Design Specification

Editor: Erik Bodin

Version 1.0

Status

Reviewed	Erik Bodin	2016-10-06
Approved	Martin Lindfors	2016-10-06

PROJECT IDENTITY

2016/HT, TIGER

Linköping University, Dept. of Electrical Engineering (ISY)

Group members

Name	Responsibility	Phone	Email
Alfred Patriksson	Head of Software	076 210 94 30	alfpa324@student.liu.se
Axel Reizenstein	Head of Hardware	070 279 29 00	axere475@student.liu.se
Erik Bodin	Head of Documentation	076 241 11 32	eribo629@student.liu.se
Gustav Ling	Head of Testing	073 525 72 76	gusli621@student.liu.se
Henric Watz	Head of Information	076 817 57 75	henwa871@student.liu.se
Ingrid Sjöberg	Project Manager	070 652 14 01	ingsj090@student.liu.se
Joakim Mörhed	Head of Design	073 520 84 59	joamo950@student.liu.se

Group e-mail: tsrt10-balrog@googlegroups.com

Customer: Saab Dynamics
Contact at customer: Torbjörn Crona, torbjorn.crona@saabgroup.com
Technical expert at customer: Erik Ekelund, erik.ekelund@saabgroup.com

Examiner: Daniel Axehill, daniel@isy.liu.se
Client: Martin Lindfors, martin.lindfors@liu.se
Supervisor: Per Boström, per.bostrom@liu.se

Contents

Document history

Version	Date	Changes	Changed by	Reviewed
0.1	2016-09-22	First draft	*	EB
0.2	2016-09-27	Second draft	*	*
0.3	2016-09-30	Third draft	*	IS
0.4	2016-10-05	Fourth draft	*	IS
1.0	2016-10-06	First approved version	*	ML

1 Introduction

This is the design specification of the minesweeper project in the project course TSRT10. The minesweeper project is a reoccurring CDIO project that each year further develops and builds off of that which has been done previous years. Going into this year's minesweeper project a *Hardware Abstraction Layer* (HAL), has been put in place. This unifies the interface between MATLAB code and the hardware of the *Balrog* minesweeper robot. This year's project will focus on high level sensor fusion for mapping, positioning, and navigation of the *Balrog* platform and its surroundings, while also providing the platform with a certain level of autonomous control.

An overview of the system can be seen in Figure 1. The *Balrog* platform transmits sensor data, states and decisions and receives control instructions for the tracks along with settings for the various modules. These components will not be modified. The scope of the main program is all software on a high level and includes positioning, navigation and control. The map and data provided by the system will be logged for external use. A Graphical User Interface will also be implemented for visualisation of the mapping and positioning of the platform.

This document will provide an explanation of the hardware of the system together with an explanation of the software system that is to be developed in the project.

