Software Design Document (SDD)

Software Engineering and Project

University of Adelaide

Prospector Sea Floor Mapping System (PG04)

Navdeep Singh (1660360)
Liang Yuan (1679380)
Zeqi Fu (1680895)
Tao Zhang (1680974)
Lili Wu (1683229)
Yi Lin (1682781)
Yann Frizenschaf (1162562)

Semester 2, 2016

Revision History			
Date	Version	Reason for Change	Author
16th Sep 2016	0.1	Created a initial template	Yann Frizenschaf
28th Sep 2016	0.2	added 3.3.3 to 3.3.8	Tao Zhang
29th Sep 2016	0.3	added system overview	Liang Yuan
2nd Oct 2016	0.4	added user interface design	Liang Yuan
2nd Oct 2016	0.5	added UML figure and content of 5.1	Yi Lin
2nd Oct 2016	0.6	modified Movement Controller and Sensor Controller	Tao Zhang
3rd Oct 2016	0.7	Added state diagram and Data Controller	Yann Frizenschaf
5th Oct 2016	0.8	Added hardware section	Yann Frizenschaf
6th Oct 2016	0.9	added the references	Yi Lin
8th Oct 2016	0.10	Added navigation section	Yann Frizenschaf
8th Oct 2016	0.11	add 3.5 section, add GUI design and modified 3.3.4	Zeqi Fu
9th Oct 2016	1.0	Draft release	Yann Frizenschaf
29th Oct 2016	2.0	Updates for final release	Yann Frizenschaf

Contents

1 Introduction **1**
1.1 Purpose . . . 1
1.2 Scope . . . 1
1.3 Overview . . . 1
1.4 Constraints . . . 2

2 System Overview **2**
2.1 Hardware . . . 3
2.2 Operational Area . . . 3
2.3 Navigation . . . 4

3 System Architecture and Components Design **5**
3.1 Architectural Description . . . 5
3.2 Component Decomposition Description . . . 6
3.3 Detailed Components Design Description . . . 7
3.3.1 C001 - Main Window Controller . . . 8
3.3.2 C002 - Connection Controller . . . 8
3.3.3 C003 - Movement Controller . . . 10
3.3.4 C003.1 - Navigation Controller . . . 12
3.3.5 C004 - Sensor Controller . . . 13
3.3.6 C005 - Mapping Controller . . . 16
3.3.7 C006 - Data Controller . . . 17
3.4 Architectural Alternatives . . . 18
3.5 Design Rationale . . . 18

4 Data Design **19**
4.1 Data Model Description . . . 19
4.2 Data Structures . . . 19
4.2.1 MapDataModel . . . 19
4.2.2 GridSquare . . . 20
4.2.3 GridSquareEnum . . . 21

5 Design Details **21**
5.1 Class Diagrams . . . 21
5.1.1 prospector.sensor . . . 21
5.1.2 prospector.mapping . . . 21
5.1.3 prospector.opsgui . . . 21
5.1.4 prospector.movement . . . 22
5.1.5 prospector.data . . . 22

5.1.6 prospector.connection 22
5.2 State diagrams 28

6 Human Interface Design **28**
6.1 Overview of the User Interface 28
6.2 Detailed Design of the User Interface 29
6.2.1 UI model 29

7 Glossary **30**

8 References **31**

1 Introduction

1.1 Purpose

The purpose of this document is to describe the detailed design and implementation of the software portion of the Prospector Seafloor Mapping System (SFM), developed for SeaFaults. As a whole, this document represents a detailed view of the software architecture, individual components and external/internal interfaces developed to deliver the functionality specified in the associated Software Requirements Specification (SRS) [8].

1.2 Scope

This document defines the design of the software component of the SFM system only. While hardware considerations are touched on in the form of interface requirements, the hardware to be used has been defined and consideration of this (beyond its effect on the requirements of the software system) are beyond the scope of both this document and the project as a whole.

The software for which the design is defined herein is intended to enable effective mapping of a defined area of sea floor in order to determine its suitability for use by a SeaFaults client. It will do this by mapping the sea floor as accurately and efficiently as possible, and providing outputs pertaining to the mapping operation once the operation is complete. The system itself makes no determination of the suitability of an area for a particular application, rather, it produces quantitative map data which can be utilised by SeaFaults and/or its clients to assess the mapped area for suitability.

1.3 Overview

The sea floor is considered a significant frontier for development of many industries including, but not limited to, scientific research, mining, transport, and maintenance of communications infrastructure. A key consideration for any activity taking place on or near the sea floor is a sound understanding of the geology and topography of the area under consideration.

The Prospector SFM system enables seafloor exploration through remote, autonomous control of a robotic vehicle which facilitates data acquisition. The primary goals of the software platform are therefore:

- to enable a human operator to send appropriate commands to the robot via a graphical user interface (GUI) in order to initiate autonomous exploration of the survey area (and intervene when necessary); and
- extract the gathered data via the same GUI into a suitable file format.

These requirements are elucidated in more detail in the SRS.

1.4 Constraints

Prior to the start of the project, the following constraints were placed on the software implementation:

- The software must be written in the Java programming language, and be compatible with the provided hardware/firmware platform: a Lego Mindstorms EV3 robot running the Lego Java Operating System (LeJOS).
- The must contain no more than 10% re-used or external code, not including code within the Java Development Kit (JDK) or LeJOS class package (determined by a standard lines-of-code metric).
- The software must be built using the ANT and MAKE build tools, and be provided with appropriate build scripts.
- The software must meet the requirements specified in the SRS.

2 System Overview

The Prospector SFM system consists of a Java application run on a desktop computer environment, communicating with the hardware platform via Remote Method Invocation interfaces provided as part of the LeJOS Java libraries. The user's primary interaction with the software system is via a GUI, built on the JavaFX platform provided as part of the JDK. This GUI allows the user to establish a connection to the remote hardware platform (hereafter referred to as the robot), change the system's operational mode, send commands directly to the robot (depending on operational mode), monitor the progress of the mapping operation, and import/export map data files.

The system is capable of operating in three modes: fully manual operation, move-to-point mode, and fully autonomous mode. For further details of the requirements of each operation mode, see the SRS [8]. In each of these modes, actuators are

controlled and sensors read via Remote Method Invocation (RMI) on the remote hardware platform. During the survey operation, the user is able to monitor the mapping progress via a near real-time display on the GUI.

Sections 2.1 to 2.3 give a high-level overview of the system's operation. Implementation details are discussed in Section 3.3.

2.1 Hardware

The following sensors and actuators are provided as part of the robot configuration:

Large servo motors: These motors provide the primary means of locomotion for the robot. Two are used for the Prospector robot. They provide tachometer feedback to one degree of accuracy [3].

Medium servo motor: This motor can be used to change sensor orientation. One of these is used in the prospector system in order to allow scanning using the ultrasonic sensor. It provides tachometer feedback to one degree of accuracy [4].

