Design Specification: Autonomous Rescue System with UAV and Tracked Vehicle

Version 0.5

Editor: David Ryberg
Date: October 26, 2018

Status

Reviewed	Benjamin Lembke	25/10 2018
Approved		

Course name:	Reglerteknisk projektkurs, CDIO	E-mail (@googlegroups.com):	rescuerangers
Project group:	Rescue Rangers	Document responsible:	David Ryberg
Course code:	TSRT10	Editors's student ID:	davry764
Project:	Autonomous Rescue System	Document name:	Design Specification

Project Identity

Group E-mail:	rescuerangers@googlegroups.com
Homepage:	
Client:	Martin Lindfors, Linköping University
	E-mail: martin.lindfors@liu.se
Customer:	Torbjörn Crona, Saab Dynamics
	E-mail: torbjorn.crona@saabgroup.com
Course Responsible:	Daniel Axehill, Linköping University
	E-mail: daniel.axehill@liu.se
Advisors:	Magnus Malmström, Linköping University
	E-mail: magnus.malmstrom@liu.se
	Erik Ekelund, Saab Dynamics
	E-mail: erik.ekelund@saabgroup.com
	Axel Reizenstein, Saab Dynamics
	E-mail: axel.reizenstein@saabgroup.com

Group Members

Initials	Name	Responsibility	Phone	E-mail (@student.liu.se)
BL	Benjamin Lembke	Head of Design	070-6394109	benle163
DR	David Ryberg	Head of Documentation	073-0364421	davry764
DE	Dennis Edblom	Head of Software	076-5938131	dened825
EH	Edvin Hansson	Head of Information	073-3004604	edvha762
EO	Emma Olsson	Head of ROS	076-7672023	emmol877
LB	Linn Berntsson	Project Manager	076-8678846	linbe413
MJ	Marcus Jackson	Head of Hardware	070-7653786	marja928
TB	Tobias Bengtsson	Head of Testing	070-6715555	tobbe592

Document History

Version	Date	Changes made	Reviewer
0.1	24/9 2018	First Draft	DR
0.2	1/10 2018	First Revision	DR
0.3	4/10 2018	Second Revision	DR
0.4	10/10 2018	Third Revision	LB
0.5	25/10 2018	Fourth revision	BL, EO

Contents

1 Introduction **1**

1.1 Definitions . . . 1

2 System Overview **3**

3 ROS: Robot Operating System **3**

3.1 ROS Theory . . . 3

3.2 ROS functions & implementation . . . 3

4 Tracked Vehicle Balrog **5**

4.1 Computing Hardware . . . 5

4.2 Sensors . . . 6

4.2.1 LiDAR . . . 6

4.3 Manual mode . . . 6

4.4 Autonomous mode . . . 6

4.4.1 Positioning . . . 7

4.4.2 Route planning . . . 10

4.4.3 Obstacle avoidance . . . 11

4.4.4 Route Following . . . 11

4.4.5 AprilTag Detection and Supply Drop-Off . . . 11

5 Quadcopter Sauron **13**

5.1 Hardware . . . 14

5.2 Sensors . . . 14

5.3 Positioning . . . 14

5.4 Overall control and route following . . . 16

5.4.1 Refererence trajectory . . . 16

5.4.2 Control problem . . . 17

5.5 Flight Control . . . 18

5.5.1 Height control . . . 18

5.5.2 Speed control . . . 18

5.5.3 Take off and Landing . . . 18

5.6 Manual Mode . . . 19

5.7 Detection and identification . . . 19

5.7.1 Detection . . . 19

5.7.2 Identification . . . 19

6 Communication **21**

6.1 Communication between Balrog and Base Station . . . 21

6.2 Communication between Sauron and Base Station . . . 21

6.3 Communication between Balrog and Sauron . . . 21

7 Graphical User Interface **22**

1 Introduction

This document is a design specification defined for a CDIO-project called "Autonomous Rescue System with UAV and Tracked Vehicle" in the course TSRT10, given at Linköping University. The document aims to provide an overview of the system as a whole and also provide detail about each subsystem included in the project. The project aims to implement an UAV (Unmanned Aerial Vehicle) and a tracked vehicle. The communication between the two vehicles aims to be done autonomously so that the operation can be coordinated between them.

This document is written according to the LIPS-model [1].

1.1 Definitions

Here are the definitions which are used in this document:

- **Sauron** - A drone/UAV/Quadcopter.
- **Balrog** - The tracked ground vehicle used to deliver rescuing supplies.
- **Base Station** - The starting position for both robots.
- **AprilTag** - A matrix of bar codes that can be detected by the robots.
- **Person/People in distress** - The target(s) the autonomous vehicles will search for. Will be represented by an AprilTag.
- **AprilTag box** - A representation of a person in distress as a box with AprilTags on all sides.
- **Supply** - A package that the robots will deliver to a person in distress. Will be represented by a standard 33 centilitre soda can.
- **Obstacles** - Objects designed to hinder movement inside the test area.
- **Operation** - The objective for the project. Sauron will identify a number of people in distress within the test area, find their positions and send this data to Balrog. After obtaining the positional data from Sauron, Balrog should create a plan for how to deliver supplies to each of the given positions and execute it autonomously. Balrog also identifies and avoids obstacles while moving around the test area. The operation ends when Sauron has made sure that there are no further people in distress within the test area, Balrog has delivered supplies to each of them, and both Sauron and Balrog have returned to the starting point.
- **Detection of person in distress** - A person in distress is considered to be detected when a frame from the onboard camera on Sauron detects an AprilTag. Two neighboring detections means that two neighboring frames from the camera feed has detected an AprilTag.
- **Identification of person in distress** - A person in distress is considered to be identified when Sauron has had a 90% detection percentage during 2 s. Only identified people in distress will be visited by Balrog.
- **User** - A person interacting with the system.
- **GUI** - Graphical User Interface.

- **SLAM** - Simultaneous Localization And Mapping.
- **LiDAR** - Light Detection And Ranging sensor.
- **ROS** - Robot Operation System.
- **IMU** - Inertial Measurement Unit

2 System Overview

In this chapter, the system is introduced and connections between the different subsystems are explained. The Rescue system consists of three main subsystems; The Base Station, Sauron and Balrog. Sauron is an UAV, which tasks are to automatically search and find a person in distress, determine the location of this person and send the location data to Balrog over a Wi-Fi connection. Balrog is a tracked vehicle that upon received location data from Sauron automatically can navigate to the person in distress. The communication with Base Station from Sauron or Balrog is made using a Wi-Fi connection. This applies for the communication between Sauron and Balrog. Base Station is a computer running ROS with the purpose to communicate and display sensor data on separate GUI:s for the other platforms, i.e. Sauron and Balrog. With remote controllers both Balrog and Sauron can be controlled manually by a human user. A schematic overview of the system can be seen in figure 1.