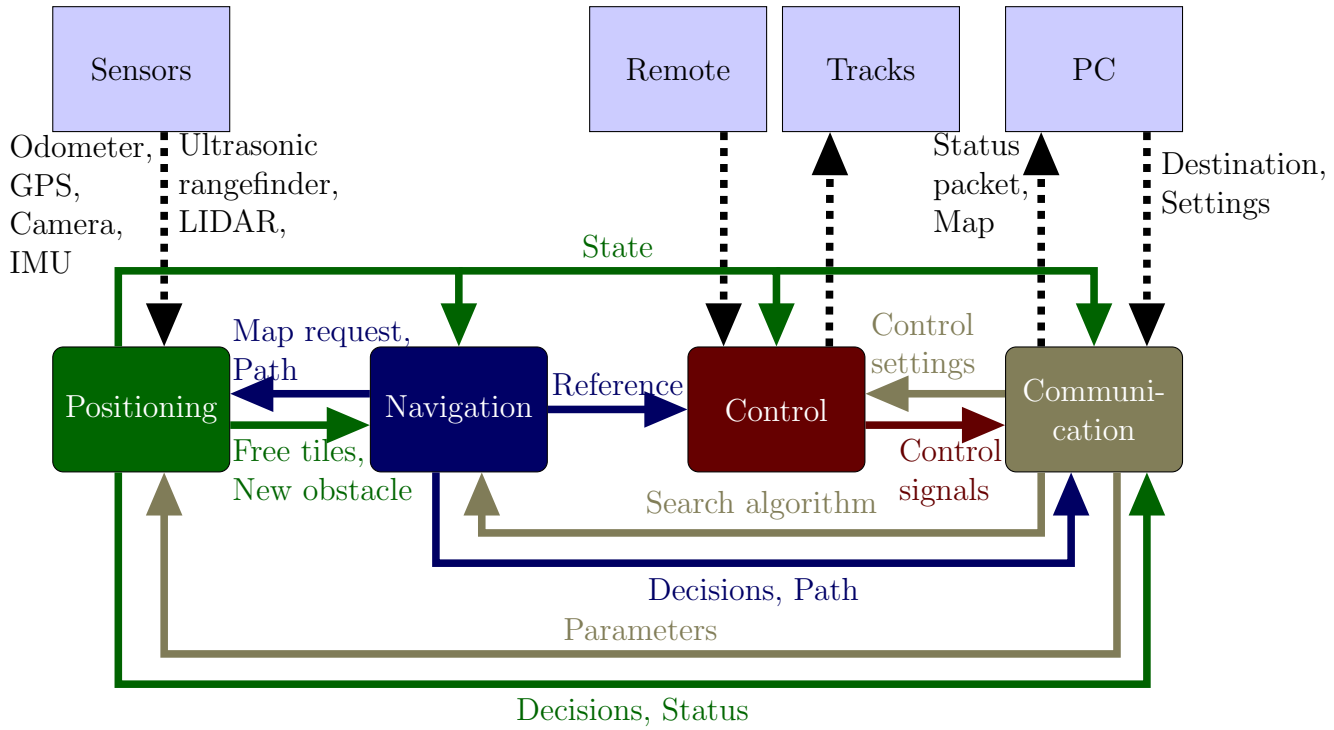

Figure 1: Overview of the structure of the system, with color coded components and data transfers.

1.1 Definitions

- *Balrog* or *Platform* - The mine sweeping vehicle that is used in this project.
- *The project group* - The group of students that will work on this project.
- *Position* - The center of the IMU unit placed on the platform, which is also the center of rotation.
- *Indoor test area* - A horizontal rectangular indoor area with side lengths from 5 to 25 meters. The test area will be static over time.
- *Obstacle* - An object that the platform cannot pass through.
- *Map* - A local made-to-scale representation of the indoor test area. It may include selected obstacles.
- *Software in the loop (SIL)* - A simulation of the complete system running locally on a host computer, often used for quick, iterative testing of implementations.
- *LIDAR* - LIght Detection And Ranging, A laser rangefinder used to measure long distances.
- *IMU* - Inertial Measurement Unit, used to measure angular velocity and acceleration.
- *Uniform-cost search* - A search algorithm that maximizes short term instead of long term profit. 'Profit' might represent time saved, money saved, distance traveled etc.
- *HAL* - A Hardware Abstraction Layer provides a unified high-level interface to access low level hardware functionality. This is so that programmers do not have to consider the details of signal timing, polling, or similar hardware considerations.
- *LQR* - Linear-quadratic regulator.

1.2 Sustainability

Sustainable code is a core requirement of the finished product and must therefore be taken into special consideration.

It can be argued that for sustainable code an object oriented structure is preferred. With an OOP structure we here look to utilise the encapsulation nature of the paradigm, rather than incorporating the OOP paradigm. With this in mind a functional, structured program with classes encapsulating core functionality is suggested.

New C++ code shall follow the google coding standard found at
"https://google.github.io/styleguide/cppguide.html"

MATLAB code shall follow the matlab style guidelines 2.0 provided by matworks found at "http://se.mathworks.com/matlabcentral/fileexchange/46056-matlab-style-guidelines-2-0"

2 General Design

The main purpose of this project is to implement an accurate positioning algorithm. The hardware and the provided software that handles communication with the hardware will not be modified. The algorithms that are implemented will be written in MATLAB and converted to C using code generation.

The positioning module will estimate the position of the platform. The positioning module will also estimate the position of objects detected by the range sensors. This information will be stored in a map which the other modules will use. The positioning module will use the *OpenSLAM* project *gmapping* for mapping and positioning.

The navigation module will use the map provided by the positioning module to determine a route for the controller to use as reference. The navigation module will contain the search algorithms used.

The control module will calculate an appropriate control signal. This signal will allow the robot to navigate the surroundings as commanded by the navigation module.

The GUI module will simply be an interface between the system and a human operator. It will receive information from all the different sub-systems and display it graphically in an understandable way. It will also be possible for an operator to input values into the system. For example it will be possible to enter a destination on the map or settings for the controller.

Figure 1 shows how the different sub-systems are connected and how information is shared between them.

2.1 System state model

The system will be represented by a dynamic model. The state vector of this model can be used as a common interface between certain modules and sub-modules. A motion model and measurement model already exists from previous projects.

3 Hardware

Overview of the hardware components used for the project that the software are to communicate with. The *Balrog* platform is equipped with a wide range of hardware. The various components are described below.