Ultrasonic sensor: This sensor allows detection of distance between the sensor and a reflective object. It has a specified dynamic range of 1-250 cm and accuracy of 1 cm [5]. Practical testing has revealed the usable dynamic range to be about 5-30 cm due to the significant spread angle of the ultrasonic beam.

Colour sensor: This sensor allows detection of colour. It has a sample rate of 1 kHz [6].

Gyro sensor: This sensor detects the orientation of the robot relative to its start position. It has a specified accuracy of +/-3 degrees and a sample rate of 1 kHz [7].

Preliminary testing has shown significant drift in the orientation accuracy of the robot after multiple turns, even when correction via the gyroscopic sensor is implemented. The implications of this drift are discussed in Section 2.3.

2.2 Operational Area

In order to map the survey area effectively, the area will be modeled as a grid, using a Cartesian coordinate system as shown in Figure 1. This allows for flexible adjustments in accuracy of the system output as the system is developed and tested; the grid square size can be decreased as accuracy of the mapping system is improved.

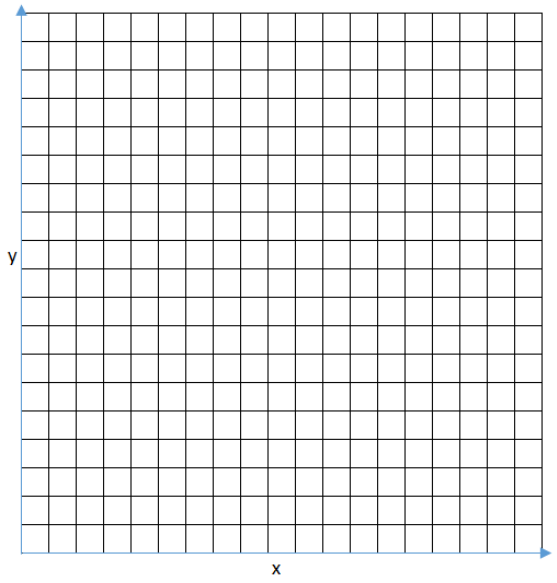

Figure 1: Cartesian grid representation of survey area. Origin (0,0) is in bottom left corner

The robot will move in increments of one grid square, with a brief pause between increments (even in automatic modes). This ensures that sensor readings can be taken in each grid square without ambiguity. In the case where a significant feature (i.e. a faultline) is narrow and falls between robot halt points, the feature will be mapped to the nearest applicable grid square.

2.3 Navigation

In move-to-point mode, the path between the start and end grid squares is calculated using an A* search method with a Manhattan heuristic (performed by the Navigation controller–see Section 3.3.4) [9]. The sequence of moves is then passed to the Movement Controller (see Section 3.3.3) which is responsible for making the appropriate RMI calls to the robot's actuators and correcting its course based on feedback from the Sensor Controller (see Section 3.3.5).

Fully autonomous mode is merely an extension of move-to-point mode in that a series of waypoints are defined based on the expected size of the survey area and any known obstacles. The navigation tree is rebuilt after each navigation step, or when an unexpected obstacle is encountered. The survey area is traversed in a spiral pattern (see Figure 2) in an attempt to minimise the number of turns required to traverse the entire area due to the angular accuracy problem mentioned in Section

2.1.

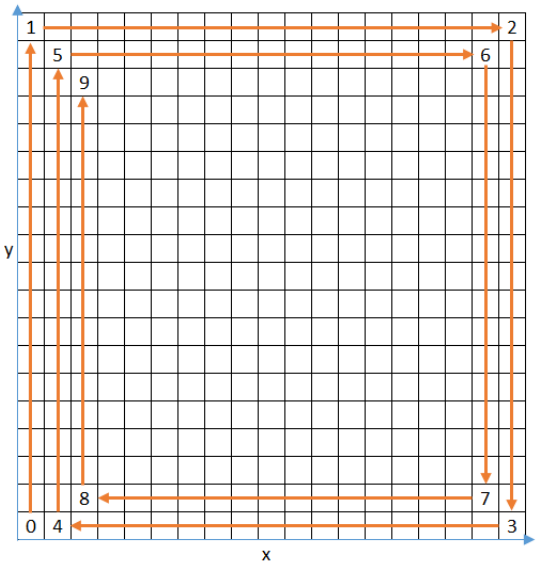

Figure 2: First nine points of autonomous navigation pattern. Point zero is the assumed start point

3 System Architecture and Components Design

3.1 Architectural Description

The Prospector SFM system consists of six major sections:

- Graphical user interface (Operations GUI)
- Data (file) input/output
- Mapping system
- Movement system
- Sensor system
- Remote connection system

Each of these systems is encapsulated in a Java package under the top-level Prospector package, as represented in Figure 3, in order to enforce an appropriate level of

separation of concerns. Communication between top-level classes in different packages is via the Observer pattern [1]; each package contains interfaces which can be implemented by other classes in order to receive asynchronous callbacks based on important events. The interfaces provided in each package are also shown in Figure 3. The details of these interfaces and their implementing classes is discussed in Section 3.3.

A key architectural decision was to execute all control logic off-board (i.e. on the operator workstation rather than on the EV3 hardware platform) and directly invoke sensor readings and actuator motion via the LeJOS-provided RMI interfaces. Alternatives to this architectural choice and the rationale for making this decision are discussed in Sections 3.4 and 3.5, respectively.

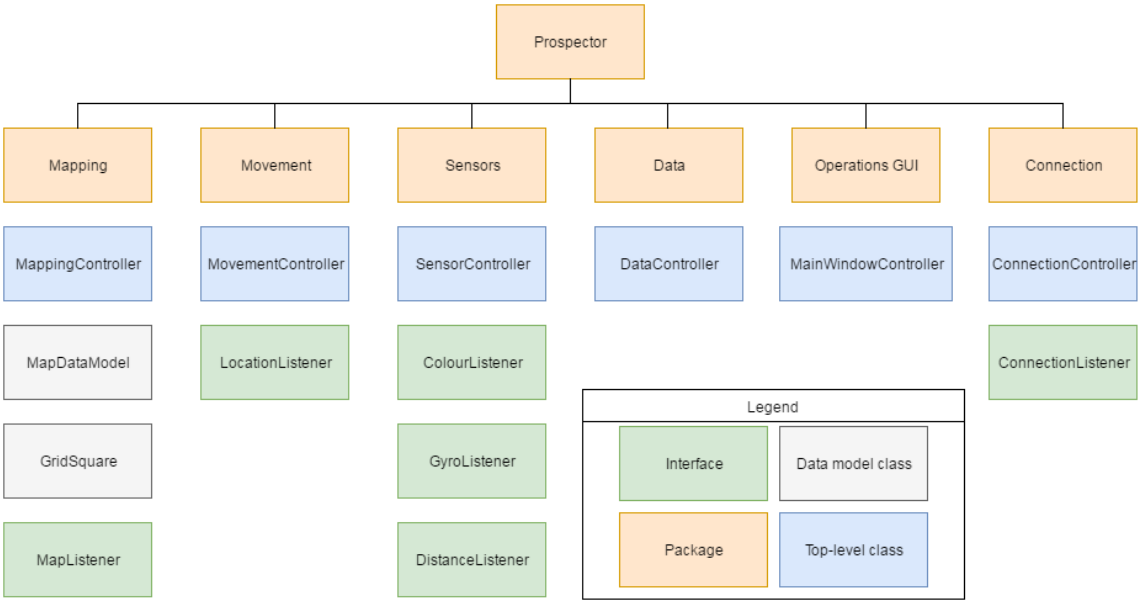

Figure 3: Prospector SFM software system architecture overview

3.2 Component Decomposition Description

Each of the major components listen in Section 3.1 communicates with one or more other major components by implementing a listener interface from the appropriate package. Each major component maintains an internal list of listeners which have registered to receive callbacks for a particular event type, and components register to receive events from each other via public registration and methods. For example, the Movement Controller instance is registered as a listener to the Connection Controller

instance and subsequently receives a callback from that instance when a connection with the remote EV3 robot has been established (or when a connection attempt has failed). For simplicity and to enforce separation of concerns, each listener interface contains only one abstract method.

