System Draft

Editor: Ingrid Sjöberg

Version 1.0

Status

Reviewed	Erik Bodin	2016–12–12
Approved	Martin Lindfors	2016-09-23

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

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	2016-09-16	First draft	*	EB
0.2	2016-09-19	Second draft	*	EB
0.3	2016-09-21	Third draft	*	EB
1.0	2016-09-23	Approved	*	EB

1 Introduction

This is the system draft 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 to unify 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. There the *Balrog* platform transmits sensor data and receives control instructions for the tracks. These components will not be modified. The scope of the main program is marked with a green square and includes mapping, 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 a brief explanation of the hardware of the system together with a explanation of the software system that is to be developed in the project.

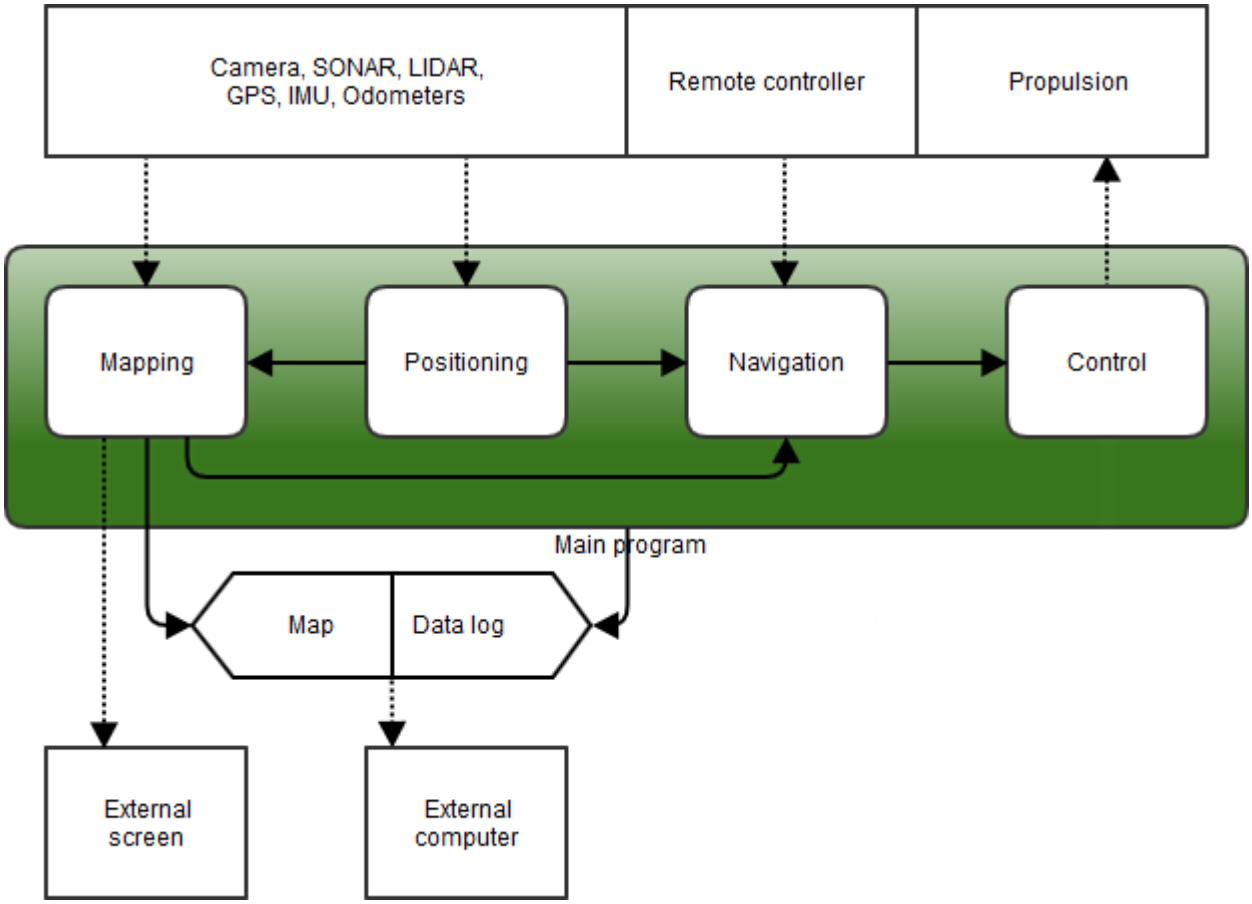

Figure 1: Illustration of the provided system together with the system that is to be developed during the project.

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, or something else.
- *HAL* - A Hardware Abstraction Layer provides a unified high-level interface to access low level hardware functionality. This is so programmers don't have to consider the details of signal timing, polling, or similar hardware considerations.

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.