3.1 Processors

There are three single-board computers with various purposes on the platform. A Raspberry Pi will handle the main program and algorithms for positioning and navigation, and two Arduinos handle communication with hardware.

3.1.1 Raspberry Pi

The main computer of the platform is a Raspberry Pi 3 model B+. It will be responsible for all major calculations and algorithms. It runs a Raspbian Jessie operating system, and is capable of communicating with an external computer using WiFi.

3.1.2 Arduino

There are two Arduinos connected to the Raspberry Pi that handle communication with hardware. These are provided as-is by Saab Dynamics and further development is not considered a part of this project.

3.2 Sensors

Several sensors are placed on the platform. Some will be used to estimate the platform's position and others will be used to detect objects and obstacles.

3.2.1 Ultrasonic rangefinders

In order to detect nearby obstacles four SF10 ultrasonic rangefinders will be placed on the platform, directed to the sides. These have a maximum effective range of eleven meters, but will most likely not be used for detection beyond three meters. The rangefinders communicate with one of the Arduinos using I2C. A measurement is done by writing to the command register of the rangefinder and reading the same register until it is no longer 255, which means that the ranging is done and the distance can be read from two other registers.

3.2.2 LIDAR

For long range detection, an SF30-C LIDAR rangefinder will be used. It can be used for distances up to 100 meters, and when connected using USB port it has an update frequency of over 380 Hz. It can be programmed to use a smoothing filter or to have a set resolution. The lowest possible resolution is three centimeters.

3.2.3 IMU

In order to estimate velocity and bearing an IMU will be used. The IMU is composed of several different sensors and can measure acceleration, angular velocity and angles.

3.2.4 Odometers

To measure the traveled path, each track is connected to an odometer. Using these, the rotation of the wheel can be measured and used to estimate the turn rate, velocity and position of the platform.

3.2.5 GPS

The IMU and odometers only provide accurate data in the short term, and only in a relative reference frame. In order to provide an absolute positioning measurement a GPS is mounted to the platform as well.

3.2.6 Camera

A forward-facing camera is be mounted on the platform in order to identify objects and/or add details to the map. It could also provide a real-time feed to an operator.

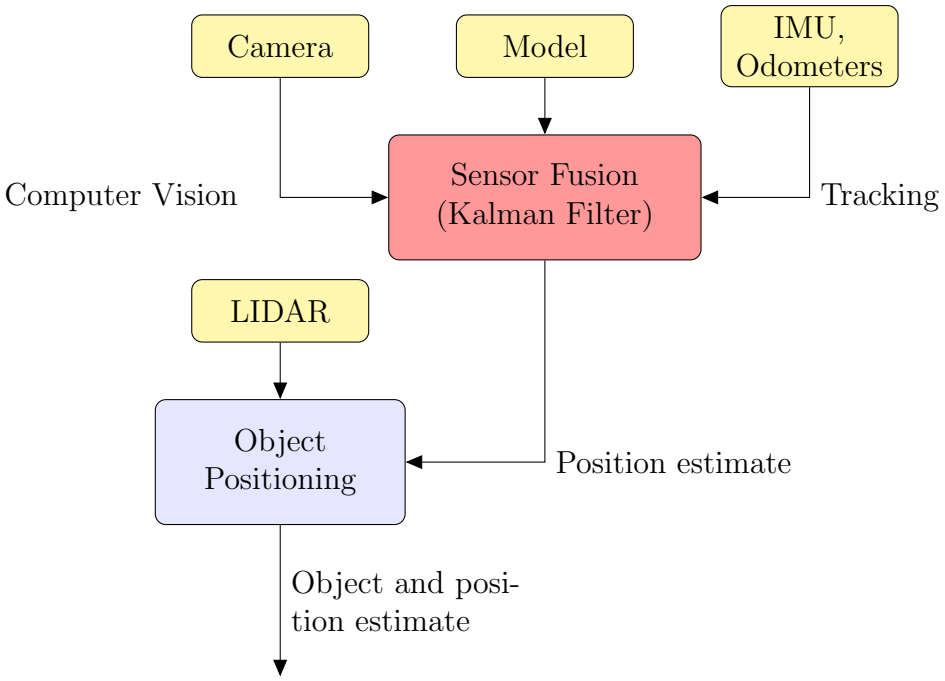

Figure 2: Overview of components and data transfer in the positioning module.

4 Positioning and Mapping

This sub-module is responsible for positioning of the platform in a local and/or global space, as well as mapping of the surrounding space.

There are a number of different sensors on the platform. These will be used to determine the position of the platform as accurately as possible within the map frame.

The platform is equipped with an inertial measurement unit (IMU), a GPS receiver, one LIDAR range finder, a camera, two odometers, and four ultrasonic range finders.