In general, components register as listeners with each other in the Prospector class, which is the main Java class for the system, prior to display of the GUI (i.e. the JavaFX *launch* method). As only one of each major class is required for operation of the system, each of these classes exists as a single static instance declared in the Prospector class and instantiated in either the static *main* method or the *launch* method.

Because the GUI framework of choice was JavaFX (see Section 6 for rationale), the Prospector main class extends the JavaFX Application class, which in turn invokes the overridden *launch* method in which GUI components are instantiated and displayed.

Figure 4 shows the channels of communication between major components via the Observer pattern. For details of the interfaces themselves, see Section 3.3. UML diagrams can be found in Section 5.1.

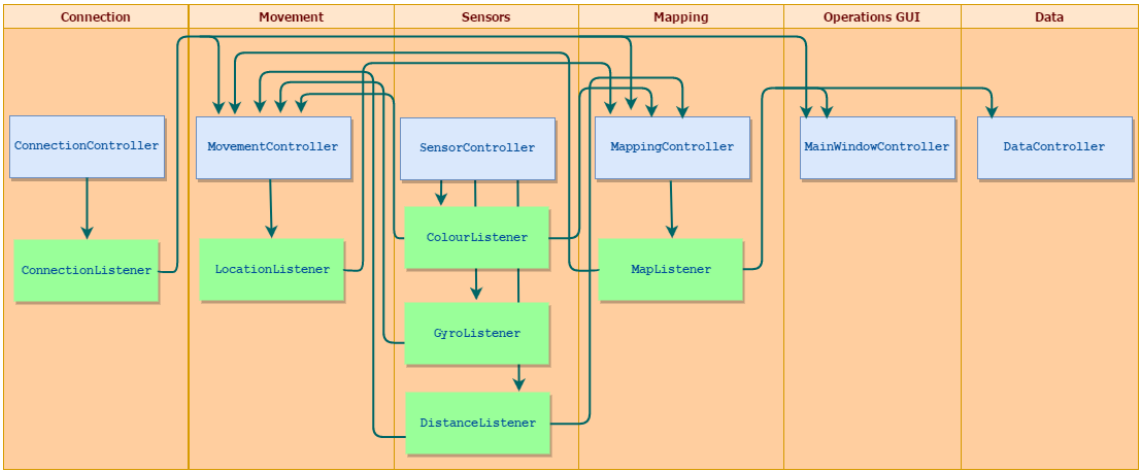

Figure 4: Communication between components via listener interfaces

3.3 Detailed Components Design Description

The functionality of each subsystem component is discussed below, along with an overview of the public methods for each top-level controller class, and the interfaces

implemented and provided by each. Also, the SRS requirements covered by each component are referenced.

3.3.1 C001 - Main Window Controller

Purpose: URC001-009, URE001-004

Function: The Main Window Controller is the controller part of the JavaFX GUI implementation (see Section 6), and converts user input (button presses, text field inputs) to a form which can be translated to commands for other system components.

Subordinates: None

Dependencies: C002, C003, C004, C005

Interfaces: The Main Window controller provides no interfaces to other components, but holds a direct reference to both the Movement and Connection controllers in order to invoke robot movement commands and connection attempts, respectively. Further details of the GUI functionality is provided in Section 6.

Data: None.

3.3.2 C002 - Connection Controller

Purpose: FRC005, FRC006, FRC007

Function: The Connection Controller does is responsible for establishing a connection to the remote robot, and creating references to each of the required hardware components so that they can be controlled and monitored via RMI. The Connection Controller's public methods are listed in Table 1.

Subordinates: None

Dependencies: None

Interfaces: The Connection Controller provides the ConnectionListener interface in order to allow other components to receive notification when a connection attempt has succeeded or failed. The methods provided by this interface are shown in Table 2.

Method name	Return type	Parameters	Description
setAddress	void	String	Sets the IP address for the remote robot connection
connect	void	void	Initiates a connection attempt to the remote robot
getConnected	boolean	void	Returns true if a connection has been established to the robot, false otherwise
closeAll	void	void	Shuts down all remote hardware ports
addConnectionListener	void	ConnectionListener	Adds a listener for connection events
getMotorA	RMIRegulatedMotor	void	Returns a reference to the robot's "A" motor, which can be used for RMI calls
getMotorB	RMIRegulatedMotor	void	Returns a reference to the robot's "B" motor, which can be used for RMI calls
getMotorD	RMIRegulatedMotor	void	Returns a reference to the robot's "D" motor, which can be used for RMI calls
getUltSample	RMISampleProvider	void	Returns a reference to the robot's ultrasonic sensor, which can be used for RMI calls
getGyroSample	RMISampleProvider	void	Returns a reference to the robot's gyroscopic sensor, which can be used for RMI calls
getColourSample	RMISampleProvider	void	Returns a reference to the robot's colour sensor, which can be used for RMI calls

Table 1: ConnectionController public methods

Method name	Return type	Parameters	Description
connectionStateChanged	void	boolean	Indicates success or failure of a connection attempt

Table 2: ConnectionListener public methods

Data: The connection controller stores a string representing the IP address to use in order to connect to the robot, and references to the individual hardware components' RMI interfaces.

3.3.3 C003 - Movement Controller

Purpose & Function: The Movement Controller is responsible for controlling the movement of the robot via remote method invocation based on user input. The implementation of the Movement Controller is split into three modes: Manual Control Mode, Move-to-Point Mode and Fully Autonomous Mode. The details of these are broken down below, and the Movement Controller's public methods are listed in Table 3.

(1) Manual Mode

Purpose: FRC005

Function: The Manual Control Mode allows the user to manually control the movement of the robot from GUI.

(2) Fully Autonomous Mode

Purpose: FRC006, FRC007

Function: The Fully Autonomous Mode allows robot to find a path from current location to destination provided by operator from GUI. It also allows robot to survey an area automatically.

(3) Move-to-Point Mode

Purpose: FRC001, FRC002, FRC003, FRC004

Function: In Move-to-Point mode, the robot will move to the specified point, avoiding obstacles and no-go-zones.