2.1 Sustainability

Sustainable code is a core requirement of the finished product and must therefore take 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 Mathworks found at "http://se.mathworks.com/matlabcentral/fileexchange/46056-matlab-style-guidelines-2-0"

2.2 Offline environment

For development purposes an offline environment is an important tool to be able to test the implementation. It is often a tedious task to test the implementation against hardware, and the ability to run implementations against already recorded data might prove to be an important tool in the development process. This can be seen as very simple type of software in the loop implementation where core functionality as positioning, mapping and navigation can quickly be tested against actual data during development.

Some functionality such as autonomous control would still have to be developed directly against hardware, given the lack of any type of simulation in this type of SIL implementation. This could be done by implementing a virtual mine sweeper that simply returns previously recorded data in the same manner as calls to the actual hardware API would.

3 Hardware

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 be 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 380Hz. 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.

4 Software

The core of the developed software system will be composed of five major components:

- *Positioning* - Responsible for estimating the state of *Balrog*.
- *Mapping* - Will use the data gathered from the ranging sensors and camera to generate a map of the surroundings.
- *Navigation* - Will use the generated map to calculate a route for *Balrog* to follow.
- *Control* - Will steer the actuators of *Balrog* to achieve the state given by *Navigation*.
- *Graphical User Interface* - Will provide a clear visualization of the positioning, mapping and navigation states.

The following sections aims to elaborate how these systems will work while still not providing complete implementation details.

4.1 Positioning

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 ultrasound range finders.

4.1.1 Interface

Communication between the positioning module and the sensors will be done using the provided HAL developed by Saab Bofors Dynamics and CDIO projects from previous years.

The positioning module will update the state vector containing the states of the vehicle including but not limited to position, speed, absolute angle and rotational speed.

4.1.2 Sensor fusion

Multiple sensors will be used to estimate the position of the platform and surrounding objects.

One suggested approach is to use the IMU, and odometers to estimate the platforms position. Here the AprilTags Visual Fiducial System could be used to estimate the platform's absolute position using the camera to avoid drift. The ultrasound and LIDAR will be used to detect obstacles, the camera could be used here to improve detection performance.

The reason GPS is not used in positioning is part due to its low accuracy and part due to performance issues indoors. The GPS might still be used for positioning the robot in a global environment.

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.

4.2 Mapping

In order to properly navigate and store data about the surrounding environment, the *Balrog* needs a good internal map model. This model needs to contain previous positions of the *Balrog* along with possible detected obstacles, mines, and other information about the environment. The mapping algorithm will be responsible for adding objects detected by the positioning algorithm to the internal map, along with possibly determining additional information about the terrain.

4.2.1 Basic structure

The basic structure for the map could be implemented as two layers. The first layer would the *Balrog's* recorded positions, represented with possible confidence bounds. This would then be drawn atop the second layer, which would be a grid with a reasonable resolution. For instance, the *Balrog* is about $0.5 \cdot 0.5$ meters, so this could be one option for the grid's resolution. Each tile in this grid could then possess information about the environment measured at that position, such as:

- *Obstacles:* If the LIDAR or ultrasonic rangefinders detect something above ground level, this could be added to the map at the estimated position of that reading. Multiple clustered detections could possibly be combined into one large object. Information about these obstacles could then be used to navigate the explored area.
- *Visual information:* It could be possible to use the camera to gather information about the environment. A simple implementation would be to in advance determine which portions of the camera image corresponds to the square in front of the platform, and extract the mean of the colour in that segment. That color could then be assigned to the correct square.
- *Topography:* Depending on the accuracy of the angles and accelerations of the IMU, it might be possible to estimate the *Balrog's* change in height. Combining the angle and the odometers could give the distance traveled along the z-axis, and combining this with the measured acceleration might give a useful approximation of the height. This could then be included in a 3D map. This could however be time consuming and might not add much relevant information, so this will only implemented as time permits.
- *Slope:* By measuring the angle at each position the mapping function can assign a value to the explored tiles, which can then be used by the navigation function.

When the *Balrog* has finished exploring it might not have complete data of every position. Therefore an algorithm to interpolate the properties of unexplored tiles would be needed. For instance the height of unexplored tiles could be interpolated from surrounding explored tiles. An example of how a map with topography and visual information could look can be seen in figure 2.