4.1 Module structure

The position will be estimated using combined computer vision (CV) and tracking. The computer vision will be done with the Raspberry Pi camera installed on the platform. This CV implementation will be dependent on certain landmarks (AprilTags) and might therefore not always be able to estimate the platform's position. Tracking will be done using a system model together with the platforms IMU and odometers. Using only a tracking estimate will result in drift but will be able to continuously track the position of the platform. Therefore the model will be expanded to include the CV position, and this model will be used to implement sensor fusion of all measurements. The GPS will be used for positioning the platform in a global environment and the LIDAR and range sensors will be used for object detection and positioning. The structure of the module can be seen in figure 2

4.2 Sub-modules

The design of each submodule will here be explained in further detail and some general implementation details will be discussed.

4.2.1 Computer Vision (CV)

Here the AprilTags Visual Fiducial System will be used to estimate the platform's absolute position in a fix reference frame. The AprilTags library will be used for absolute 3D positioning using fixed markers placed in the environment. Markers can be found in the April tags wiki at https://april.eecs.umich.edu/wiki/AprilTags.

The location of all markers relative to the map are constant and known. If the distance and angle of an AprilTag relative to the platform is determined, it should be possible to combine this information with the AprilTag's constant position to determine the platform's position relative to the map mathematically. In order to get a good approximation of the AprilTag's position the camera will have to be calibrated.

A usage example of the AprilTag API can be found at the AprilTag-C wiki at https://april.eecs.umich.edu/wiki/AprilTags-C. The output of this sub-module should be an estimated position of the platform relative to the map with covariances. Input should be only the picture taken from the Raspberry Pi camera.

4.2.2 GPS

The reason GPS is not used in the fixed frame positioning is partly due to its low accuracy and partly due to performance issues indoors.

The GPS may still be used for positioning the robot in a global environment. The position of fixed reference system can be estimated using the GPS and current position of the platform in the fixed reference system. Using a filter the mapping between the global and local fixed reference system may be estimated placing the robot in a global environment.

4.2.3 Tracking

The tracking module will estimate the platform position using the IMU and odometers. A motion model and measurement model already exist from previous projects, and will be expanded to include the position from the computer vision sub-module.

4.2.4 Measurement fusion

Previous projects have implemented a particle filter, which will be used in this project as well. This particle filter, combined will the motion- and measurement models, will be used in order to combine the computer vision algorithm and sensor data into a position estimate. This filter will have to be designed so that it can handle the situations where no AprilTag is visible and no position estimate can be done using computer vision.

4.2.5 Object detection and positioning

The ultrasonic range finders and LIDAR will be used for obstacle detection.

The Object detection will return positions at which an object has been detected, with a covariance matrix depending on sensor performance at the given distance of the object from the sensor. This will result in points where an obstacle has been detected being delivered to the mapping module.

The camera could be used here to improve detection performance by angling the platform for range detection. Basing the detection on whether the camera finds potential objects.

4.2.6 Mapping

The map generated by the laser-based SLAM will be expanded to include a grid, composed of $0.5 \cdot 0.5$ tiles that can be explored or unexplored, in order to keep track of where the *Balrog* has been. This map can then be used to determine a path to the closest unexplored tile. The tiles will be open to expansions in order to contain additional information, such as ground colour or slope.

4.3 Interface

Internal interface between sub modules and external interface between the positioning module and other modules are described below.

4.3.1 Internal

Internal communication within the positioning module will be in the form of state vectors with associated covariance matrices. Here each position estimate will produce a state vector that will be used in the measurement fusion. The measurement fusion will result in an additional state vector for the external interface.

4.3.2 External

Communication between the positioning module and the sensors will be done using the provided HAL toolbox, which has been developed by Saab Dynamics and CDIO projects from previous years. This toolbox contains functions used for initialisation and reading from sensors.

The positioning module will update the state vector containing the states of the vehicle. The state vector will therefore have to be passed by reference to the positioning module.

The object position measurement will be passed as an object list where each measured object will be a struct containing a position and covariance matrix.

It might be possible to improve the positioning using the LIDAR and a SLAM algorithm. By using gmapping or openSLAM, it might be possible to determine both the position of the obstacles and the platform. The SLAM approach could be further improved during the mapping phase using a extended Kalman filter or particle filter.

5 Navigation

The navigation module will be responsible for determining the path that the control module will attempt to follow. It will be adaptive, and update the route if a new obstacle that obstructs the current path is detected. There will be two available modes for the navigation module:

- Exploring mode.
- Guided mode.

