Software Design Document

for

Lunar Rover Mapping Robot

Version 2.0

Group: UG12

THE UNIVERSITY
of ADELAIDE

Contents

Revision History

Version	Date	Reason for changes
1.0	3/10/2017	Initial draft
2.0	28/10/2017	Final version

List of Figures

1 Introduction

1.1 Purpose

This document details the software design for the Lunar Rover Mapping Robot Project. It presents the detailed description of the system and will explain system architecture and module design. This document is a guide for building the system of the project.

1.2 Scope

This document contains a complete description of the design of the Lunar Rover Mapping Robot Project to satisfy the requirements as specified in Software Requirement Specification.

1.3 References

[1] Google Lunar XPRIZE. 2017. *Google Lunar XPRIZE*. [ONLINE] Available at: `https://lunar.xprize.org`. [Accessed 4 September 2017].

1.4 Overview

The Software Design Document is divided into 7 sections. The sections are listed below:

1. System Overview
   This section gives the overview of the system.
2. System Architecture and Components Design
   This section describes the whole architecture and component designed of the system.
3. Data Design
   This section describes how the data is stored and its representation in the system.
4. Design Details
   This section gives the detail of the class design in the system.
5. Human Interface Design
   This section describes the design details of the User Interface.
6. Resource Estimates
   This section lists the requirements of the computer resource estimation for operating the system.
7. Definitions, Acronyms, and Abbreviations
   This section lists all the definitions, acronyms and abbreviations that used in this document

1.5 Constraints

The biggest constraint in the design and implementation of the system is time. The allocated time is roughly 3 months for design, development, testing and documentation of both the hardware and software components of the project. A significant portion of the allocated time would be spent on the team becoming familiar with the development tools to aid in the development of the system. Furthermore, the team only have access to one robot unit which makes the

development of the system a challenge due to the fact that each sub group in the team doing different parts of the project needs to take turns in gaining access to the robot.

2 System Overview

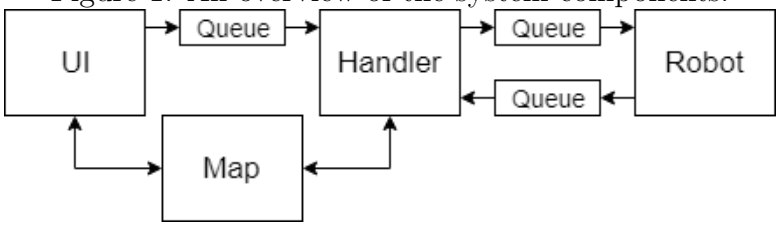
The system is comprised of three modules: the UI, Handler, and Robot. The UI provides a graphical user interface for receiving commands from users and showing information which is fetched from the Handler. The Handler handles event commands, sensor data transfers and provides path finding. The Robot manages the low-level functionality and sends sensor data to the Handler. Queues are used to communicate messages between each module.

3 System Architecture and Components Design

3.1 Architectural Description

The software consists of three primary modules, named **UI**, **Handler**, and **Robot**, in addition to a **Map** data structure. The UI is responsible for displaying information to the user through a graphical user interface (GUI), and providing user input information to the Handler. The Handler is responsible for taking information from the UI and Robot modules, making high-level decisions, and communicating commands to the Robot. The Handler is responsible for tasks such as pathfinding and processing sensor data to be stored into the Map data structure, which holds the information about the surveyed area. The Robot component translates high-level motion commands to actuation of motors, and communicates processed sensor data back to the Handler.

Figure 1: An overview of the system components.



Commands and messages are communicated between modules through the use of queue objects. This is so the system can easily be modified to be multithreaded if required.

3.2 Component Decomposition Description

The main components (**UI**, **Handler**, and **Robot**) have been divided up into several sub-components, as well as several small components that exist outside of or are shared between them. The *Application* component controls each of the main components, setting them up at system startup and managing their behaviour. The *Map* component contains information about the surveyed location and is shared between parts of the UI and Handler. Three queues are used to communicate between high-level modules: *UIEventQueue*, *HandlerToRobotQueue*, and *RobotToHandlerQueue*.

3.3 Detailed Components Design Description

3.3.1 User Interface

Purpose: Satisfies user interface requirements (SRS Section 5.1).
Function: This component displays the user interface for the user. It communicates UI events, such as a key or on-screen button press, to the Handler module.
Subordinates: UIEventQueue, Map, Handler
Dependencies: Must place UI events onto the UIEvent Queue.
Interfaces: The component receives input from the user, and reads information from the Map and Handler. The component sends information to the Handler through the UIEventQueue.