Method name	Return type	Parameters	Description
setMoveMode	void	MoveMode	Sets the move mode of Movement Controller
startSurvey	void	Integer[]	Starts survey based on current mode. Survey a path from current position to a destination in Move-to-Point mode. Survey whole area in Fully Autonomous mode.
connect	void	Navigation	Connects MovementController class to Navigation class
connect	void	MappingController	Connects MovementController class to MappingController class
forwardCommand	void	void	Processes a manual forward command
backwardCommand	void	void	Processes a manual backward command
leftCommand	void	void	Processes a manual left turn command
rightCommand	void	void	Processes a manual right turn command
stop	void	void	Stops the robot from moving
addLocationListener	void	LocationListener	Adds a LocationListenter to the internally maintained list
setLocation	void	double, double, float	Sets the current location and broadcast location to every listener

continued on next page

continued from previous page			
Method name	**Return type**	**Parameters**	**Description**
startSurvey	void	integer[ ]	Initiates survey or movement to specified x, y destination, depending on mode

Table 3: MovementController public methods

Subordinates: C003.1

Dependencies: C002, C004, C005

Interfaces: The Movement Controller provides the LocationListener interface in order to allow other components to receive notification when the robot move to a new position. The methods provided by this interface are shown in Table4.

Method name	**Return type**	**Parameters**	**Description**
locationReceived	void	double, double, float, float	Indicates the location and direction of the robot

Table 4: LocationListener public methods

Data: The movement controller stores a integer representing the direction of robot(North, East, South, West). The movement controller stores an enum which named MoveMode representing the mode of movement (Manual, Move-to-Point or Fully Autonomous).

3.3.4 C003.1 - Navigation Controller

Purpose: FRC004, FRC006, FRC007

Function: As a subordinate of the Movement Controller, the Navigation controller provides the path-finding functionality required to enable autonomous robot movement. Given a destination in (x,y) grid coordinates, it performs an A* search algorithm in order to find a path from its current location to the destination. When invoked from the Movement Controller, it returns a list of movements required to reach the destination. This controller's public methods are listed in Table 5.

Subordinates: None

Dependencies: C003, C005

Method name	Return type	Parameters	Description
run	List<Integer[ ]>	Integer[ ]	Given a destination in (x, y) coordinates, returns a list of waypoints in (x, y) coordinates based on A* algorithm
connect	void	MovementController	Sets a reference to the MovementController class

Table 5: NavigationController public methods

Interfaces: None.

Data: The navigation controller stores and updates the tree of movements required to reach the current target destination.

3.3.5 C004 - Sensor Controller

Purpose & function: The Sensor Controller is responsible for controlling the sensors of the robot. The implementation of the Sensor Controller is made up of Color Sensor Controller, Gyro Sensor Controller and Ultrasonic Sensor Controller. Purpose and function for each part is as follow. The Sensor Controller's public methods are listed in Table 6

Method name	Return type	Parameters	Description
addColourListener	void	ColourListener	Adds a ColourListener to the list, it can detect any color changing from color sensor
addDistanceListener	void	DistanceListener	Adds a DistanceListener to the list, it can receive distance change from movement Class
addAngleListener	void	GyroListener	Adds a AngleListener to the list, it can detect any angle changing form angle sensor

continued on next page

continued from previous page

Method name	**Return type**	**Parameters**	**Description**
broadcastColour ToListeners	void	void	Informs all color listeners of the current color reading, and it can be called any time
broadcastDistance ToListeners	void	void	Informs all distance listeners of the current distance reading, and it can be called any time
broadcastAngle ToListeners	void	void	Informs all angel listeners of the current angel reading, and it can be called any time
start	void	void	Starts thread of sensor to detect distance, gyro and colour
getDistance	float	void	Gets data of distance from ultrasonic sensor
startDistanceThread	void	void	Starts a thread to keep reading distance from ultrasonic sensor and broadcast the data to every listeners
startGyroThread	void	void	Starts a thread to keep reading angle from gyro sensor and broadcast the data to every listeners
startColourThread	void	void	Starts a thread to keep reading colourid from colour sensor and broadcast the data to every listeners

continued on next page

continued from previous page			
Method name	**Return type**	**Parameters**	**Description**
shutdown	void	void	Closes threads of distance, gryo and colour

Table 6: SensorController public methods

Subordinates: None

Dependencies: None

Interfaces: The Sensor Controller provides the ColourListener, GyroListener and DistanceListener interfaces. The ColourListener allows other components to receive the colorID detected by the colour sensor. The GyroListener allows other components to receive the rotation angle detected by the gyro sensor. The DistanceListener allows other components to receive the distance between an obstacle and robot detected by the Ultrasonic sensor. The methods provided by these interfaces are shown in Table7.

Data: The sensor controller stores a float representing the color, a float representing the distance and a float representing the rotation angle degree.

(1) Color Sensor Controller

Purpose: FRC002, FRC004, FRC0015, FRC0016

Function: The color sensor will detects the colorID of current position of robot and broadcasts it to other components.

(2) Gyro Sensor Controller

Method name	**Return type**	**Parameters**	**Description**
colourDetected	void	float	Indicates the color detected by the color sensor
distanceReceived	void	float	Indicates the distance detected by the ultrasonic sensor
angleReceived	void	float	Indicates the angle detected by the gyro sensor

Table 7: SensorListener public methods

Purpose: FRC001

Function: The Gyro sensor will detects the rotation angle of robot and broadcasts it to other components.

(3) Ultrasonic Sensor Controller

Purpose: FRC003

Function: The Ultrasonic sensor will detects the distance around current position of a robot and broadcasts it to other components.

3.3.6 C005 - Mapping Controller

Purpose: URC005, FRC008, FRC009

Function: The Mapping Controller maintains the internal map data model representing the surveyed area, and determines the position of mapped environmental features by synthesising the input from the sensors (via the Sensor Controller) with the robot's current position (via the Movement Controller). This controller's public methods are listed in Table 8.

Subordinates: None

Dependencies: C003, C004

Interfaces: The Mapping Controller implements the LocationListener, ColourListener and DistanceListener interfaces. The Mapping Controller provides the MapListener interface in order to allow other components to receive notification when the internal map data model has been updated. The methods provided by this interface are shown in Table 9.

Data: The Mapping Controller stores the current data model of the survey area in a MapDataModel object (see section 4).

Method name	Return type	Parameters	Description
addMapListener	void	MapListener	Registers a listener for map data model updates
setMapMode	void	boolean	Turns the mapping functionality on or off.
getDataModel	MapDataModel	void	Returns the current version of the map data model representing the survey area
clearMap	void	void	Clears the map data model; marks all areas as unexplored
shutdown	void	void	Stops the scheduled dispatch of data model updates to listeners

Table 8: MappingController public methods

Method name	Return type	Parameters	Description
mapDataReceived	void	MapDataModel	Passes the current version of the map data model

Table 9: MapListener public methods

3.3.7 C006 - Data Controller

Purpose: FRC011, FRC012, FRC013

Function: The Data Controller handles the import and export of map data files, both in response to user demand and on a periodic basis for data backup and recovery. This controller's public methods are listed in Table 10.