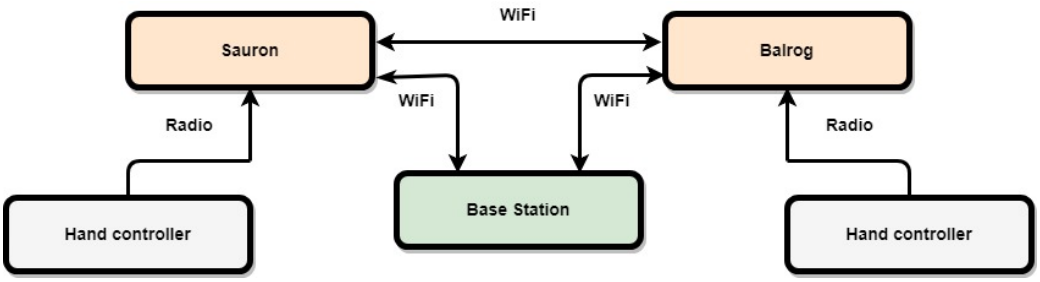

Figure 1: Schematic overview of the Rescue System.

3 ROS: Robot Operating System

In this section, the Robot Operating System (ROS) is introduced and explained for a deeper understanding.

3.1 ROS Theory

ROS is an open source software with a collaborative development system. ROS enables communication between different modules in a robotic system through a set of libraries and tools. It is also possible to integrate with other libraries, such as OpenCV and Gazebo, which enables the user in a simple way to simulate and integrate functions relying on computer vision, path planning and much more without needing to change extensive lines of code. ROS runs in a Linux environment with code support for C++ and Python.

3.2 ROS functions & implementation

- **catkin workspace** - a folder where the user can install, build or modify catkin packages. Catkin is the official building system for ROS. Catkin generates targets (libraries, executable programs etc., i.e. anything but static code) from raw source code. It is possible to build several, interdependent packages at the same time and

in the same workspace. A catkin package can be built in the same way as a CMake project. [1]

A catkin workspace usually contain up to four different spaces (sub-folders). *Source space* contains catkin packages source code. The recommendation is to keep this space unchanged by installing, configuring or building.

Build space contains files and cache for CMake and catkin. In this space the build of catkin packages into *Source space* is invoked by CMake. The recommendation is to contain *Build space* in the catkin workspace and placed outside the *Source space*.

Devel space stands for development space. It contains the built targets before they are installed. This environment can be used to do testing and development. The organization of the targets equals the layout for installing them. The *Devel space* is recommended to be a peer of the *Build space* directory.

Install space contains the built target after their installation. The install location is set by default to /user/local, but this is not a good destination for ROS packages, therefore the install location should be changed. Other than that there is no recommendation for the *Install space* placement.

- **Nodes** - A node controls a computing process and ROS usually have many nodes controlling different computing processes. One node might provide a live-stream of a sensor and another might control a quadcopters propeller motors while another plans the quadcopters path. A ROS client library is used to write a node, using `roscpp` or `rospy` depending if writing in C ++ or in Python.
- **Messages** - The nodes communicate through messages, composed as a data structure.
- **ROS Topic** - The name of a topic identify the content of the message and topics are the named buses used for nodes to send messages over. Each topic is defined by the type of ROS message used to publish it. A node can receive any message as long as their types match. Current transport that are supported in ROS are TCP/IP-based and UDP-based message transport, where TCP/IP stands for Transmission Control Protocol/Internet Protocol and UDP stands for User Datagram Protocol. To interact with ROS topics the command-line tool `rostopic` can be used.
- **Master** - Name registration for services and topics, a look-up system. Nodes connects directly to each other, the Master only enables nodes to find each other, lets them invoke services and exchange messages.
- **Publish/Subscribe** - A Publishers and subscribers are normally not aware of each other. A subscriber that has an interest of a certain kind of data can subscribe to a topic and receive messages from nodes publishing those kind of data. There is no limit to how many nodes a node can publish and subscribe to.
- **Services** - For request and reply interactions, that is more suitable for situations craving a 2-way specific data exchange. With use of a provider client system can a client request a service by the name set by the provider of the service and wait for the response.
- **Bags** - Stores message data and provides possibility to access this data and play it back. ROS message data can be sensor data, communication data and so on.

[1] CMake is a open source software where you can compile, build and test packages.

4 Tracked Vehicle Balrog

This section provides an in-depth view of the functionality and subsystems of Balrog, as well as a list of the most essential components of the robot. Most parts of the hardware are a products from previous projects, and will be considered as one component. An overview of the hardware components comprising Balrog is given in figure 2.

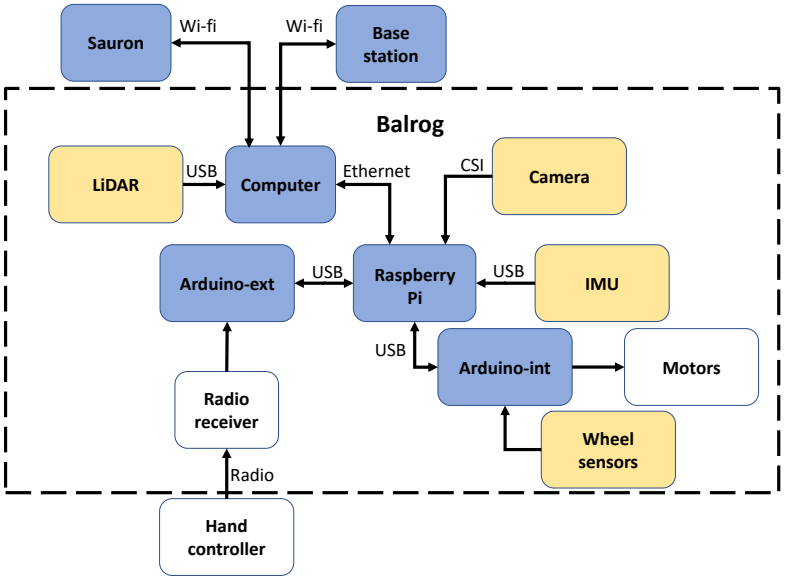

Figure 2: Overview of the hardware components and their connections in Balrog.

