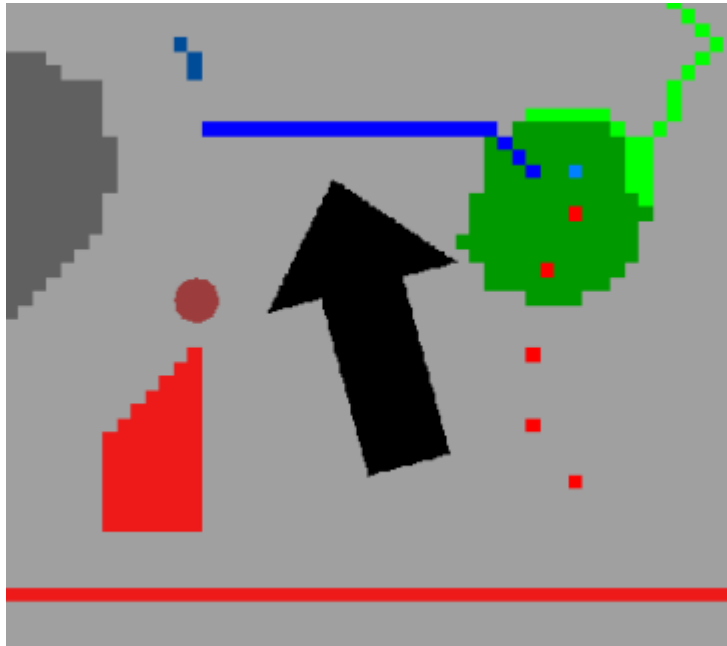


Figure 9b: GUI Mini Map

The Mini Map displays the map with the Rover icon being zoomed on and centrally fixed. The elements presented on the Rover Map are also visible on the Mini Map; however, the Mini Map cannot be used to add map functions. Its intended purpose is to allow the user to keep track of the Rover and its surroundings, while Rover Map has functions, such as zooming and panning, that potentially allows the user to lose track of the Rover.

6.4 Legend


 Rover	 Obstacle
 Destination	 Radiation
 Explored	 Rover landing
 Unexplored	 Apollo landing
 Circle NGZ	 Vehicle track
 NGZ	 Footprint track
 Crater	 Landing track

Figure 9c: GUI Legend

The Legend in the GUI like with most maps identifies the features that are represented in the map by symbols and colours. The GUI's legend comes with a feature that allows the user to change the colour scheme of the what is displayed on the map which is done by clicking on the specific element. When an element is selected, a second window opens, displaying the possible colour options an element can have.