Method name	Return type	Parameters	Description
writeFileOut	void	MapDataModel, File	Writes the map data model to an XML file
readFileIn	MapDataModel	URL	Reads a file specified by the URL and returns a MapDataModel object

Table 10: DataController public methods

Subordinates: None

Dependencies: C005

Interfaces: The Data Controller implements the MapListener interface in order to receive map data model updates for periodic backup.

Data: None.

3.4 Architectural Alternatives

A major architectural decision made early in the design process was to keep all logic and control modules off-board, that is, executed on the operator's workstation. An alternative would have been to compile some or all of the control logic into a LeJOS-executable JAR file to be run on the robot hardware itself, leaving only GUI and file input/output code to be run on the operator's workstation. Communication between the workstation and the robot would have been via a custom-developed communication protocol built on Java sockets. This approach would have had the following advantages:

- Ability to continue to execute on-board logic functions such as survey area exploration and updates to internal data model even in the event of transient loss of communication to operator workstation
- Ability to execute emergency evasive action/stop even in the event of loss of communication to operator workstation

Disadvantages of such an approach include:

- Requirement to design and implement a custom communications protocol, representing an additional possible failure mode during operation
- Requirement to design and implement two separate but related Java projects: one for the LeJOS platform (on-board) and one for the operator workstation (off-board)

3.5 Design Rationale

The decision to implement each major functional area of the system as a separate major class in its own package is consistent with the principle of separation of concerns, the advantages of which have been well-established in the literature [2]. The use of the observer pattern with asynchronous callbacks and (where appropriate) scheduled dispatch of data to registered listeners is also consistent with best practices in Java programming [1]. This approach allows for unit testing of each component, as well as progressive integration testing (provided interconnected components can be mocked where necessary for testing).

The architectural alternative discussed in Section 3.4 was discarded due to its added complexity. As well as this, the LeJOS platform already provides a low-level RMI mechanism which can be used for off-board control of motors, making the development of a custom communications protocol an unnecessary additional component.

The major concession made in order to accommodate this design is the step-wise (rather than continuous) movement of the robot. This ensures that in the event of a loss of communication between the robot and workstation, the robot will not move further than one grid square before coming to a halt.

4 Data Design

4.1 Data Model Description

The Prospector SFM software system contains one major data structure used for passing state information between the separate system components. This the MapDataModel class, which includes the robot position model.

4.2 Data Structures

4.2.1 MapDataModel

The MapDataModel object represents the current state of the mapped survey area. It fundamentally consists a grid of squares of predefined size, with each grid square represented by a GridSquare data object. The map data model is initialize once when it receives its dimensions (width, height) via its constructor from configured parameters, and is subsequently maintained by the Mapping Controller. The data members of the MapDataModel class are listed below:

GridSquare[ ][ ] mapGrid: A two-dimensional matrix of GridSquare objects of configurable size representing the survey area.

Integer[ ] robotPosition: An integer array of length 2 representing the robot's current cartesian coordinates at the time the MapDataModel object was updated.

The public methods of the MapDataModel object are listed in Table 11.

Method name	Return type	Parameters	Description
getWidth	Integer	void	Returns the width of the map data model grid
getHeight	Integer	void	Returns the height of the map data model grid
validPoint	boolean	Integer[ ]	Returns true if the provided array represents a point within the defined grid
getGridSquare	GridSquare	Integer[ ]	Returns the grid square at the specified point
setGridSquare	void	Integer[ ], GridSquare	Sets the value of the grid square at the specified point
clear	void	void	Sets the entire map grid to UNKNOWN
setRobotPosition	void	Integer[ ]	Sets the current grid position of the robot
getRobotPosition	Integer[ ]	void	Returns the current grid position of the robot

Table 11: MapDataModel public methods

4.2.2 GridSquare

The GridSquare class represents a single square of the map data model grid. It consists of type designation in the form of a GridSquareEnum member and an Integer member representing a property of the GridSquare instance. This property is used in the case where a GridSquare object represents a fault line, and is used to store the color value of that fault line. The data members of the GridSquare class are listed below:

GridSquareEnum value: The type of grid square represented by this object.

Integer property: A numerical representation of a relevant property of this grid square.

The only public methods of the GridSquare class are the public accessors of its two private members.

4.2.3 GridSquareEnum

The GridSquareEnum class is an integer-backed enumeration representing the possible contents of a grid square. Its possible values and their meaning are shown in Table 12.

Name	Value	Description
UNKNOWN	0	Unexplored area
BLANK	1	Explored and empty
FAULTLINE	2	Fault line
OBSTACLE	3	Three-dimensional obstacle
BOUNDARY	4	Survey area boundary
NGZ	5	Defined No-Go Zone
EXTRACTION	6	Robot extraction container area
ROBOT	7	Robot position at time of data model update

Table 12: GridSquareEnum values

5 Design Details

5.1 Class Diagrams

The Prospector Sea Floor Mapping System divided into six parts (packages), and the UML diagram of their relationships is shown in Figure 5. In addition, more specific UML diagrams of each part are provided in Figures 6 - 11.

5.1.1 prospector.sensor

The UML diagram of sensor control system shown in Figure 6.

5.1.2 prospector.mapping

The UML diagram of mapping system shown in Figure 7.

5.1.3 prospector.opsgui

The UML diagram of GUI shown in Figure 8.