3.3.2 Handler

Purpose: Partly satisfies user requirements SF01 (Manual Control) and SF02 (Autonomous Control), along with the Robot.
Function: Processes sensor data received from the Robot and UI events received from the User Interface. Updates the Map data structure and communicates commands to the Robot.
Subordinates: Map, UIEventQueue, HandlerToRobotQueue, RobotToHandlerQueue
Dependencies: Must send commands to the robot through the HandlerToRobotQueue.
Interfaces: Sends and receives data to and from the Robot through *HandlerToRobotQueue* and *RobotToHandlerQueue* respectively. Receives data from the UI through *UIEventQueue*. Processes and stores Robot sensor data in the *Map*.
Data: Stored in the *Map* data structure. Also stores some robot properties, such as position and direction.

3.3.3 Robot

Purpose: Partly satisfies user requirements SF01 (Manual Control) and SF02 (Autonomous Control), along with the Handler.
Function: Received high-level commands from the Handler and translates them into motor actuation. Sends sensor data to the Handler.
Subordinates: HandlerToRobotQueue, RobotToHandlerQueue
Dependencies: Must send status and sensor data to the Handler through RobotToHandlerQueue.
Interfaces: Sends data to the Handler through RobotToHandlerQueue and receives data from the Handler through HandlerToRobotQueue.
Data: Stores handles for the component motors and various data describing the robot state, including current position and direction.

3.3.4 Map

Purpose: Satisfies user requirement SF03 (Mapping), along with the ColorSensorInterpreter and DistanceSensorInterpreter components.
Function: Stores the location of features detected by the Robot.
Subordinates: N/A
Dependencies: N/A
Interfaces: Allows methods to retrieve feature likelihoods (how probable there is a feature at a location) at a given position and set feature likelihoods at any position. Also provides some utilities to get colours at a position (using an assigned colour table for features) and get the most likely feature at a position.
Data: Stores a matrix of likelihoods for each possible feature in the map. The grid size of each element in the matrix (in metres) is specified when the Map is created.

3.3.5 UIEventQueue

Purpose: Aids in satisfying requirements through the UserInterface and Handler components, and the performance requirement NF-R02 (Real-Time Communication).
Function: Communicates UI input events to the Handler module.
Subordinates: UserInterface, Handler
Dependencies: N/A
Interfaces: Allows messages to be added and removed
Data: Stores the messages as a FIFO queue.

3.3.6 HandlerToRobotQueue

Purpose: Aids in satisfying requirements through the Handler and Robot components, and the performance requirement NF-R02 (Real-Time Communication).
Function: Communicates high-level commands from the Handler to the Robot.
Subordinates: Handler, Robot
Dependencies: N/A
Interfaces: Allows commands to be added and removed.
Data: Stores the commands as a FIFO queue.

3.3.7 RobotToHandlerQueue

Purpose: Aids in satisfying requirements through the Handler and Robot components, and the performance requirement NF-R02 (Real-Time Communication).
Function: Communicates robot state and sensor data from the Robot to the Handler.
Subordinates: Handler, Robot
Dependencies: N/A
Interfaces: Allows messages to be added and removed.
Data: Stores the messages as a FIFO queue.

3.3.8 ColorSensorInterpreter

Purpose: When used with the map, satisfies user requirement SF03 (Mapping).
Function: Translates sensed colours from the Robot's colour sensor into detected features.
Subordinates: Map
Dependencies: Must store interpreted data into the Map.
Interfaces: Provides methods to assign colors to features and translate a sensed color into a feature, and store that feature into a Map.

3.3.9 DistanceSensorInterpreter

Purpose: When used with the map, satisfies user requirement SF03 (Mapping).
Function: Translates sensed distances from the Robot's ultrasonic sensor into an obstacle at

a certain position.
Subordinates: Map
Dependencies: Must store interpreted data into the Map.
Interfaces: Provides a method to translate a sensed distance into a position in the Map, and store an obstacle feature at that position.

3.3.10 PathFinding

Purpose: This component when used with the Map component satisfies the user requirement SF02 (Autonomous Control).
Function: Finds the shortest path from the Robot's position to a destination point on the map while avoiding any impassable areas.
Subordinates: Map, Node
Dependencies: The found path must be within the map.
Interfaces: Receives a Cartesian coordinates of the current position of the Robot and the destination point from the Handler and returns the shortest path, which is a collection of Cartesian coordinates, between the two points.
Data: The estimated travel cost from the Robot's position to each inspected grid then to the destination point.