The exploring mode will continuously find paths to unexplored tiles until the whole area is explored, while the guided mode will find a path to a user-selected coordinate. The map that it will use as a basis for the search is divided into a number of tiles, and as such there are different search algorithms that can be used. The algorithms which will be implemented are the following:

- Uniform-cost search.
- Greedy search.

The user will be able to choose which search algorithm is to be used through the GUI.

5.1 Implementation of search algorithms

The following subsection will give a description about how the search algorithm will be implemented. Finding the optimal route to explore all tiles is problematic, both because it is computationally expensive and because the map will be updated with information about obstacles over time. Therefore, the navigation algorithm will use a *Uniform-cost search* or a *Greedy search* to find the next undiscovered tile which takes the least time to arrive to, referred to as *cost*. The algorithm will base its results on the properties of each tile. The following tile properties might be of interest for the navigation system:

- Obstacles - If there is an obstacle in a tile, the proposed navigation route should not go through that tile.
- Type of ground - It might take more time to drive through sand than grass.
- Slope - It could take more time to try to climb steep slopes.
- Explored - If a tile has been explored, one could make it expensive to travel to tiles without adjacent explored tiles. This promotes a search pattern where the searched area is continuous.

5.1.1 Uniform-cost search

First and foremost, the goal of the search is to find the cheapest unexplored tile. Listed below is a sequence of events that implements this.

1. Create a list of tiles, *listOfTiles_list*, with the current tile as the only object in the list. This tile should have cost 0.
2. Pick the cheapest tile in *listOfTiles_list*.
3. Check if the tile is unexplored, if true then *return* the route to the tile which is based on the previous tiles.
4. Remove the tile from *listOfTiles_list* and add all adjacent tiles which cannot be found in *listOfTiles*, which will decrease the search complexity, to the list with respective cost and previous tiles as properties.
5. Iterate from 2.

5.1.2 Greedy search

Greedy search works a lot like *Uniform-cost search* except that instead of a graph search it works as a tree search. That means it will choose the cheapest tile at every decision, creating an non complete and non optimal search method. The algorithm is described in steps below:

1. Create a list of tiles, *listOfTiles_list*, with the current tile as the only object in the list.
2. Pick the cheapest tile in *listOfTiles_list*.
3. Check if the tile is unexplored, if true then *return* the route to the tile which is based on the previous tiles.
4. Remove all tiles from *listOfTiles_list* and add all adjacent tiles for the cheapest tile to the list with respective cost and previous tiles as properties.
5. Iterate from 2.

5.1.3 Cost function

Both *Uniform-cost search* and *Greedy search* are based on the cost of each tile which are evaluated by the cost function. The cost function should represent the time to reach a certain tile but it should also promote a continuous explored area to minimize the amount of isolated unexplored tiles when the map is almost completely searched. One alternative to achieve this is to punish turning more than driving forward and punishing tiles without adjacent explored tiles. One possible cost functions which promotes this behaviour is seen in (1).

$$f_{cost}(tile_{current}, tile_{goal}) = \#forward + 2 * \#turns + \#unexplored\_adjacent\_tiles(tile_{goal}) \quad (1)$$

5.2 Module structure

The navigation module will use the map generated by the mapping system to provide the control system with a route which it will try to follow. The generated map will be composed by a number of tiles. Each tile will have properties which describes the attributes of the tile in the real world. These attributes can be used by the search algorithm to determine the cost of the path. The navigation module will either find a path to a specified destination, or a path that leads to a nearby unexplored tile. If a new obstacle obstructing the current path is detected, the navigation module will detect this and generate a new path.

5.3 Sub-modules

The navigation module can be divided into two tasks, one active and one passive. The active one will be responsible for path generation, and the passive one will detect when the current path is obstructed. The navigation module will select a destination in two ways, depending on the current mode. If the platform is exploring, there is no destination initially, the destination will instead be selected when the algorithm finds an unexplored tile and keep iterating until the whole map is explored. If the user has selected a destination, this will be the tile to which the navigation algorithm will attempt to find a path.

5.3.1 Path finding

Whether the platform is in exploring or guide mode, the navigation module will be responsible for finding a path. The path will depend on the chosen search algorithm. The difference between the two user modes, exploring and guided mode, are the goal state. In the prior case the goal state will be an unexplored tile and in the latter the coordinates of the tile.

5.3.2 Obstacle avoidance

When the navigation module has found a path, it will receive updates from the positioning module when a tile that was previously considered to be clear has an obstacle. If this new obstacle is in the current path the navigation module will restart its search for the closest unexplored tile. If the obstacle is at the destination, the platform will behave differently depending on the current mode. If it is exploring it will find a new unexplored tile if possible. If the platform is attempting to reach a user-selected destination, that destination is now unreachable and the platform will stop and wait for new instructions.