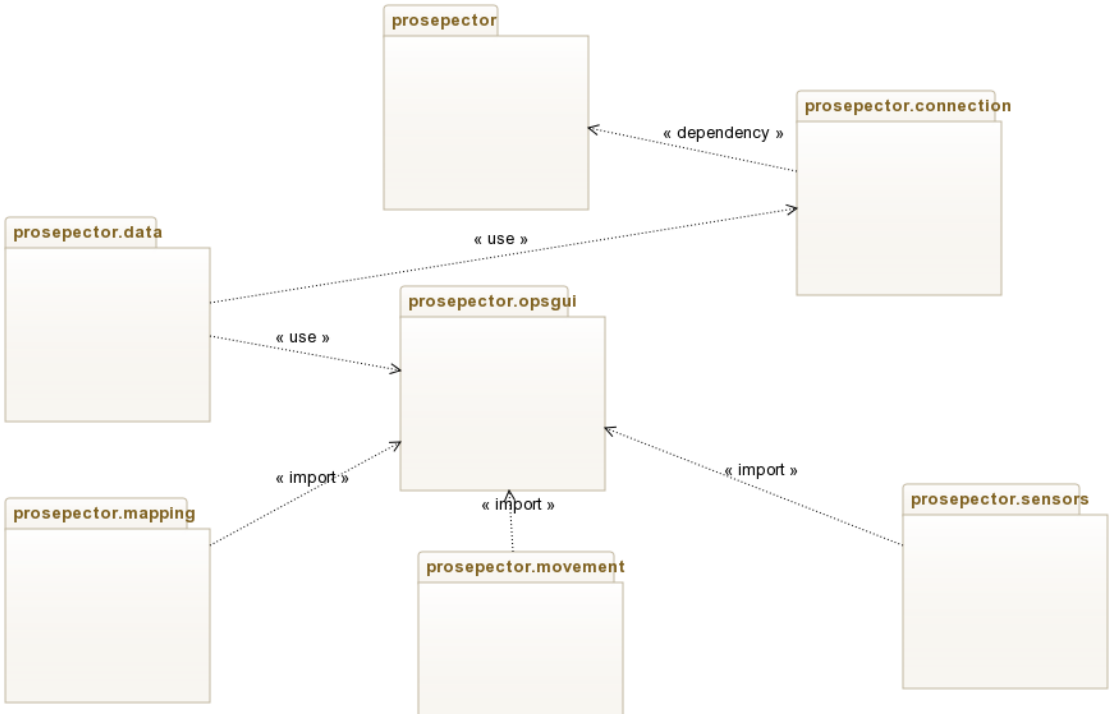

Figure 5: The UML diagram of the Prospector Sea Floor Mapping System

5.1.4 prospector.movement

The UML diagram of movement system shown in Figure 9.

5.1.5 prospector.data

The UML diagram of data collection system shown in Figure 10.

5.1.6 prospector.connection

The UML diagram of connection system shown in Figure 11.