3.3.11 Node

Purpose: This component when used with the PathFinding component assists in satisfying the user requirement SF02 (Autonomous Control).
Function: Stores the estimated costs g (the cost to travel from the Robot's position to this node/point), h (the displacement cost between this point and the destination point) and f (the total cost, i.e. $h + g$) and its adjacent nodes which are used by the PathFinding component to find the shortest path.
Subordinates: Point
Dependencies: N/A
Interfaces: Calculates and stores the estimated costs and sends it to the PathFinding component when it is needed.
Data: The estimated costs and this node's adjacent nodes.

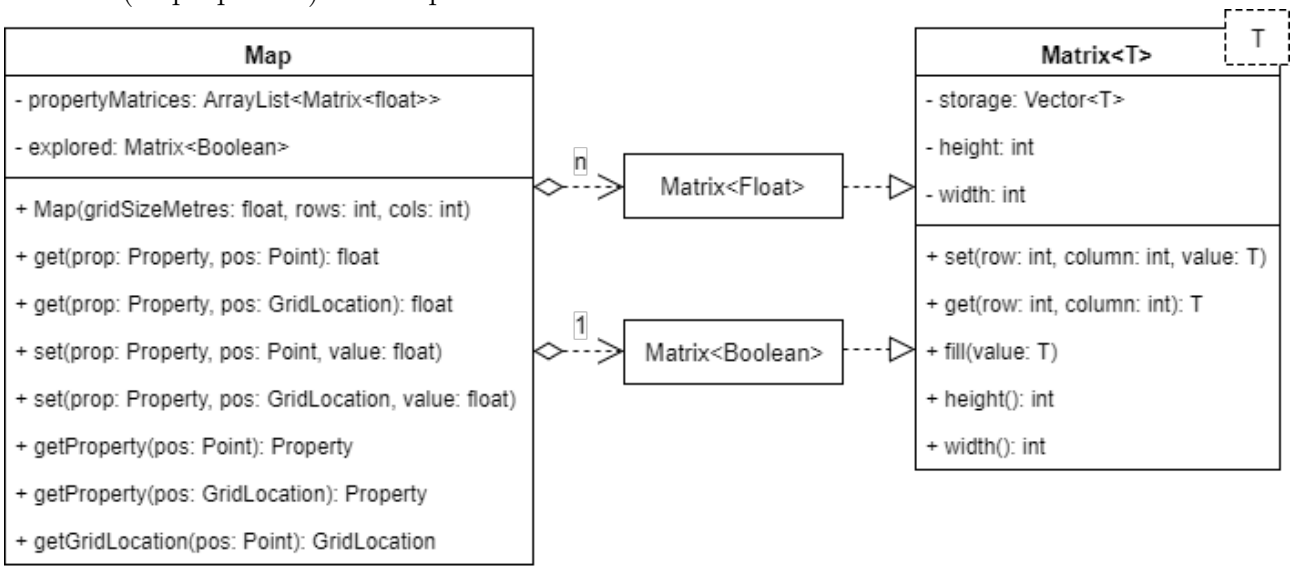
3.4 Architectural Alternatives

Aside from our modular, communication queue-based approach, we also considered a layered architecture. With this architecture, our independent modules (UI, Handler, and Robot) would be replaced with a stack of software layers from high-level (the user interface), through intermediate layers (the equivalent of the Handler module), with the lowest level being direct control of the actuation of the robot motors.

3.5 Design Rationale

We decided on using the communication queue-based architecture over the layered architecture due to the difference in the ease of switching each architecture to a multithreaded variant. When analysing the software requirements, we found that a multithreaded system may be required to stop high-level logic from blocking low-level controls and preventing quick responses to environmental changes at a low level, e.g. stopping the robot before it collided with an obstacle. A layered approach would be difficult to arrange in such a way that it could be easily split into several independent threads of execution. However, the approach of using communication queues means that our design can be modified to have any of the three main components (UI, Handler, and Robot) moved into a separate thread of execution using a thread-safe queue object. This means that our communication queue method is more robust to this architectural change if it was required.

Figure 2: The Map data structure represented as a UML class diagram. n is the number of features (or properties) the Map can store.



4 Data Design

4.1 Database Description

The system must store the data that the robot has sensed into some representation of the surveyed location. To do this, we have designed a *Map* data structure which stores a grid of likelihoods for each feature that is required to be detected by the robot.

4.2 Data Structures

The *Map* consists of a number of stacked matrices, one for each feature (or *property* in the codebase) that the robot can detect. Each matrix has the same dimensions. The surveyed location is divided up into a grid, and the elements in each matrix correspond with a square in the grid. When the Map is initialized, the grid dimensions (in *rows* and *columns*) and grid element size (in *metres*) are specified. Each element in each matrix contains a value varying between 0.0 and 1.0. This value represents a likelihood that a feature is present in this grid element. This allows the Handler logic to account for sensor error when utilizing the Map data.