5.4 Interface

The navigation module will send decisions to the GUI using the communicator. When the desired search algorithm is changed by the GUI this will be sent via the communicator. The navigation module will send a reference state vector to the controller module. Before doing a search it will send a request for the latest map to the positioning module and

receive an struct as a response. During navigation, the positioning module will send the location of tiles where new obstacles are detected.

6 Automatic Control

The automatic control (AC) module is a tool available to the navigation agent. In autonomous mode the navigation agent uses the AC module to perform its search in a completely independent fashion. Without the AC module the navigation module can only give suggestions to a human operator who then has to perform them manually.

6.1 Module structure

The control unit will ensure that the robot follows the trajectory set by the navigation module. A Linear-quadratic regulator will be implemented so that the robot follows the reference trajectory set by the navigation unit. A Collision detection sub-module will be implemented as a fail-safe to avoid collision in the case that the mapping or navigation modules make an error. The collision detection sub-module will here alter the control signal of the LQ-regulator to avoid collision. The structure of the control unit can be seen in figure 3

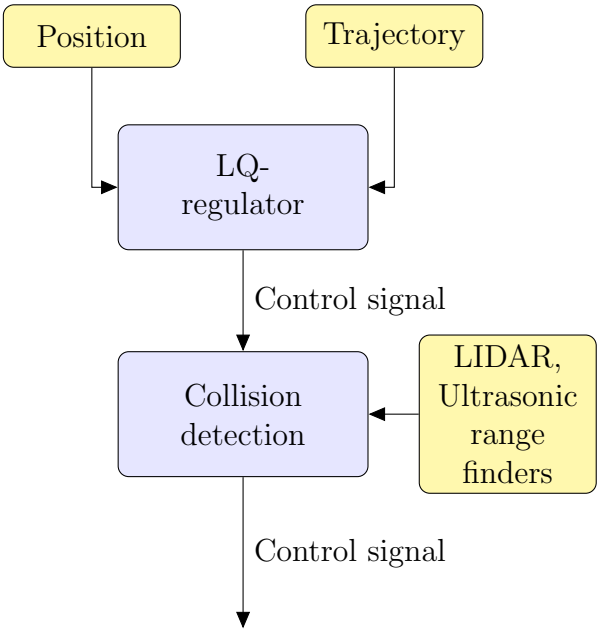

Figure 3: Overview over the structure of the control module.

6.2 Sub-modules

The design of each sub-module will here be explained in further detail and some general implementation details will be discussed.

6.2.1 Controller

The controller will follow the reference given by the navigation unit using a LQR structure. The LQ regulator is estimated using MATLAB system control toolbox. The system will here be described using the state model from section 2.1.

The Q matrices will initially be set so that the main penalty in the regulator is that of the control error.

Functionality to alter the control parameters and estimate a new controller through the GUI will then be implemented for trimming the system.

Here the reference is given by the navigation module while the measurements is given by the positioning module.

6.2.2 Collision detection

Filtered distance measurements from the LIDAR and range sensors will be used for collision detection to prevent damage to the robot. In case of a detected collision the controller will avoid collision and ask for a new trajectory from the navigation unit. Collisions will be avoided by altering the already determined control signal from the LQ controller.

6.3 Interface

The controller will receive sensor data and send the control signal to the DC-engine using the already implemented HAL.

Using the communicator, the controller will send decision information to the GUI and receive control parameters and settings from the GUI.

Interface between the Navigation and Positioning modules will be done using state vectors where both the position and trajectory will be represented using system states.

7 Communicator

In order to facilitate communication between the GUI and the *Balrog* there will be a communication module, or communicator. It will act as a buffer between the TCP/IP link and the other modules on the *Balrog*.

7.1 Module structure

In order to handle transmitting and receiving data, the communicator will:

- Receive data from all modules.
- Send data over the TCP/IP link.
- Receive data over the TCP/IP link.
- Pass data along to the proper recipient.

To simplify these operations, all data will have 4 identifiers: the sender, the recipient, message type, and the size of the message. These identifiers are defined in tables 1, 2, 3, 4, 5, and 6. The general sizes of different messages are defined in table 7.

7.2 Sub-modules

The communicator will be divided into 2 parts, the receiver and the transmitter. These will both communicate with the modules as well as the GUI.

7.2.1 Receiver

The receiver will receive data transmitted from the GUI and store it in a buffer. Using the identifiers embedded in the message it will then determine the proper recipient and forward the message to that module.

7.2.2 transmitter

The transmitter will receive data from all the modules. It will add the proper identifiers to these messages and then forward them to the GUI at appropriate intervals.