The Figure also provides some detail about how data is transported between components in Balrog and the parts of the system connected to it.

4.1 Computing Hardware

From the description in figure 2 of the hardware components included in Balrog, the following contributes to Balrog's internal computing capability in the following ways:

- **Computer**: The computer aboard Balrog serves many purposes. Firstly, it serves as communication link to Base Station computer and Sauron via Wi-Fi. Secondly, the computer runs ROS and thirdly it receives sensor data from the Raspberry Pi and LiDAR.
- **Arduino-int:** An Arduino that handles the propulsion and the motors as well as the wheel sensors. Connected to the Raspberry Pi via USB.
- **Arduino-ext:** Another Arduino. This one handles communication with the hand controller via radio. Also connected to the Raspberry Pi via USB.
- **Raspberry Pi:** A Raspberry Pi 3 B+ that is the main communication hub within the Balrog. It receives the video data from the camera and forwards it to the on-board computer, it communicates with Arduino-ext to receive control commands from the hand controller during manual mode, and with the computer to get commands from Base Station during autonomous mode. These commands are then communicated to Arduino-int. The Raspberry Pi also communicates with the IMU via USB.

4.2 Sensors

From the description in figure 2 of the hardware components included in Balrog, the following concludes the setup of Balrog's sensors.

- **LiDAR:** Connects with *Computer* via USB. A new LiDAR is implemented to Balrog this year, see section 4.2.1 for more information. Used for obstacle detection.
- **Wheel sensors:** Connects with *Arduino-int*.
- **IMU:** Connects with *Raspberry Pi* via USB. Includes accelerometers, gyroscopes and magnetometers.
- **Camera:** Connects with *Raspberry Pi* via CSI (Camera Serial Interface). Used to detect AprilTags.

4.2.1 LiDAR

A new LiDAR (Light Detection and Ranging instrument), more specifically a *SLAMTEC RPLIDAR A2M8-R4*, will be installed on Balrog. The measurement performance data from its data sheet, [2], is therefore presented in table 1.

Specification					
Item	Unit	Min	Normal	Max	Comments
Measurement range	Meter	-	0.15-18 meter	-	Based on white objects as reflections.
Angular range	Degree	-	0-360°	-	-
Measurement resolution	Millimeter	-	< 0.5	-	The object is in 1.5 m range.
Angular resolution	Degree	0.45	0.9	1.35	When the scan rate is set as 10 Hz.
Sample duration	Milliseconds	-	0.25	-	-
Measurement frequency	Hz	2000	≥ 4000	8000	-
Scan rate	Hz	5	10	15	Typical value based around 400 samples per round.

Table 1: Specification SLAMTEC RPLIDAR A2M8-R4

4.3 Manual mode

In manual mode the user can drive Balrog using a hand controller that communicates with Arduino-ext over radio. In this case Balrog does not consider obstacles or people in distress.

4.4 Autonomous mode

Balrog's main purpose during the operation is to aid the people in distress located within the test area. In the beginning of the operation, Balrog has a set status 'All not found'.

The positions of the people are received from Sauron as it maps the test area. Balrog stores the positions in a queue, sorted by the time it receives them. When Sauron has concluded that there can be no further people in distress within the area, Sauron instead sends a signal that all people in distress are found. Balrog then changes status to 'All found'. This communication between Sauron and Balrog as well as the queue stacking occurs simultaneously with the other functions.

The rest of the operation will be executed in the way described below and in the flow chart in figure 3. In the beginning of the flowchart, Balrog will be in SLAM-mode, mapping the area.

1. Balrog checks if the queue is empty.
   (a) If the queue is non-empty, the planning program overrides the mapping of the area (SLAM). The planning program takes the first position in the queue and plans a route from Balrog's current position to the one of the person in distress. Balrog follows this route. When Balrog has followed this route and successfully aided the person in distress, the planning program removes the position from the queue and then plans and executes a route to the next person.
   (b) If the queue is empty, Balrog will go to step 2.
2. Balrog checks the 'All found'-status.
   (a) If the status is 'All not found', Balrog enters SLAM-mode and starts mapping the surrounding area. Simultaneously, Balrog will go to step 1.
   (b) If the status is 'All found', Balrog will go to step 3.
3. Balrog plans and executes a route back to its starting position. Lastly Balrog changes the status to 'All not found', leaving it ready for a new mission.

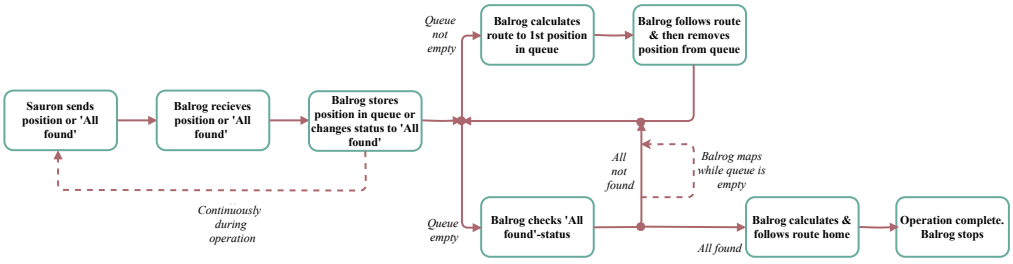

Figure 3: Flowchart of Balrog's operation in general.

4.4.1 Positioning

Balrog has implemented software to detect obstacles and determine their positions within the test area as part of the SLAM algorithm. Included in the SLAM algorithm is also a dynamic update of the estimated position of Balrog in a local coordinate system. As Balrog also is equipped with a GPS (Global Positioning System) that provides a position in global coordinate system. As Sauron also operates in a global coordinate system, it is

desirable to be able to interchange positions given in the global system to Balrog's local one.

There are two alternative solutions to this problem. The first one is to simply fit a very good and dependable GPS to Balrog, so that both robots operate in the same coordinate frame. However, this might be expensive. It might also not work perfectly, meaning that it could still require some adjustment. In that case the other solution might still be needed to be implemented. The other solution is to determine the relation between the local and global coordinates using existing sensors and a touch of sensor fusion.

Transforming a position between the two coordinate systems requires three parameters to be known: the offset in x- and y-coordinate $(\Delta x, \Delta y)$ and the rotation angle, θ between the two coordinate axes. To illustrate the problem, Figure 4 below is provided.