The Map provides utilities to convert from points in metre coordinates to locations in the Map grid, and to retrieve and set values in the Map. It can also determine the most likely feature at a position.

5 Design Details

5.1 Class Diagrams

See figure 18 at the end of the document.

5.2 State diagrams

As the Handler and Robot are separate modules, they are presented as two different state machines. The UI does not have any state, as it is implemented through the Java event system.

Figure 3: The state diagram for the Handler component

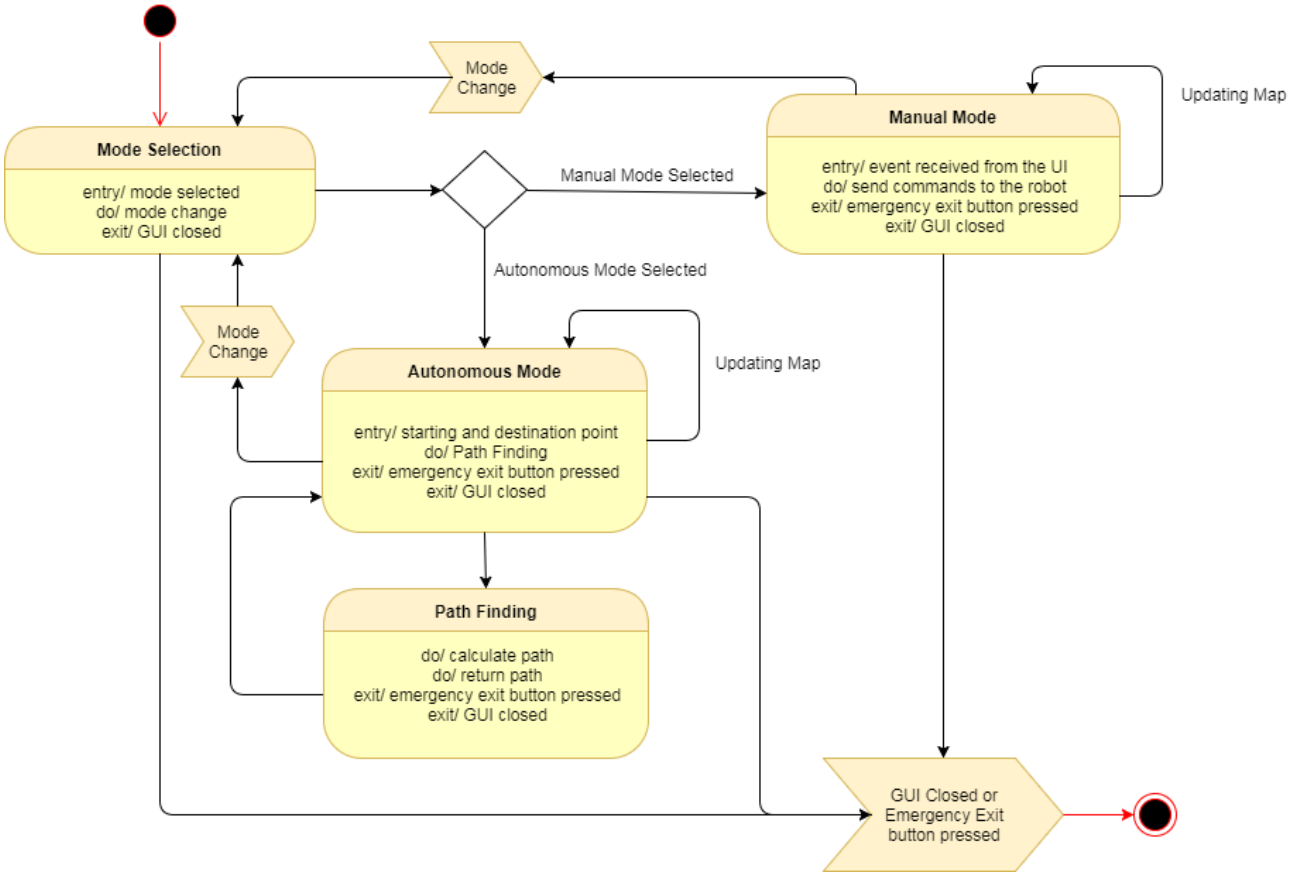
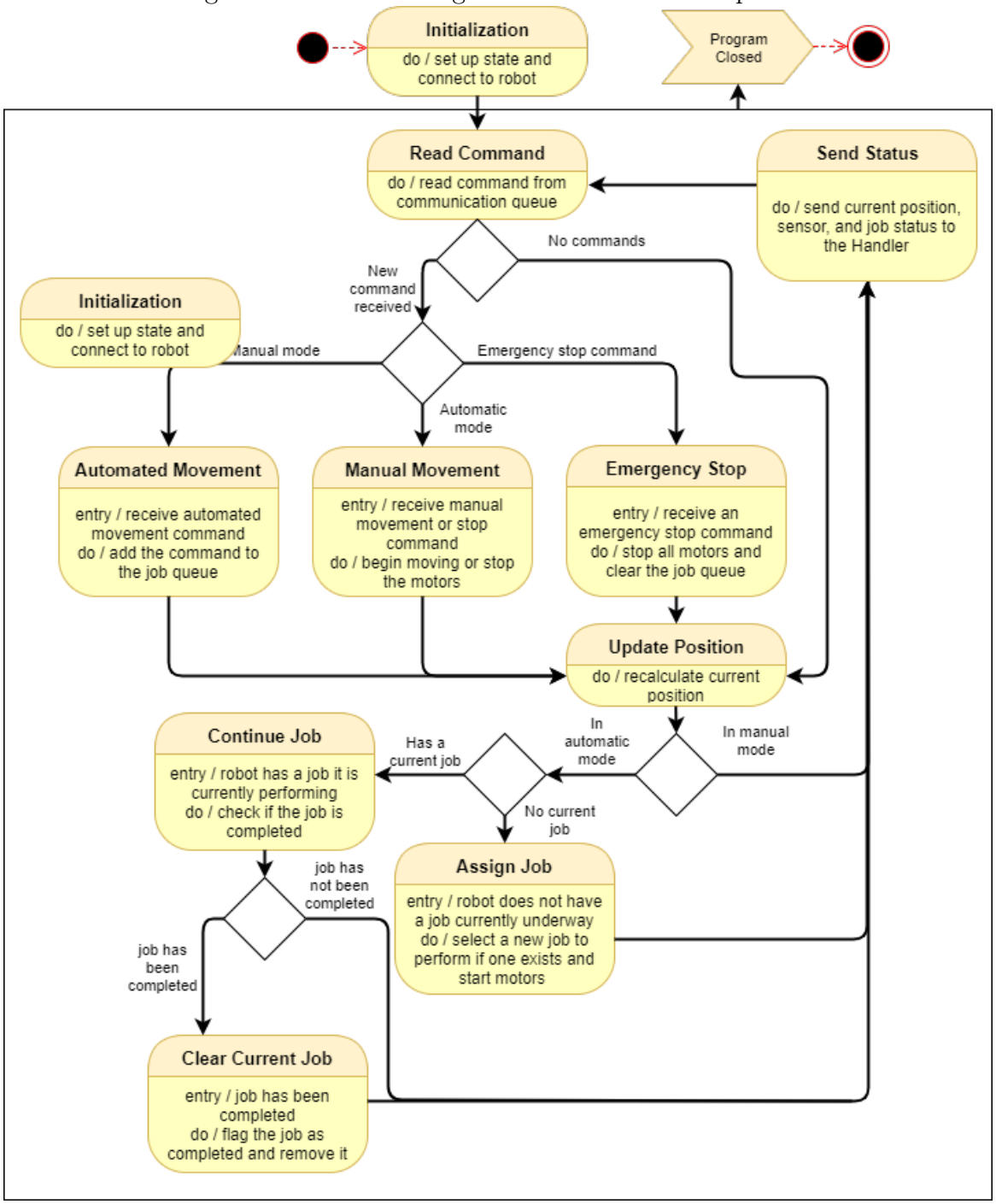
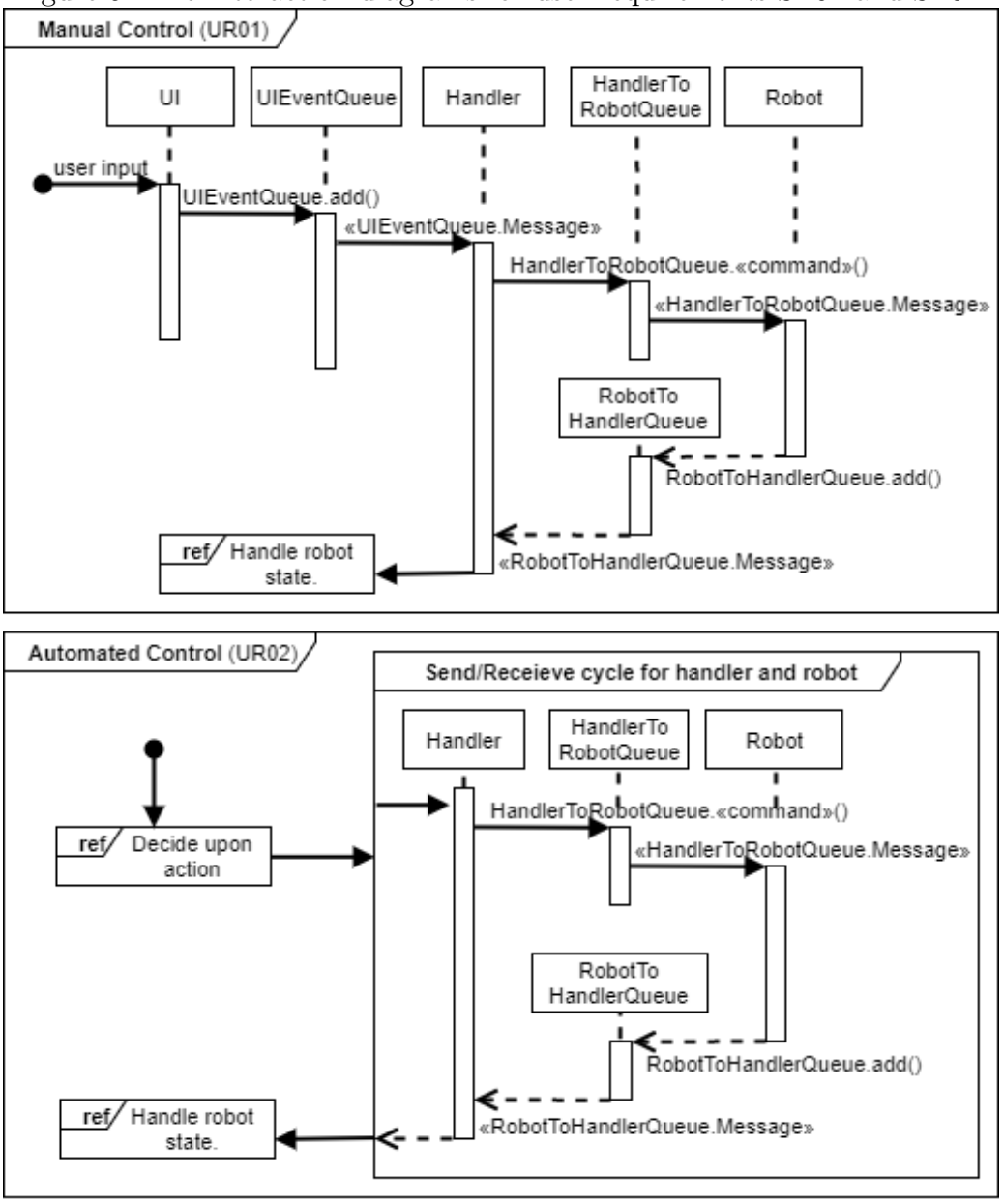