7.3 Interface

The communicator will communicate with the GUI using TCP/IP. This is further described in section 8. The communicator will also send and receive data from the other 3 modules on the *Balrog*. This is described in greater detail in sections 4, 5, and 6.

8 Graphical User Interface

The main task of the GUI is to provide the operator with clear information about the decision process of *Balrog* in real time, and to allow the user to modify internal settings in the *Balrog*. The GUI is to be implemented in C++. An early draft of the GUI can be seen in figure 4.

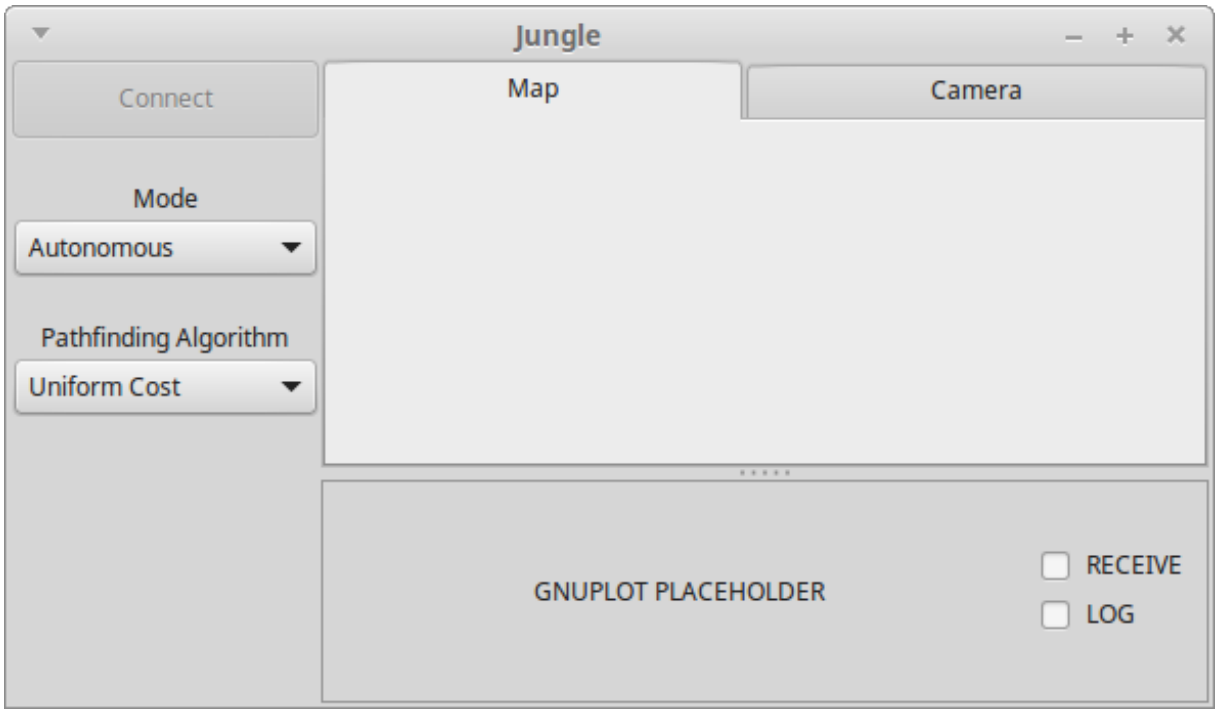

Figure 4: An incomplete version of what the GUI will look like.

8.1 Sub-modules

The program will be multi-threaded. One thread is responsible for rendering the GUI and receiving user inputs. The other thread is responsible for network communication and retrieving data.

There will also be a module that runs on the Balrog itself. This module will collect sensor data and transmit it to the GUI. This module will be structurally simple, essentially a finite state machine that toggles data feeds, along with a message passing function to redirect received commands to the correct modules.

8.1.1 Data display

One primary feature of the GUI will be to show which parts of the map the *Balrog* has explored, and its position relative to them. As such, the communication module on the *Balrog* will transmit its location to the GUI, along with any tiles it explores. Transmitting detected obstacles might put stress on the channel, and is therefore optional. On this map, any routes selected by *Balrog* will also be shown.

For each of the sensors there will be a graph, plotting the sensor value over time. When there are more than one sensor of one type, they will be plotted in the same graph with different colors. These graphs can be toggled individually. There will also be a text field, to which the *Balrog* can send decisions, warnings and notifications. This text field can be toggled as well.

When filming at decent quality, the stream rate will be about 15 Mbit/s. This means that with a good connection, it will be possible to stream the camera view directly to the GUI. Since this stream will require a large portion of the connection it will be possible to toggle.