6.4.1 Main Control Panel

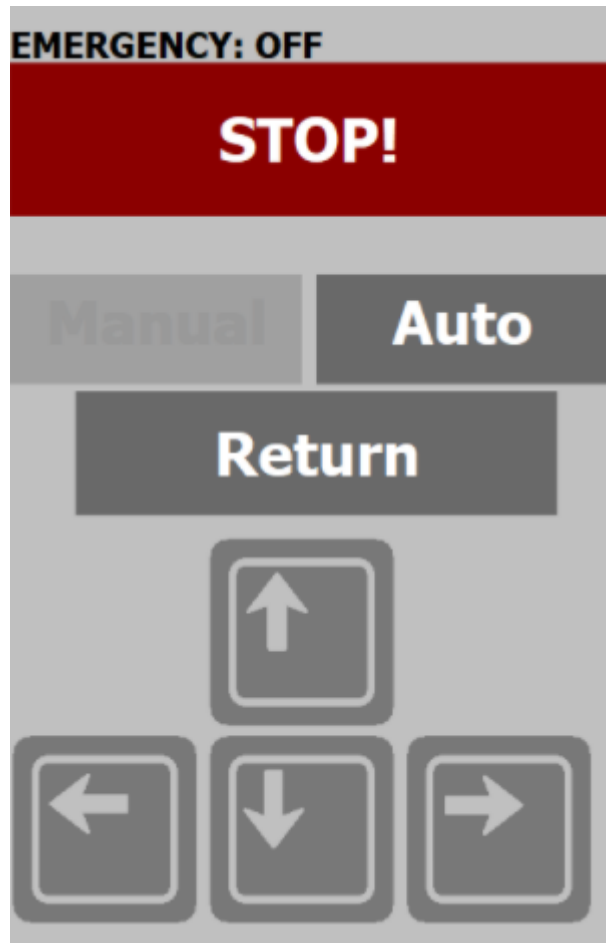


Figure 9d: GUI Control Panel

The Main Control Panel consists of buttons that allow the user to issue commands directly to the Rover. All buttons can be pressed using the mouse left click, however, the arrow buttons can also be pressed by using a computer's arrow keys. Each button send different commands to the Rover which tell it to perform certain actions:

- **Stop Button** - The stop button issues an emergency stop command to the Rover which tells it to cease all operation and operate the Rover in manual mode. Depending on whether an emergency stop command has been issued, there may be some text above the button. The command is disabled when the button is double clicked within 1.5 seconds. Otherwise, the Rover will stay in emergency mode.
- **Manual/Auto Buttons** - Rover has two main states during operations; a manual mode and an automatic mode which are controlled by these buttons. Aside from letting the Rover know which mode it is in, it will enable/disable some features in the GUI. During manual mode, arrow buttons are enabled and set destination button is disabled, automatic mode will have the arrow buttons disabled and set destination button enabled.

- **Return Button** - The return button sends a command to Rover to go back to its starting position or landing site. The Rover has an inbuilt function to return once it has completed surveying an area, the return button is to allow the user to force the Rover to return when area has been surveyed to satisfactory degree.
- **Arrow Buttons** - Arrow buttons allow the user to control the Rover's movement while in manual mode. The up button tells the Rover to go forward and the left/right tells the Rover to rotate left/right. The down button for the Rover is slightly different in that it will tell the Rover to trace back its movement, this means it will play back its logged movement in reverse.

6.4.2 Toolbar

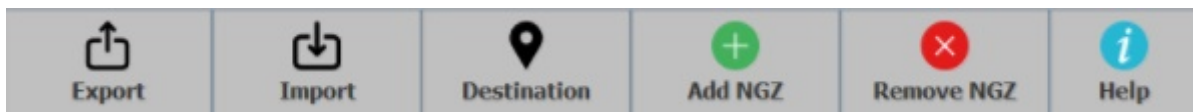


Figure 9e: GUI Toolbar

The Toolbar allows the user to perform various operations depending on which button is pressed, most operations are related to the Rover Map. The Toolbar consists of:

Export/Import Buttons - The export/import buttons allow user to either save the current map data to xml file in a DTD structure or read and display an existing xml file. When either button is pressed, a window will pop up, presenting folder type to allow the user to navigate to the folder and xml type files to save/load the the map data.

Destination Button - The destination button can only be used when the Rover is currently in auto mode. The destination button allows the user to set points on the Rover Map when enabled. Clicking the destination button again or the other toolbar buttons will disable the adding destination button mode.

Add/Remove NGZ Buttons - These buttons allow the user to draw/erase user drawn NGZs on the Rover Map when enabled. Buttons are disabled by pressing either of the add/remove or destination button.

Help Button - The help button in the Toolbar opens the user manual for the GUI which has details explaining all the function the GUI is capable of utilising.