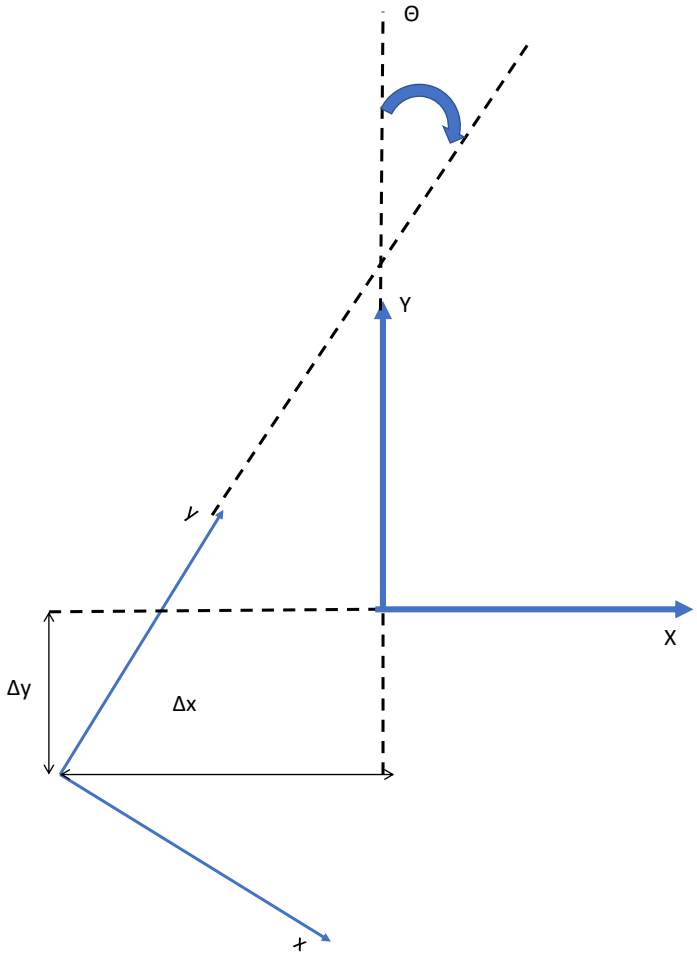

Figure 4: The relation between the local and global coordinate systems.

Translating Figure 4 into an equation gives

$$\begin{bmatrix} x^g \\ y^g \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x^l + \Delta x \\ y^l + \Delta y \end{bmatrix} \tag{1}$$

Where superscript l denotes local coordinates and superscript g denotes global ones. Estimating the unknown parameters $\Delta x, \Delta y$ and θ is done using a separable least squares solution, where it is observed that the problem is linear in Δ, and that a 1D-grid of values for θ in the region of $[0,\ 2\pi]$ will provide values for the parameters that minimize the sum

of the two-norm of the prediction errors:

$$\sum_k \left|\left| \begin{bmatrix} x^g \\ y^g \end{bmatrix}_k - \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x^l + \Delta x \\ y^l + \Delta y \end{bmatrix}_k \right|\right|_2^2 \tag{2}$$

where sub-index k denotes sample instance. This should be done both in an initial stage, as calibration, and continuously throughout the operation, as a greater amount of samples from Balrog's GPS will provide a better estimate of the parameters. It might be desirable to prevent Balrog from acting on any provided target coordinates until the prediction error is sufficiently small. In this case, sufficient is defined as finding parameter values corresponding to a global optimum.

4.4.2 Route planning

When all aspects of the positioning are working, the route planning is just a case of finding the fastest or shortest way to a given point from Balrog's current position.

When planning the route, three main alternative to plan the route will be considered:

1. Alternative 1 - Queue system:
   This is the system described previously, and which is depicted in figure 3. Balrog simply stores the incoming target positions in a queue, a list which implements a 'First Come, First Served' principle. This means that Balrog only considers one route at a time - the one from its current position to the first target position in the queue. When one target is considered to have be aided, it calculates the route to the next one in line. When the line is empty and it is determined that the test area is fully mapped (and that there therefore can be no more people in distress left to aid), Balrog calculates a route back to the starting position. If Balrog at any given moment during operation has an empty queue, but space left to explore, it will revert to mapping the test area.

2. Alternative 2 - Shortest route
   This alternative provides a slightly more optimized route planning system than Alternative 1. When Balrog has more than one person in the queue, Balrog calculates routes from its own position to each of the positions in the queue. Balrog then chooses the shortest route and travel to that person in distress. That is, Balrog only calculates the route between two positions at a time, not a route for the entire queue. The travel time would most likely decrease, but the number of possible routes would each time depend linearly on the number of people in distress. To create a more optimized route planning system like this is a requirement of priority 2, so this will be done only if there is time for it.

3. Alternative 3 - Re-evaluation:
   Every time Balrog obtains a new target position, it will solve a new Travelling Salesman Problem (TSP), which includes every other known target position/node plus the newly obtained one and finds an optimal route that visits all of them. The problem here will be that Balrog might be re-calculating its optimal route too often, and not focus enough on actually getting to a target to aid them. A solution would then be to make a compromise between Plan 2 and 3, and only re-calculate the optimal route after reaching the current target point. This problem formulation could also obtain a further dimension if the targets had different weights; some people might be more highly prioritized than others.

   However, TSP is in NP complexity, meaning that when the number of people in distress increases, the number of possible routes increases exponentially, as do the

worst-case running time. This might cause a problem if the number of people in distress is high. Therefore, this alternative will most likely not be done.

4.4.3 Obstacle avoidance

Depending on whether Balrog's SLAM algorithm has allowed it to map the entire test area or not, it might encounter unknown obstacles on its way to the target location. If Balrog encounters an obstacle along a planned route, it has implemented functionality to re-calculate the route after conclusively determining the size of the obstacle using its on-board sensors and some simple (or more intricate) obstacle-avoidance algorithm.

4.4.4 Route Following

The Balrog platform has a built-in controller that ensures that the robot does not diverge too greatly from its planned trajectory. The controller is split in two separate regulators: one for linear velocity and one for angular velocity, briefly described below.

The controller is a remnant from previous years and is implemented as a ROS node based on Matlab code. The two regulators together provide the control signals for the two tracks, which are sent from the Raspberry Pi to the Arduino.