Figure 4: The state diagram for the Robot component



5.3 Interaction Diagrams

Figure 5: The interaction diagrams for user requirements SF01 and SF02.



6 Human Interface Design

6.1 Overview of the User Interface

The User Interface is implemented based on MVC (Model View Controller) software architecture pattern. User can interact with the software through the GUI, which facilitates user's utilization of program. The GUI provides buttons to allow user operating the software by clicking on it or press the key. And the data of map structure will be rendered to a region of GUI which will display the visualization of map data.

6.2 Detailed Design of the User Interface

There are two screens of GUI. One is connection screen and another one is the general screen. The connection screen is displayed at the launch of the program. It asks user to enter the IP address to connect robot. General screen will show up to the user once the connection is built.

- The Connection Screen of GUI has mainly two parts. The message panel will display the current status of the connection and the selection panel allows user to select the connection mode

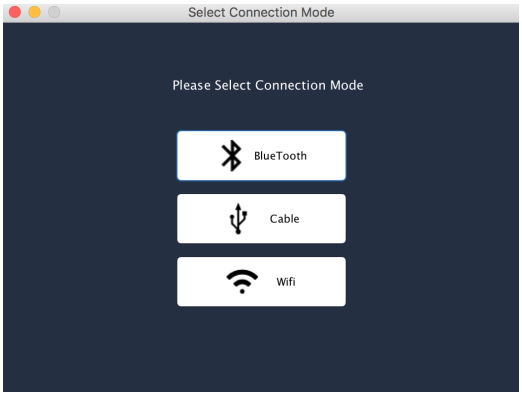


Figure 6: Connection Screen

  - "Bluetooth" button should be selected by click when the connection is via Bluetooth. Robot will have the IP address 10.0.1.1.
  - "Cable" button should be selected by click when the connection is via cable. Robot will have the IP address 10.0.1.1.
  - "WiFi" button should be selected by click when the connection is via WiFi. The robot will be assigned an IP address and it is displayed on the brick screen. It will pop up a prompt to ask User enter the IP address.