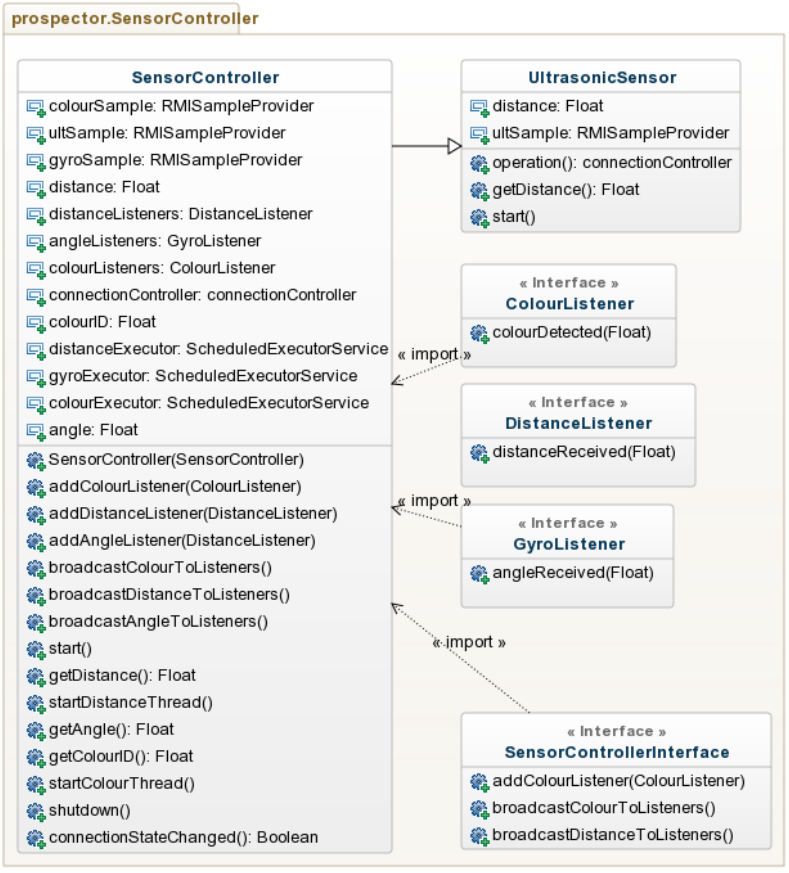

Figure 6: The UML diagram of sensor controller

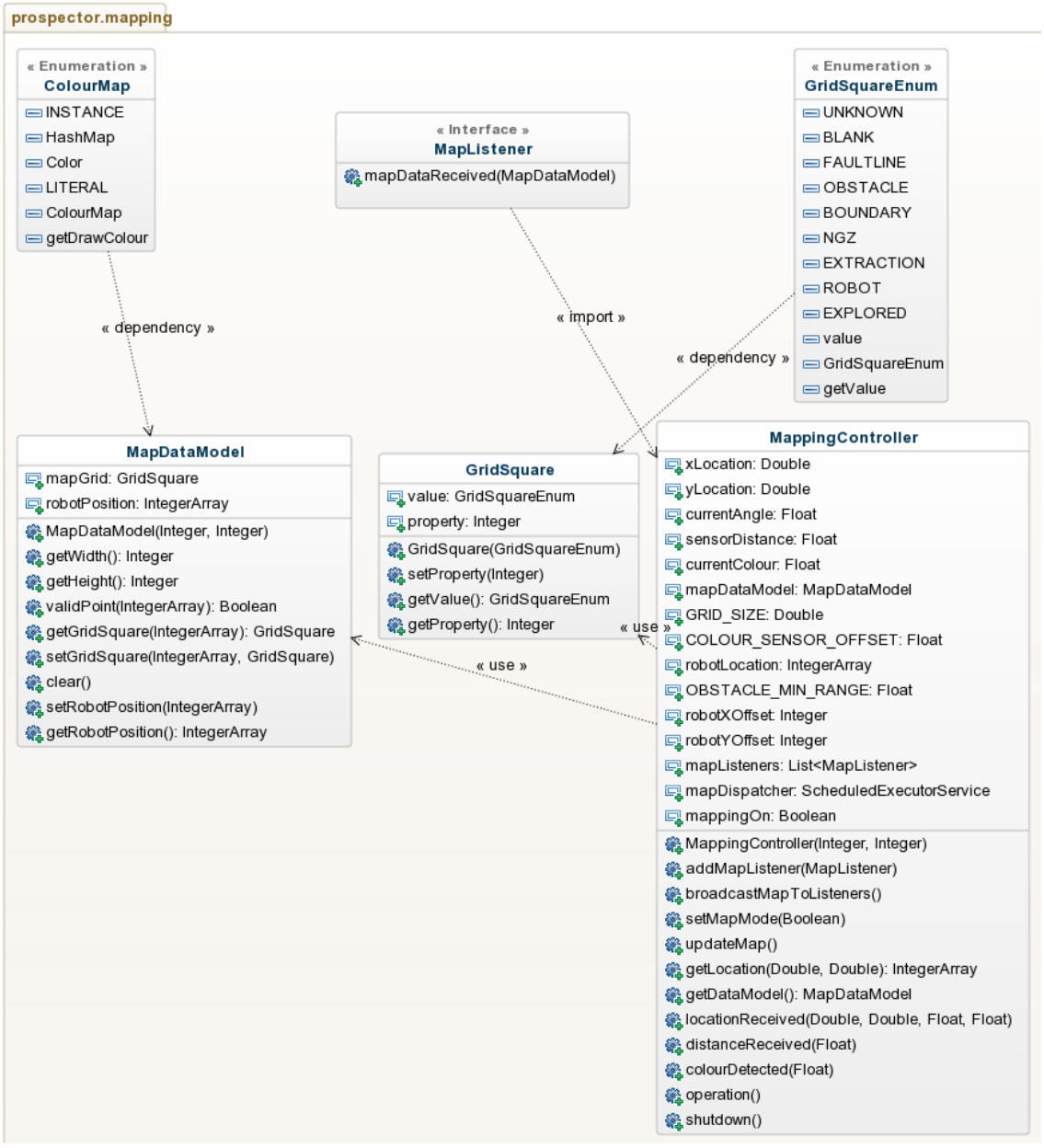

Figure 7: The UML diagram of mapping

Figure 8: The UML diagram of GUI

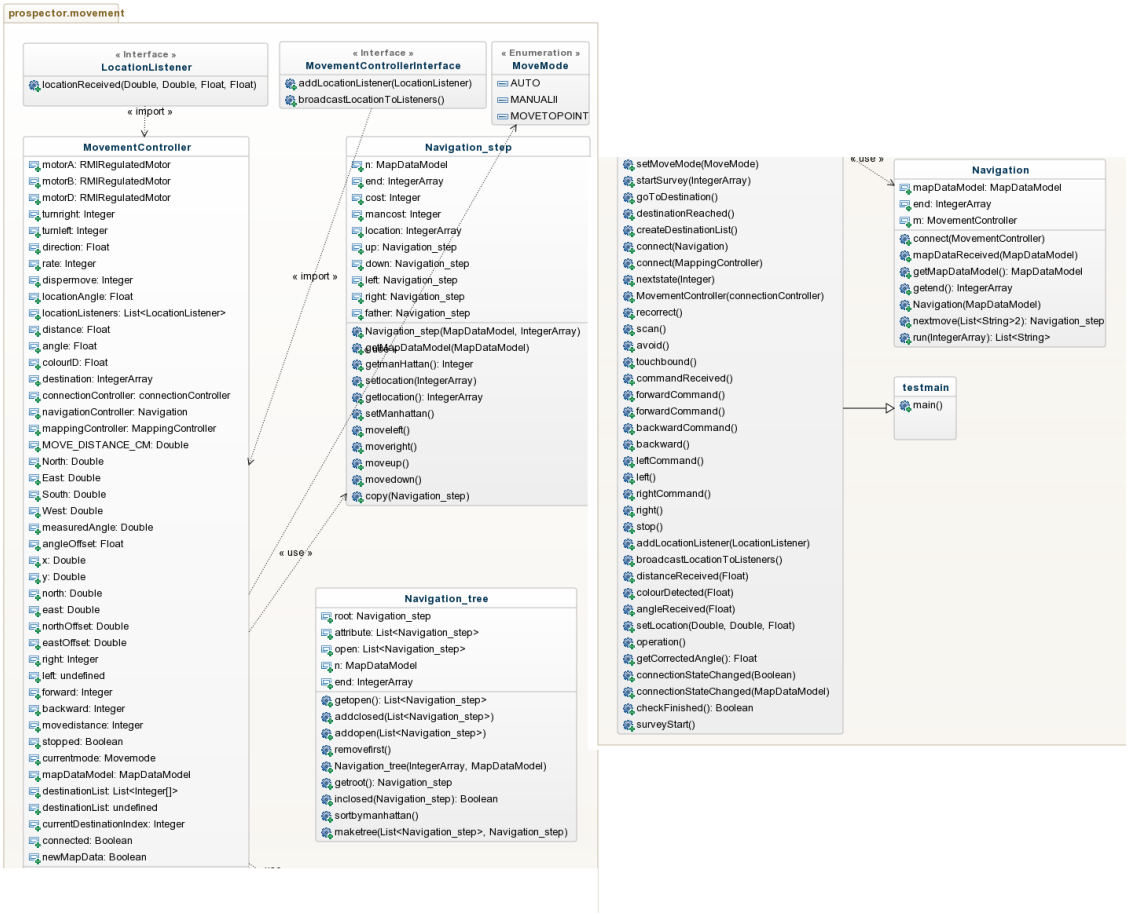

Figure 9: The UML diagram of movement

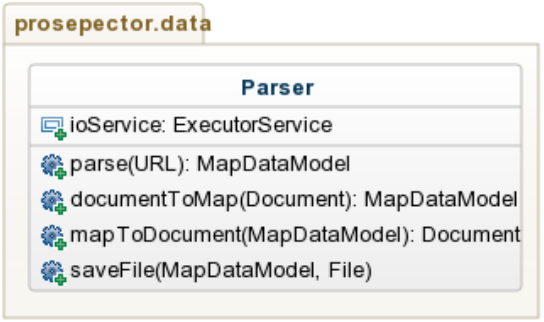

Figure 10: The UML diagram of data collection

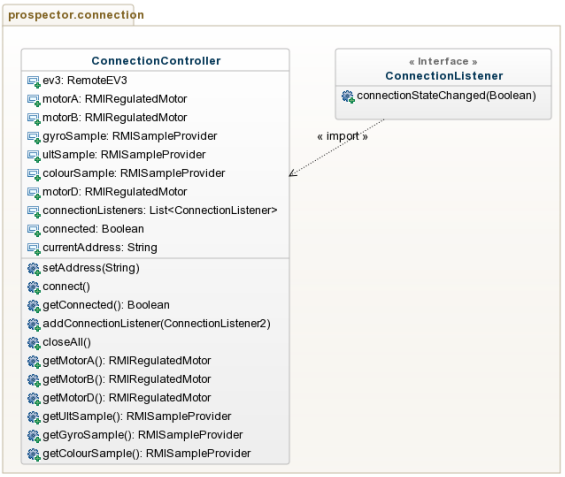

Figure 11: The UML diagram of connection

5.2 State diagrams

The state diagram for the Navigation Controller in fully autonomous mode is shown in Figure 12.