Linear Velocity
The regulator of the linear velocity is a simple P-controller that uses the euclidean distance between Balrog's current position and the target position as reference signal. To prevent too aggressive and sudden movement for small differences in distance, and to compensate for uncertainty in sensor values, a tolerance value is included in the difference in distance. The P-controller provided good enough performance during previous projects, but should the need arise for faster performance, the controller could be extended with a D-part, acting on the velocity. This could smooth Balrog's driving pattern, preventing overshoots in speed and too sudden braking.

Angular velocity
Regulating the angular velocity of Balrog is done controlling the angle between a straight line to the target coordinate, and Balrog's current angle. A P-controller is used, as it was deemed sufficient during previous projects. Should further precision and faster performance be needed, the controller could be extended with a D-part, acting on the angular velocity with effects similar to the introduction of a D-part in the linear case.

4.4.5 AprilTag Detection and Supply Drop-Off

The camera on Balrog is used to identify AprilTags marking people in distress. This functionality is necessary for Balrog to be able to separate obstacles and people in distress when moving around the test area. When Sauron has provided Balrog with the position of a person in distress and Balrog is following its calculated path to said target, the camera will serve as a double-check to make sure that Balrog moves towards the correct person by validating the identity of the intended target.

A way to do this is to allow Sauron to, together with the position of the person in distress, send some other information that is unique for that specific AprilTag, for example a four bit number combination. When Balrog arrives to the AprilTag it could then check that this combination matches the one that Sauron sent, thus making sure that it is the correct person. If it is not correct, or if Balrog is unable to detect the AprilTag, a few alternative solutions to handle the problem will be considered.

- Create a possibility for Balrog to communicate to Sauron that the aiding was unsuccessful. In this case, Sauron could fly back to that position once it has completed its other task, and send the AprilTag-data to Balrog once more. In this case, Balrog would have to have a status check on each of the positions in the queue, in order to keep track of the number of times a position has been tried. If Balrog has tried a position twice, the operation should be marked as unsuccessful.
- Allow Balrog to drop the person in distress from the queue and send information to Base Station that the aiding was unsuccessful.
- Allow Balrog to shift the position to the end of the queue, and try once again when all other people have been visited. Again, a status check would have to be implemented, as in the first alternative.
- When all other people have been seen to, let Balrog scan the entire area once, to try to find the missing person. Again, a status check would have to be implemented, as in the alternative above.

One could possibly create a solution of a combination of these options. For example, the following flow could be used:

1. Balrog notifies Sauron that the position could not be found, together with the specific position. Balrog changes the position status to 'Visited once' and shift the position to the end of the queue.
2. Sauron flies back to the position and, if possible, scans the AprilTag again and sends the data to Balrog. Balrog visits the position again, once it reaches it in the queue. If not successful, Balrog changes the position status to 'Visited twice' and shifts the position to the end of the queue.
3. When Balrog reaches a position in the queue with status 'Visited twice', Balrog will, if all other people have been seen to, scan the entire area using the minesweeper functionality that has previously been implemented [3].

In the flow presented above, step number 3 could be skipped and Balrog could instead simply notify Base Station and drop the position entirely from the queue.

In addition to this, the LiDAR can be used to make sure that Balrog's position is sufficiently close to aid the person.

5 Quadcopter Sauron

The quadcopter's objective is to scan a certain area for people in distress, determining their positions and transmitting it to Base Station. In order to do that the quadcopter needs some functionality. First it needs functioning hardware, which in this case is provided by a manufacturer. The quadcopter needs sensors to execute its tasks. GPS, IMU, barometer and compass are used for 3D positioning and control of the quadcopter's flying and route following. The camera are used for detection and identification of a person in distress. The problem of transmission of data to Base Station is handled in dept in chapter 6.2.

Sauron's functionality is described by figure 5. The main objective, that of determining the positions of the people in distress, PD, are in red in the center. The functionality needed for that objective are the boxes in blue and sensors are marked with yellow boxes.

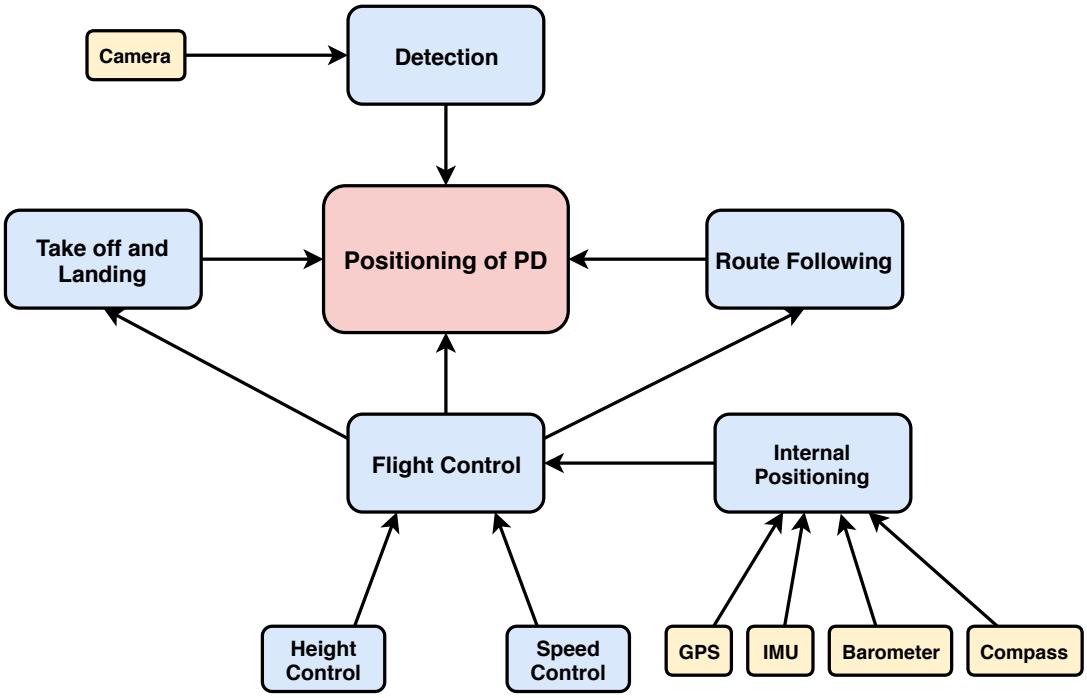

Figure 5: Description on functionality and dependencies for Sauron.