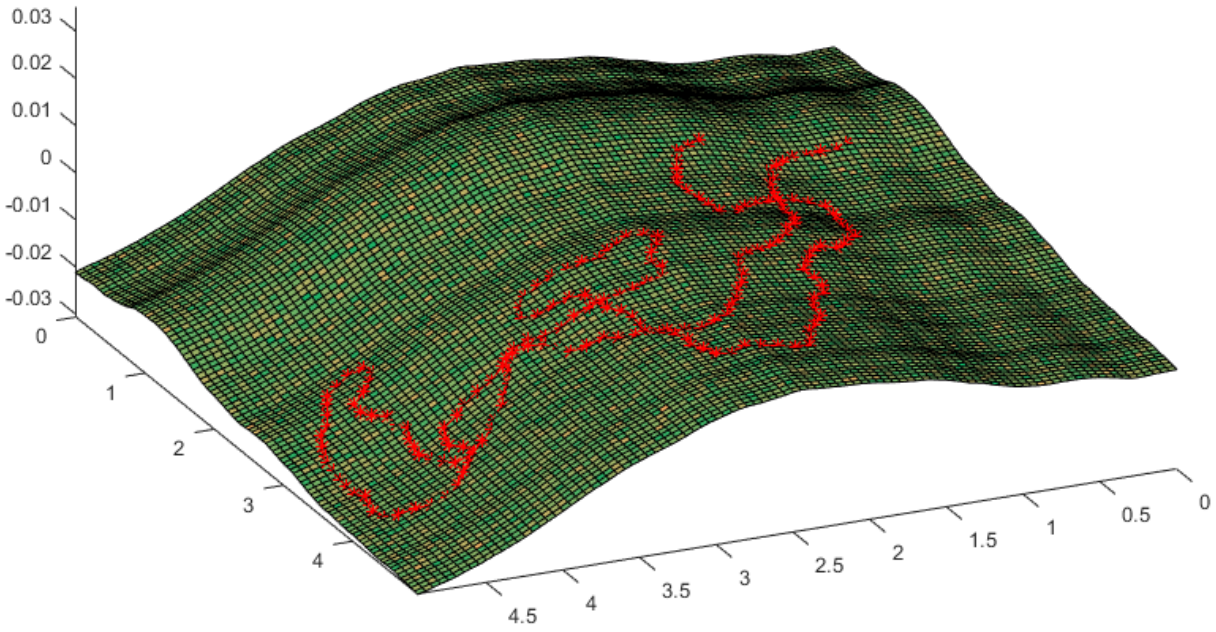

Figure 2: Example of a map where all the above properties have been implemented.

4.3 Navigation

The *navigation system* 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. Depending on the performance of the positioning the resolution of this map can be of various size. Each tile will have properties which describes the attributes of the tile in the real world.

As it is too computationally expensive to find an optimal route to cover each undiscovered tile, the navigation algorithm might use a *Uniform-cost search* to find the next undiscovered tile which takes the least time to arrive to, referred to as *cost*. The algorithm will then 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 be very slow 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 very contiguous.

4.3.1 Implementation of *Uniform-cost search*

The following section should give a hint about how the search algorithm could be implemented. First and foremost, the goal of the search is to find the cheapest unexplored tile. An appropriate order of events to do this is listed below.

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 to the list with respectively cost and previously tiles as properties.
5. Iterate from 2.

Note that there are several more functions regarding for example *search complexity* that needs to be developed.

4.4 Automatic Control

The automatic control 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.

4.4.1 Interface

The AC component requires commands from the navigation component. It also needs accurate state estimates that are provided by the positioning module. The output interface for the AC module is the wheel speeds. The interface between the AC component and the navigation component could be implemented in a number of ways which will be discussed.

A minimalistic approach to the control interface would be to simply give the control unit a heading or a vehicle angle to control against. This approach would leave the control unit very minimalistic and maybe even able to be implemented using a simple PD-controller.

For improved robustness a goal position could also be implemented in the interface allowing the controller to better determine desired speeds.

A more rigorous solution to the interface could be to include future values in the reference signal to the controller. Knowing future values of both desired position and vehicle angle allow the controller to better plan its output signal.

Given MATLAB's copy on edit implementation and that the data is needed to be saved by the navigation component the latter would to be preferred, due to the extra free information that may be used by the controller.

4.4.2 Controller

A controller will be implemented that follows the reference trajectory provided by the routing component. Here a model of the vehicle may be required in order to achieve desired performance.

Both LQ and PID based control algorithms have their advantages and disadvantages. Both will have to be investigated for the final design but either is expected to give adequate results.

Given that the controller has access to future reference values according to section 4.4.1 a control structure that handles this is appropriate but not required. An alternative would be that the interface are allowed to give a trajectory as reference that then would need to be translated into a reference signal to control against.

4.5 Graphical User Interface

The main task of the GUI is to provide the observers with clear information about the decision process of *Balrog* in real time. To accomplish this several attributes need to be displayed.

The following properties are to be shown:

- *Positions* - The estimated position of various obstacles.
- *Map* - The generated map will be shown as a background to the other properties.
- *Route* - The route which *Balrog* is intended to follow.

The GUI is to be implemented in C++.

4.5.1 Communication

Communication betwen 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.

The exact protocols for data transmissions are yet to be determined but TCP/IP along with some low-processing video compression algorithm seem like reasonable first picks.

References

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