- The are mainly three sections on the general screen which are map panel, control panel and switch panel.

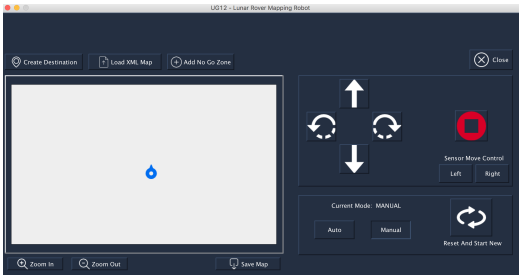


Figure 7: General Screen

- Control Panel - Robot movement control (SF01 Manual Control)

Figure 8: Moving Control

  - The button with up arrow will send forward command when user click on it or press up arrow key on the keyboard.
  - The button with down arrow will send backward command when user click on it or press down arrow key on the keyboard.
  - The button with left rotate arrow will send left rotate command when user click on it or press left arrow key on the keyboard.
  - The button with right rotate arrow will send right rotate command when user click on it or press right arrow key on the keyboard.

- Control Panel - Colour Sensor movement

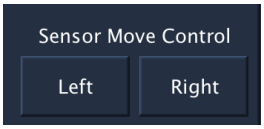


Figure 9: Colour Sensor Move Control

  - The left button will send left moving command to move left the colour sensor when user click on it or press "A".

  - The right button will send right moving command to move right the colour sensor when user click on it or press "D".

- Control Panel - Emergency Stop (SF01 Manual Control)

Figure 10: Emergency Stop

  - The button will send emergency stop command to the robot to stop the robot from movement immediately.

- Switch Panel - Automatic and Manual Switch (R01 Manual Control and R02 Autonomous Control)

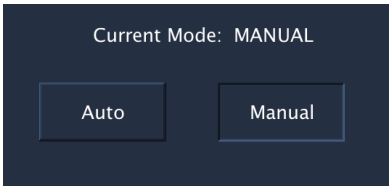


Figure 11: Automatic and Manual Switch

  - When manual button is on click, it will switch the robot to the manual mode which allows user to manually control the robot and enable sending robot movement commands.
  - When Automatic button is on click, it will switch the robot to the automatic mode which enable self path finding of the robot and disable sending robot movement commands.
  - The current mode can be viewed on the status window.

- Switch Panel - Reset and Restart

Figure 12: Reset and Restart

  - When the robot has reached the destination, The reset button allows User has the option to reset the figures of map.

- Map Panel - Map display region (SF03 Mapping)

Figure 13: Map Display

  - This region will display the map visualization by rendering the map data. It displays the robot position and map features. It also allow user to modify the map by marking no go zones or creating destination.

- Map Panel - Map Editing Buttons (SF03 Mapping)

Figure 14: Map Editing buttons

  - The map editing buttons are placed on the top of the map display region. They have the functionalities to allow user modify the map.
  - When user click on "Create Destination" button, it allows user to create a destination by clicking the specific point on map. Then the text on this button will become "confirm", user needs to click on it to confirm the destination.
  - When user click on "Load XML map" button, it allows user to load the data of DTD.xml into the map structure. The DTD.xml will also be rendered and displayed to user at the same time.
  - When user click on "Add No Go Zone", It allows user to mark the NGZs on the map. User can use the mouse to drag a shape on the map display region. Then the text on this button will become "confirm", user needs to click on it to confirm the NGZs.

- Map Panel - Zoom In/Out

Figure 15: Zoom In/Zoom Out

- One click on "Zoom in" button will enlarge the components displayed in the map 1.1 times bigger.
- One click on "Zoom out" button will narrow the components displayed in the map 1.1 times smaller.
- When the map is zoomed in, user can click on the map to move the camera.

- Map Panel - Save Map (SF03 Mapping)

Figure 16: Save Map

- "Save Map" Button allows user to save the data of current map. All features of the map will be rendered into format.

- Close

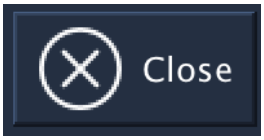


Figure 17: Close Button

- When use click on close button, it will send command to close all the motor and sensor ports and then exit the program.