5.1 Hardware

Sauron is a X8+ quadcopter built by 3D Robotics. The onboard computing hardware consists of the following:

- **PixHawk**: A flight controller hardware running the open source autopilot software Ardupilot. This software provides tools for simulation, analysis and data-logging in addition to enable navigation and control of Sauron.
- **Raspberry Pi**: A Raspberry Pi model 3 that is placed onboard Saruon and connected to PixHawk. The Raspberry is communicating with Base Station through Wi-Fi. The software consists of Ubuntu Mate Xenial running ROS as operating system. The Raspberry Pi enables tracking of a known object, using computer vision.
- **Wi-Fi Range Extender**: Increasing the range for Wi-Fi to enable secure data transmit to Base Station.

Off-board hardware consist of:

- **RC hand controller**: A controller for manually controlling Sauron. In automatic flight mode it also have switches to manually override automatic mode and take control, as well as a override landing button, enabling Sauron to land. It is sending over a 2.4 GHz network.
- **Router**: Connects Sauron to base Station through 5 GHz Wi-Fi and a cable, placed between router and Base Station. Should be placed close and with no obstacles placed in between Sauron and router.

5.2 Sensors

The onboard sensors on Sauron is following:

- **Integrated sensors**: Flight control sensors with access through Ardupilot include a GPS, a compass, barometer for measuring altitude and an IMU (Inertial Measurement Unit) which has a magnetometer, an accelerometer and a gyroscope.
- **Integrated camera**: A Raspberry Pi camera module V2, mounted facing downwards, enabling detection of AprilTags placed on obstacles and people in distress.

5.3 Positioning

Sauron has a four dimensional orientation. Its position is determined by coordinates in the plane which are measured by GPS. Sauron's altitude in relation to the ground are measured with the barometers. Finally Sauron have an orientation in the plane. The positions from the GPS are in global coordinates WGS84 (World Geodetic System 1984). The orientation is relative to the north direction and the altitude is relative to the ground from where the quadcopter starts. To be able to manage Sauron in a local frame, a transformation between a global framework and a local one is necessary. First a definition of a local frame is done.

Define the position where Sauron takes off before a mission as *home location*. Further define the coordinate system with home location as origin and y-axis pointing in the face

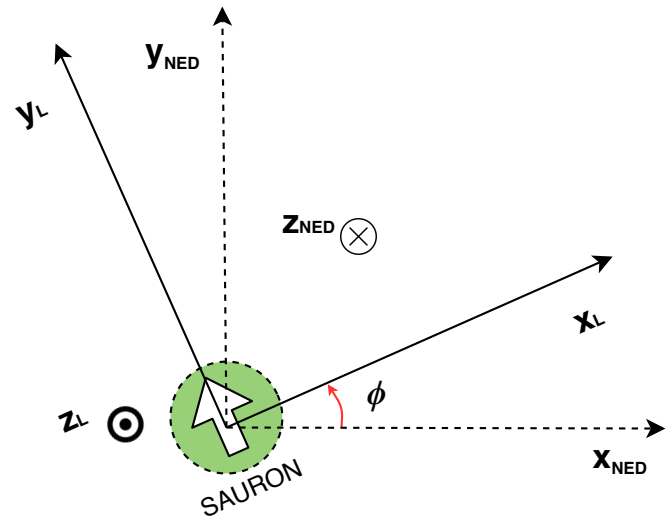

Figure 6: NED-frame coordinates relative to local-frame coordinates.

forward direction of Sauron and x-axis right orthogonal to that axis as the *local frame*-coordinate system.

Fortunately the API (DroneKit) which are used to program the quadcopter have functionality to transform global coordinates to so called NED-frame coordinates. The NED coordinate system have it's y-axis pointing to the *north*,the x-axis pointing to the *east* and z-axis pointing *down* towards earth, yielding a right handed orthogonal coordinate system, hence NED. The origin for the NED-frame coordinate system in the built in function in DroneKit is specified to the position where the quadcopter takes off hence, home location as defined earlier.

That is, to complete the transition between global and local coordinates the problem is to transform from NED to local frame and vice versa. The relation between the two coordinate systems can be seen in figure 6. In the figure, the circle with a cross inside is defined as direction down in the plane and the circle with a dot is defined as direction up from the plane.

The transformation from NED-coordinates to local frame coordinates is given by the rotation defined by Equation 3.

$$\begin{bmatrix} x_L \\ y_L \\ z_L \end{bmatrix} = \begin{bmatrix} \cos\phi & \sin\phi & 0 \\ -\sin\phi & \cos\phi & 0 \\ 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} x_{NED} \\ y_{NED} \\ z_{NED} \end{bmatrix} \tag{3}$$

To go the other way, that is from local frame to NED-coordinates yields the rotation given by Equation 4.

$$\begin{bmatrix} x_{NED} \\ y_{NED} \\ z_{NED} \end{bmatrix} = \begin{bmatrix} \cos\phi & -\sin\phi & 0 \\ \sin\phi & \cos\phi & 0 \\ 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} x_L \\ y_L \\ z_L \end{bmatrix} \tag{4}$$

5.4 Overall control and route following

The route following of Sauron should take measurements from the GPS and comparing it to a pre-specified trajectory. The trajectory are specified in the geometrical plane. The control error (i.e. the difference between the trajectory and the position at a given point) are used to feedback the motors and regulate the position such that it approaches the trajectory.

Something needs to be said about the level of which the control are going to be executed. The quadcopter comes with a lot of functionality already existing. For example there is already functionality to go from a certain position to another. The task isn't therefore to control the motors directly but rather to use the built in flight control to ensure that the quadcopter keeps to the trajectory. A high level controlling.

5.4.1 Refererence trajectory

In order for Sauron to follow a specific route, a reference trajectory needs to be defined. In addition, how the trajectory should be updated must also be defined.

The easiest way of making a trajectory would be to define reference points in the plane only where new directions should be carried out. A new reference point is set when the positional measurements indicates that it is close enough then a new reference point is set.

An illustration of this approach can be seen in figure 7. Define r_i as the i:th point in a series of points which are pre-specified positions in the local frame coordinate system. To go to the next point a certain function is called. When r_1 is sufficiently close (meaning have surpassed a lower border of distance to the point) the reference is updated to r_2 and so on. Observe that this is only the approach of how and when the reference should be updated, not a design for the trajectory, which will be possible to define by the user upon start.