6.4.3 Status Bar

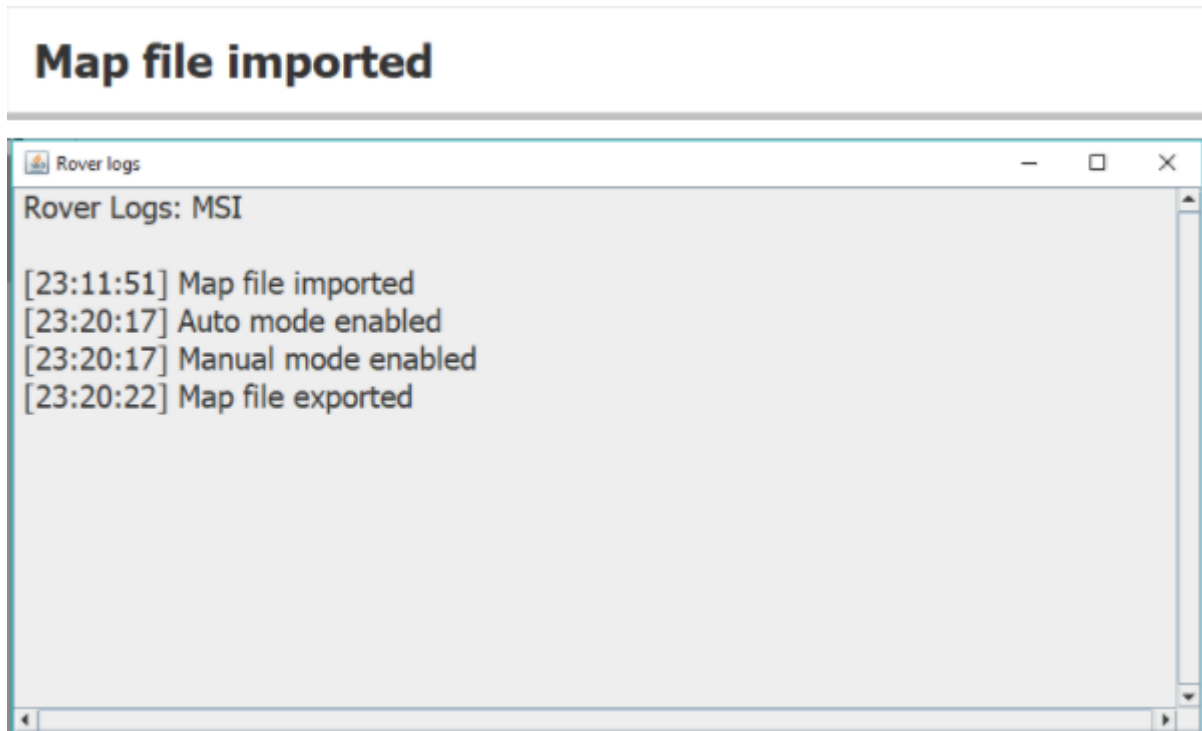


Figure 9f: GUI Status bar

The Status Bar displays a text message which informs the user of the current state of the GUI and the Rover. The displayed text changes based on what buttons are being pressed by the user. When the user clicks on the Status Bar, a separate window opens up displaying a log of what states occurred along with a time stamp showing when the state occurred. The log only keeps the record of states since the program has been opened.

7 Resource Estimates

7.1 Minimum Requirements

Generally speaking, a computer would be suitable for designing or operating this project if it has the capability to run all of the necessary software, such as a modern operating system and Eclipse. Below are some of the minimum computer standards found among the design team.

Operating System: Windows 7/10 32/64 bit

RAM: 4 GB

CPU: AMD A6-6200 APU with Radeon (TM) HD Graphics

CPU Speed: 2.0GHz

Graphics Card: AMD Radeon HD8400

Storage: 1 MB

Java version: Java SE 1.7.0 80

Eclipse Version: Eclipse Mars 4.2

7.2 Recommended Requirements

While a user could probably make do with the minimum required standards for their computer to utilise the Lunar Rover project, or indeed to design it, a higher standard of performance is necessary for true efficiency. A respectable set of standards found commonly has been listed below that would be desirable for any user.

Operating System: Windows 7 / 10 32/64-bit

RAM: 8GB

CPU: Intel(R) Core(TM) i5-4200U

CPU Speed: 2.6 GHz

Graphics Card: Intel(R) HD Graphics 4400

Storage: 10 MB

Java Version: Java SE 1.8.0144

Eclipse Version: Eclipse Oxygen 4.7.0

A Definitions

SDD	Software Design Document
NGZ	No Go Zone
GUI	Graphical User Interface
DTD	Document Type Definition
PC	Personal Computer
SRS	Software Requirement Specification
EV3	Lego Mindstorms EV3 Robot