7 Resource Estimates

The program is not computationally intensive, and does not require a large amount of space. Any modern computer system should have sufficient CPU, GPU, RAM, and HDD space to run this program. 500MB of RAM, and 200MB of HDD space should be sufficient.
The program requires Java 7 to run, and a WiFi adapter to remotely connect to the EV3 brick used in the Lunar Rover.

8 Definitions, Acronyms, and Abbreviations

CPU: Central Processing Unit
GPU: Graphics Processing Unit
GUI: Graphical User Interface
HDD: Hard Disk Drive
RAM: Random Access Memory
SDD: Software Design Document
SPMP: Software Project Management Plan
SRS: Software Requirements Specification
UI: User Interface

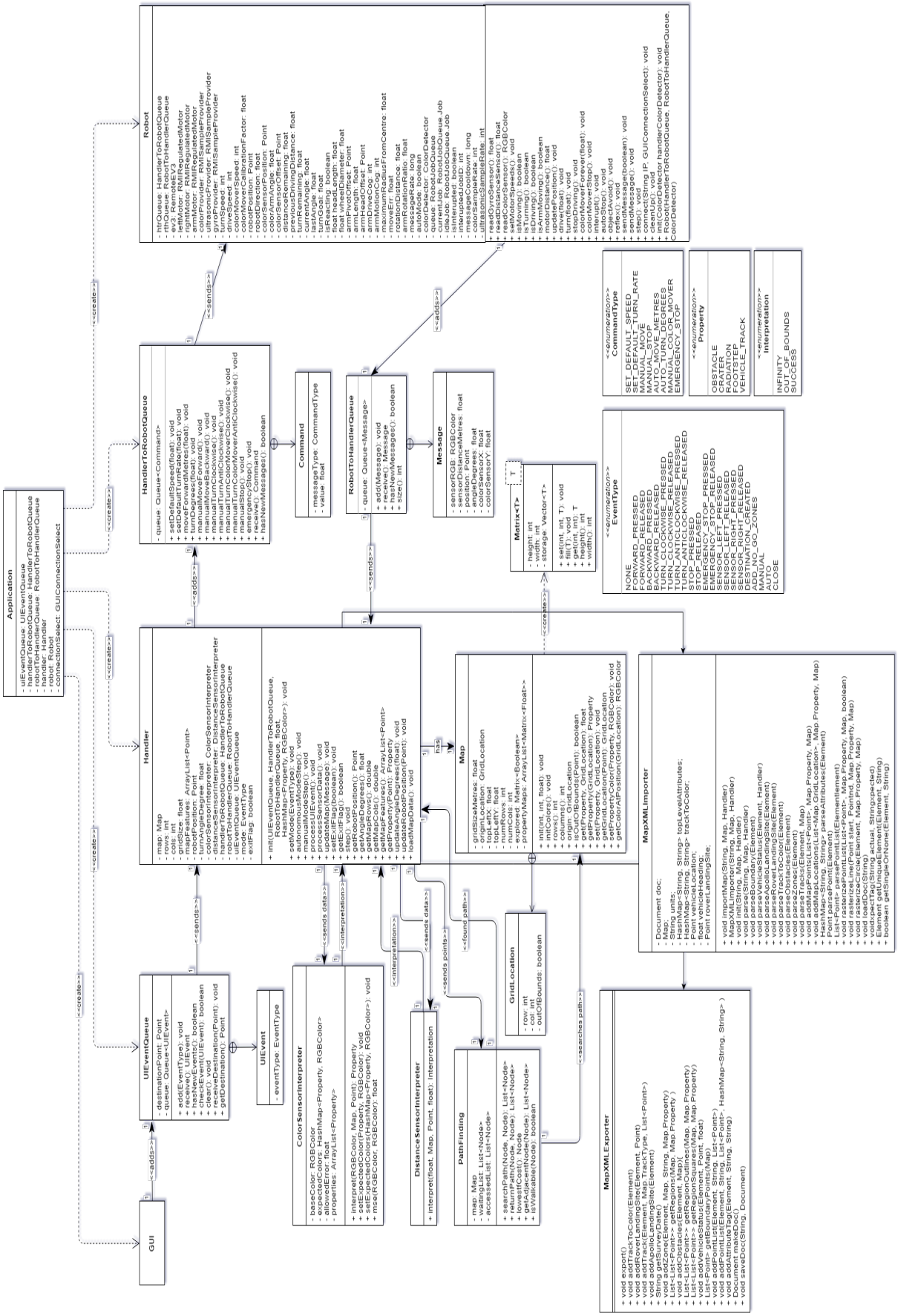


Figure 18: Class Diagram