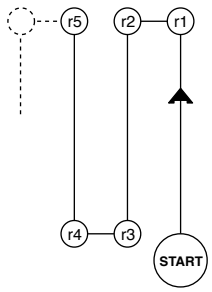

Figure 7: Trajectory example.

5.4.2 Control problem

The problem needs to be formulated further. In order to be able to easily refer to different variables, the variable names showed in the general figure 9 are going to be used throughout this sub chapter. In figure 9 u stands for input (control variable), z the controlled variable, n is measurement noise and y are the measured signals.

Controlled variables:
The variable that are going to be controlled are Saurons's orthogonal distance to the trajectory line, which is defined as the line between the point Sauron has left and the point Sauron is going to visit. In Figure 8 the control idea are shown. Let d be the orthogonal vector from Sauron to the line between point r_1 and r_2 which is part of the trajectory. In the notion of figure 9, let's call the control variable z_d. With basic algebra the distance d is calculated as:

$$d = \frac{|\ \vec{a} \times \vec{u}\ |}{|\ \vec{a}\ |} \tag{5}$$

where: $\vec{a} = r_1 - r_0$ is the direction vector from r_0 to r_1 and $\vec{u} = s - r_0$, where s is the current position of Sauron. These points are all known.

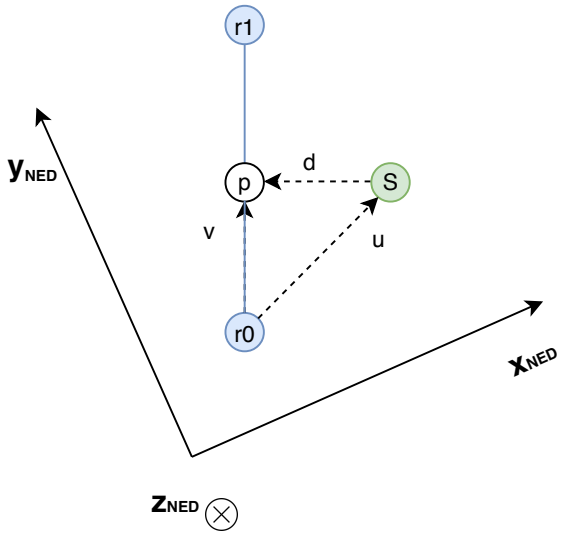

Figure 8: Reference points in reference trajectory.

Input variables:

To control the distance to the trajectory there is a neat way of controlling the quadcopters velocity. It is possible to set a velocity vector for the quadcopter. How this velocity vector should look like could be done in many ways. The approach here is to add to the velocities that already are carried out by the function that takes Sauron between two points. Hence the best way is to add velocity in the direction of $\vec{d}$ as in Figure 8. The direction of $\vec{d}$ is calculated as:

The vector from r_0 to p is $\vec{v} = proj_{\vec{a}}(\vec{u}) = \frac{\vec{u} \cdot \vec{a}}{\vec{a} \cdot \vec{a}} \vec{a}$
The point p becomes: $p = \vec{Op} = \vec{Or_0} + \vec{v}$
And now we can calculate $\vec{d}$ as: $\vec{d} = \vec{Op} - \vec{OS}$

The normed vector $\vec{d}$ becomes the control variable, we denote it u_d.

In summation, we have a SISO (Single Input Single Output) system with one input variable to control one output variable.

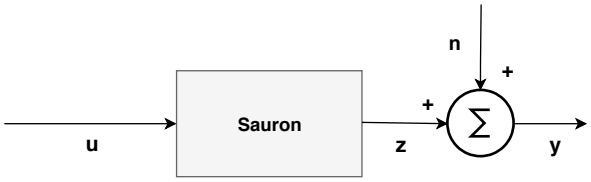

Figure 9: Control problem for route following Sauron.

Control Design:
A PID controller with tuned parameters from simulated step response experiment is a good start.

5.5 Flight Control

The sensors in chapter 5.2 are to be used for Sauron to fly autonomously. Sauron's autonomous flight can - beside the route following - be divided in three important problems: Control of Sauron's height above the ground, control of speed and control of landing and take off.

Some of this functionality regarding flight control are already provided by the manufacturer. The objective here is to integrate this functionality in an autonomous way. In the following sub chapters there are given a brief description of the three different important problems and the approach of integrating it in the autonomous functionality.

5.5.1 Height control

The goal is to get the quadcopter to take off and approach a pre-specified height. This can be done by the same approach as in the route following chapter.

5.5.2 Speed control

In order to ensure that the speed limit isn't surpassed there is a need to control the speed of Sauron. This can be done by limit the velocities that is carried out to Sauron.

5.5.3 Take off and Landing

The take off procedure is only a matter of height and speed control which are already dealt with. The landing procedure are more delicate. There are built in landing routines for the quadcopter which could be used for landing. This will be used in first hand. Otherwise the approach is to land in two steps. First being to approach a certain level of height with normal height control. The second step is a more tentative, cautious procedure. In this step the height control are slow to ensure that the quadcopter doesn't smash into the

ground. The confirmation of landing could be a decision from both barometer, gyroscope and camera.

5.6 Manual Mode

Autonomous functionality set aside, Sauron should be able to be maneuvered manually. This is done by the RC-controller.

5.7 Detection and identification

Detection and identification of people in distress represented by AprilTags is done with the camera installed on Sauron. Sauron searches the test area until an two or more detections are made then Sauron will identify the detected object to check if it's a person in distress or a false alarm.

5.7.1 Detection

Detection of people in distress is done by taking pictures and then processing them to see if any AprilTags can be detected. Sauron will cover the whole test area with pictures to find every person in distress. There are two different strategies for covering the test area.

One strategy is to fly 2-3 m in the air and take pictures while moving and then when two or more detections are made Sauron will move over the detected target to identify it. After identification Sauron will continue on its path and look for more people in distress until the whole test area is covered. This strategy requires a frame rate high enough to take at least 2 pictures of every AprilTag while moving. To get a high enough frame rate a lower resolution will be used.