8.1.2 Options

When the map has been explored to the fullest extent possible, it will be possible to set a destination that the *Balrog* will attempt to reach.

It will be possible to choose between all implemented search algorithms in order to test and troubleshoot them.

The *Balrog* will have several modes, such as exploration, navigation and remote controlled. The current mode can be selected from the GUI.

8.1.3 Communicator

The Communicator will be the code on the *Balrog* responsible for collating sensor data and transmitting it to the GUI. It also acts as a hub, distributing commands sent from the GUI to the modules.

The Communicator also implements Dead Man's Grip functionality. The Communicator sends regular pings to the GUI, and if the GUI ever fails to respond in time then the Communicator will issue Emergency Stop messages to all modules.

Whenever connection is re-initiated the Communicator will receive a full desired state description from the GUI.

8.1.4 External interface

Communication between Balrog and the GUI will be transmitted over standard WLAN 802.11 wireless link. The Raspberry Pi 3 model B+ supports 802.11n Wireless LAN.

IEEE 802.11n can maintain single-antenna baud rates of up to 72 Mbit/s, but this is very optimistic for real-life applications. Assuming a TCP/IP throughput decrease of about 50% along with 1/10th working speed this gives an approximate 3 Mbit/s effective data rate.

This rate should be enough to transmit full sensor data, as well as compressed video, but individual data streams shall be possible to disable or decimate before transmission in order to reduce the demands on the wireless link.

Data transmission will use TCP/IP along with h.264 video compression algorithm.

Module	Identifier
GUI	0x01
Communicator	0x02
Positioning	0x03
Mapping	0x04
Control	0x05
Navigation	0x06

Table 1: Definitions of identifiers for all modules.

Command	Identifier
Ping	0x00
Start	0x01
Stop	0x02
Emergency Stop	0x03
Reset	0x04
Send Status	0x05
Dump State	0x06
Log Text	0x07

Table 2: Definitions of identifiers for all commands.

Data Stream	Identifier
State Vector	0x01
Internal Map	0x02
Control Signal	0x03
Mapping	0x04
Control	0x05

Table 3: Definitions of identifiers for data streams.

Range Sensors	Identifier
LIDAR	0x06
FL Ultrasound	0x07
FR Ultrasound	0x08
BL Ultrasound	0x09
BR Ultrasound	0x0A

Table 4: Definitions of identifiers for range sensor data.

8.1.5 Communication Protocol

Messages are formatted according to the following structure.

[2 Byte - Sending Module] — [2 Byte - Receiver name] — [2 Byte - Message Type] — [2 Byte - Number of samples] [N Bytes - Data]

Table X gives the identifiers that specify what module a message is addressed to.

Each module can specify its own commands but all modules need to include and respond to the predefined commands in table 2. The command values in the range 0x00-0x0F are reserved and may not be used by other modules.

Each data stream (from sensors or otherwise) has a unique identifier to distinguish them. These are defined in tables 3, 4, 5, and 6.

The minimum TCP/IP message size is 40 bytes when not transmitting any data.

The maximum message size for a TCP/IP packet is 64 Kb, so this will not be a problem.

IMU Streams	Identifier
X-Accelerometer	0x0B
Y-Accelerometer	0x0C
Z-Accelerometer	0x0D
X-Axis Gyro	0x0E
Y-Axis Gyro	0x0F
Z-Axis Gyro	0x10

Table 5: Definitions of identifiers for IMU data.

Other Sensors	Identifier
GPS Longitude	0x11
GPS Latitude	0x12
GPS Altitude	0x13
Left Odometer	0x14
Right Odometer	0x15
Text Message	0x16

Table 6: Definitions of identifiers for other sensor data.

Data Stream	Size (Byte)	Structure
State Vector	32	3 float, 2 double
Internal Map	Map size	IxJ uint8
Control Signal	2	2 uint8
Mapping		
Control	8	2 double
X-Accelerometer	8	1 float
Y-Accelerometer	8	1 float
Z-Accelerometer	8	1 float
X-Axis Gyro	8	1 float
Y-Axis Gyro	8	1 float
Z-Axis Gyro	8	1 float
LIDAR	2	2 uint8
FL Ultrasound	2	2 uint8
FR Ultrasound	2	2 uint8
BL Ultrasound	2	2 uint8
BR Ultrasound	2	2 uint8
GPS Longitude	4	1 double
GPS Latitude	4	1 double
GPS Altitude	4	1 double
Left Odometer	1	1 uint8
Right Odometer	1	1 uint8
Text Message	1	1 uint8

Table 7: The size and composition of all messages.

References

[1] *LIPS – nivå 1. Version 1.0.* Tomas Svensson och Christian Krysander. Compendium, LiTH, 2002.