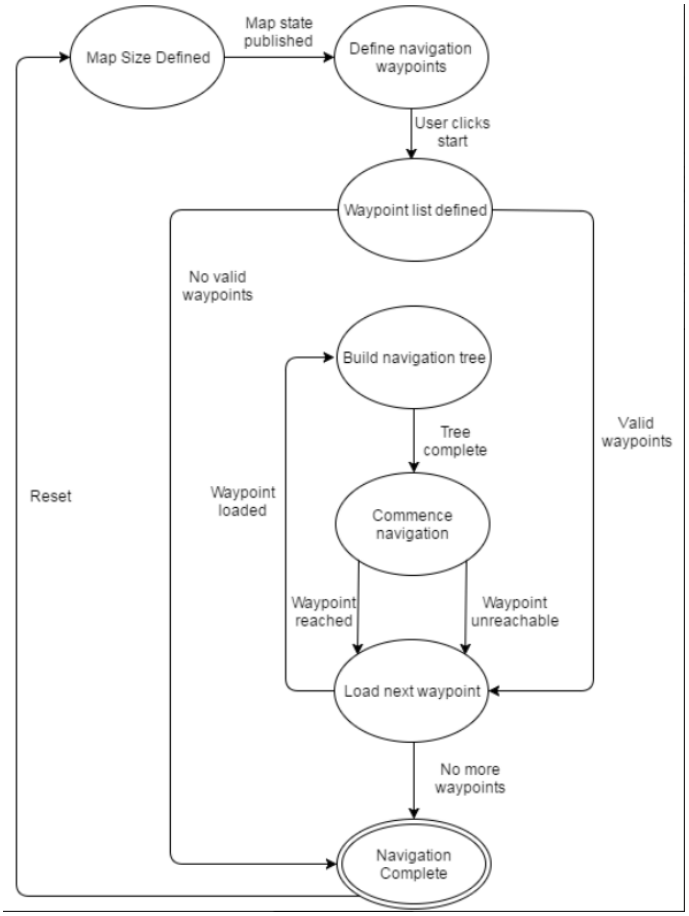

Figure 12: The state diagram of the Navigation controller in autonomous mode

6 Human Interface Design

6.1 Overview of the User Interface

The operation is on the right of the user interface. There are four buttons to control the movement of robot, and a stop button in the center of navigation buttons. The manual or automatic button determine the button is controlled manually or

automatically. The "save map" stores the map which detected by sensor and "load map" button get a map input which means map is already given. The "set location" button sets the current location of robot manually. In addition, there is a text field showing the data of the current status of robot.

6.2 Detailed Design of the User Interface

6.2.1 UI model

The GUI for the Prospector SFM uses a Model-View-Controller (MVC) pattern in order to decouple control logic and internal data modeling from the view presented to the user. To this end, JavaFX was selected over Swing (the main alternative) as the UI framework of choice due to its ability to separate the declarative FXML layout (the view) from the linked controller class, which in turn is separate from the data model classes.

The user interface is shown in Figure 13. The various parts are identified below:

1. Five buttons are provided for users to control robot manually, including forward, backward, turn left, turn right and a stop button. When the user press a button, the robot will move according to the button.
2. A drop-down menu allows robot control mode selection (Manual, Move-to-Point or Fully Autonomous Mode).
3. At any time, the current map can be saved to an XML file, a map can be loaded, or the current map display can be cleared.
4. Users can set the current position and orientation of the robot.
5. A panel displays real-time information from the robot.
6. The user can define the robot's IP address and initiate the remote connection.
7. Updates to the map may be enabled or disabled
8. A map legend is displayed next to the map.
9. A real-time map shows survey progress.

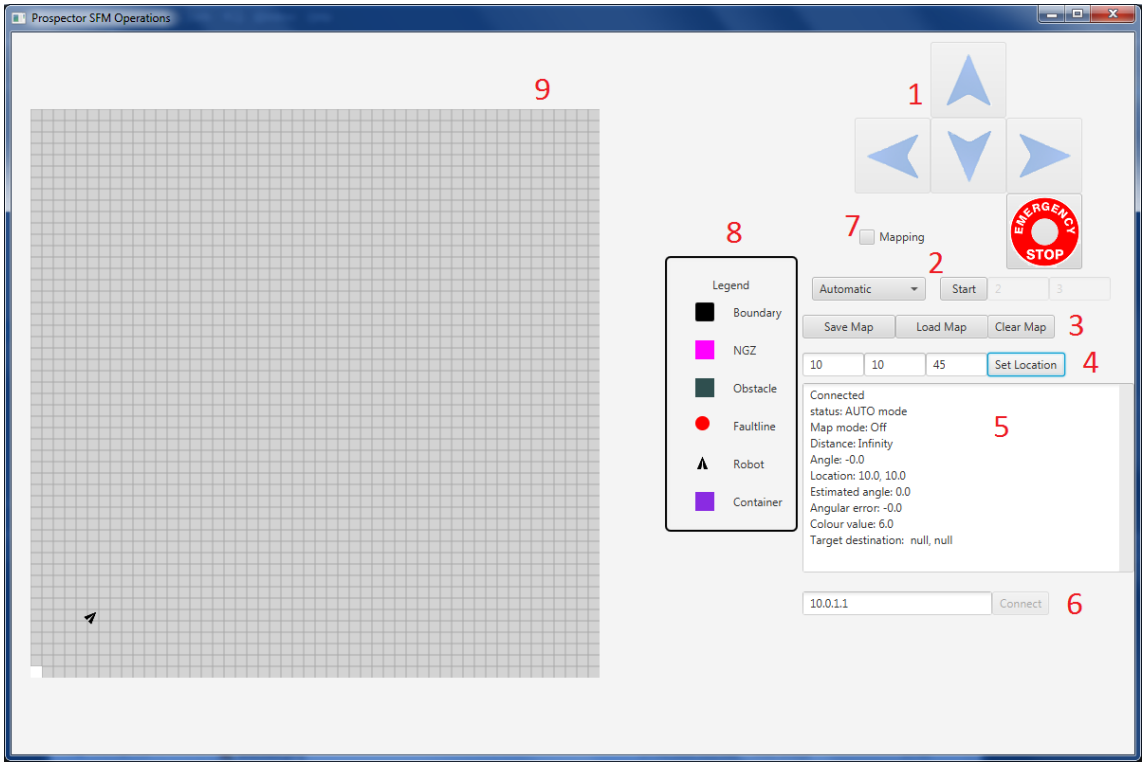

Figure 13: User interface

7 Glossary

GUI Graphical User Interface

LeJOS The Lego Java Operating System

RMI Remote Method Invocation

SFM Sea Floor Mapping

SRS Software Requirements Specification

SDD Software Design Document

MVC Model View Controller

8 References

[1] Buschmann, F, Henney, K and Schmidt, DC 2007. *Pattern-Oriented Software Architecture: On Patterns and Pattern Languages*, Volume 5. Wiley and sons, West Sussex, England.

[2] Hürsch WL and Lopes CV 1995. *Separation of Concerns*. Northeastern University, Boston, USA. Technical report NU-CCS-95-03.

[3] LEGO Education, no date. *EV3 Large Servo Motor*. Available online 1/10/2016 <https://education.lego.com/en-us/products/ev3-large-servo-motor/45502>

[4] LEGO Education, no date. *EV3 Medium Servo Motor*. Available online 1/10/2016 <https://shop.lego.com/en-US/EV3-Medium-Servo-Motor-45503>

[5] LEGO Education, no date. *EV3 Ultrasonic Sensor*. Available online 1/10/2016 <https://shop.lego.com/en-US/EV3-Ultrasonic-Sensor-45504>

[6] LEGO Education, no date. *EV3 Color Sensor*. Available online 1/10/2016 <https://education.lego.com/en-us/products/ev3-color-sensor/45506>

[7] LEGO Education, no date. *EV3 Gyro Sensor*. Available online 1/10/2016 <https://education.lego.com/en-us/products/ev3-gyro-sensor-/45505>

[8] Prospector Team 2016. *Software Requirements Specification: Prospector Seafloor Mapping System*. Version 0.1 (Draft).

[9] Jones, MT 2008. *Artificial Intelligence: A Systems Approach*. Jones and Bartlett Publishers, Sudbury, Massachusetts.