The other strategy is to fly 3-10 m up in the air to view a larger area and then look for people in distress. After flying up Sauron will stay still and take 3-10 pictures of an area and then move to another area until the whole test area is covered. Since Sauron will be staying still there is no need for a high frame rate and high resolution pictures can be taken and processed. Since the images are of a larger area there can be more than one person in each image and if two or more images in a row detects the same person, the location of the person will be marked for identification. To make sure it is the same person some assumptions are made. The assumptions are that all people in distress are standing still, that Sauron will not move more than 0.2 m in any direction, that Sauron will not rotate more than 5 degrees in any direction and that the people in distress are at least 2 meters apart from each other. Based on these assumptions Sauron will calculate the position of all the detections, and if the calculated position moves less than 1 m between frames it is considered the same person. The detection will be done with an AprilTag ROS package which can be used to get an estimated distance and angle of the AprilTag from the camera. Then with the position and height of Sauron which is given by the GPS and barometer, the location of the distressed people can be estimated. After estimating and marking the locations for identification Sauron will move to the location to identify it.

5.7.2 Identification

Identification is implemented to avoid false alarms from the detection. Identification is done by having Sauron hover over the spot two consecutive detection were made while taking pictures to see if more detections can be made. If enough detections are made the

target is identified and the location is sent to Balrog. If there aren't enough detections it will be treated as a false alarm.

6 Communication

All the communication between the three main systems is done through ROS (Robotic Operation System) using Wi-Fi connection. A overview of the communication between the system can be seen in Figure 10.

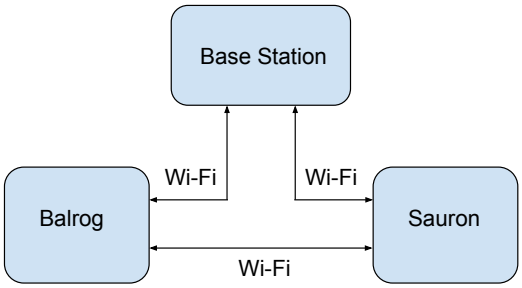

Figure 10: Overview of the communication between the systems.

6.1 Communication between Balrog and Base Station

The communication between Balrog and Base Station is done through Wi-Fi. The data sent from Balrog to Base Station is a live feed from the camera, LiDAR data, position data and the estimated map. The data sent from Base Station to Balrog is instruction data such as start, stop, and switch between autonomous and manual mode. This is mainly the communication between the user and Balrog. Balrog transmit its current position to base station with an interval between transmissions of 1 second.

6.2 Communication between Sauron and Base Station

The communication between Sauron and Base Station is done through Wi-Fi, with a 5 GHz network, using the onboard Raspberry Pi. In the event that the network need to be changed communication should not be done over 2.4 GHz. This since sending at the same as the hand controller could cause delays and lags. The data sent from Sauron to Base Station is a video feed, location data and sensor data. The Base Station sends back instruction data such as start, stop, and switch between autonomous and manual mode. Sauron transmit its current position to base station with an interval between transmissions of 1 second.

6.3 Communication between Balrog and Sauron

Communication between Balrog and Sauron is done through ROS with Wi-Fi. The data sent between the systems are position data of located people in distress. Sauron will transmit its current position to Balrog with an interval between transmissions of 1 second. After finding all people in distress on the map Sauron sends a confirmation to Balrog.

7 Graphical User Interface

In the GUI, see figure 11, the user will be able to start and stop a mission, change parameters and see the progress of the robots. The main idea with the GUI is that the user shall be able to execute a simple mission without any trouble.

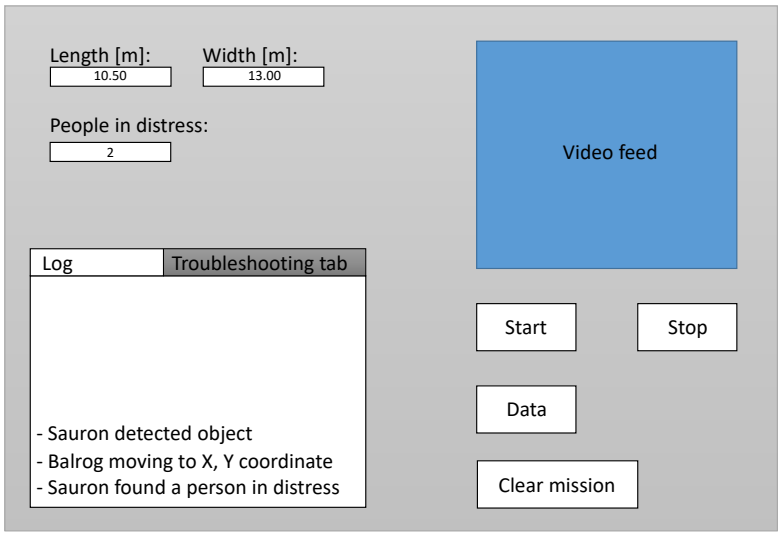

Figure 11: Sketch of the GUI.

First the user have the option to change the dimensions of the search field, based from Base Station, and other parameters. The user can't change any parameter during a running mission. Afterwards the user can start the mission by clicking on the button "Start". If the user want to abort the mission the user can click on the button "Stop". This will trigger the robots to stop the mission in a way were Sauron lands and Balrog stops his engine. Once the mission starts the user will be able to see the progress in a window in the GUI. There is also a troubleshooting tab in the interface which the user can use for troubleshooting. While the mission is running the user will be able to see a log of messages displayed in the troubleshooting tab, such as "Sauron detected object", "Balrog moving to X,Y coordinate" and "Sauron found a person in distress".

If Sauron finds a person in distress a parameter "People in distress" will add one to it's counter which could be seen in the GUI. Simultaneously a video feed from Balrog will be shown in the GUI. The video feed is transmitted from a Raspberry Pi which is attached on Balrog. After a mission has been finished the data could be shown in the GUI by clicking on the button "Data".

The user can then click "Clear mission" and start another mission afterwards. Once the "Clear mission" has been used, the same interface will be presented as before, where the user can change dimensions and parameters.

REFERENCES

[1] T. Svensson, C. Krysander och Studentlitteratur 2011, "Lips", Studentlitteratur AB Lund, [Internet]: `www.studentlitteratur.eng`

[2] Slamtec, "RPLIDAR A2: Low Cost 360 Degree Laser Range Scanner", 2017, [Internet]: `http://www.bournemouth.ac.uk/library/using/guide_to_citing_internet_sourc.html`

[3] Smaugs, "Technical Documentation, Minesweeper", 2017, [Internet]: `http://www.isy.liu.se/edu/projekt/tsrt10/2017/minesweeper/dok/CDIO_Tech_doc0.4.pdf`

[4] Rescue Rangers, "Requirement Specification, Autonomous Rescue System with UAV and Tracked Vehicle", 2018.