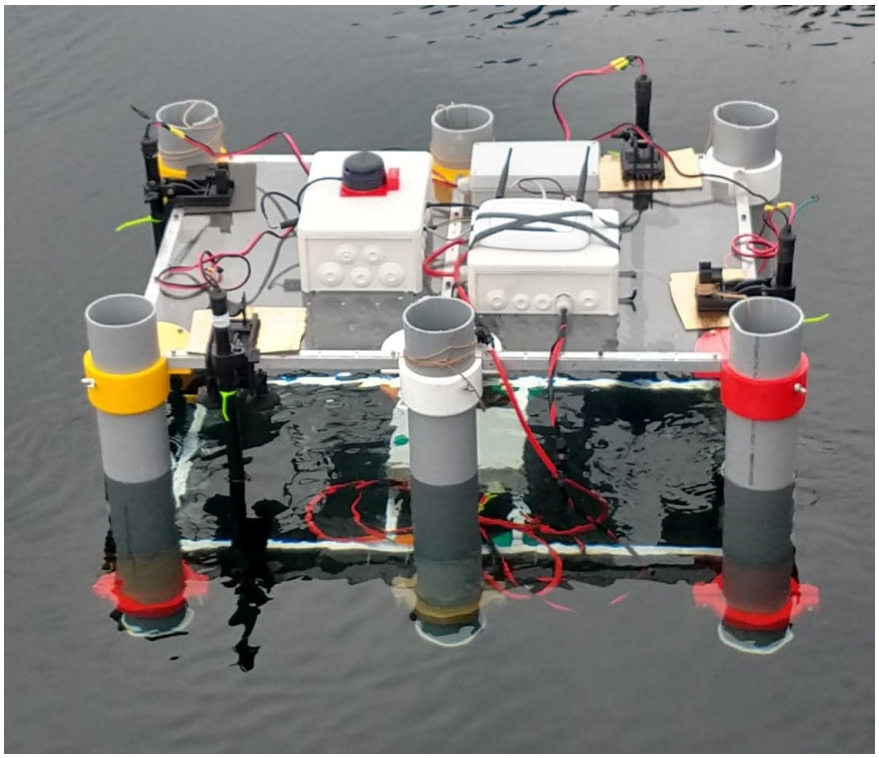
Bachelor's thesis

Håkon Bjerkgaard Waldum
Ruben Ole Berg Natvik
Ruben Svedal Jørundland
Vebjørn Rimstad Wille

Autonomous Inshore Navigation with Lidar

May 2020

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of ICT and Natural Sciences



NTNU
Norwegian University of
Science and Technology

Bachelor's thesis

2020

NTNU
Norwegian University of
Science and Technology

Håkon Bjerkgaard Waldum
Ruben Ole Berg Natvik
Ruben Svedal Jørundland
Vebjørn Rimstad Wille

Autonomous Inshore Navigation with Lidar

Bachelor's thesis
May 2020

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of ICT and Natural Sciences

NTNU
Norwegian University of
Science and Technology

NTNU

Kunnskap for en bedre verden

Autonomous Inshore Navigation with Lidar

Håkon Bjerkgaard Waldum, Ruben Natvik, Ruben Svedal Jørundland, Vebjørn Rimstad Wille

Mai 2020

PROJECT / BACHELOR THESIS

Department of ICT and Natural Sciences

Norwegian University of Science and Technology

Supervisor 1: Ottar L. Osen

Supervisor 2: Robin T. Bye

Preface

This bachelor thesis is written by four students from Automatiseringsteknikk at NTNU Ålesund. The group consists of members with a varying degree of practical experience in the field, as well as varying fields of interest.

The goal of this project is to see what practical application mapping and localizing with a lidar at sea can have and how realistic it is to get it working without any huge problems.

This type of technology is quite interesting because of the doors it opens when looking towards creating more autonomous vehicles (both at sea as well as land). There are so many aspects to take into consideration when mapping and localizing, that there are huge potential pitfalls when creating such a system.

All of this combined made the thesis quite intriguing for all of the project members, and made every day working with it creatively interesting.

Acknowledgement

We would like to thank

- Our supervisors for all the help and guidance through the project
- Family and friends for supporting us through the semester
- Andreas Nordal, for all the help with hydrostatic calculations
- Håvard Lien, for all help with hydrostatic calculations and how to improve the boat-aspect of the prototype
- Obs! Bygg Ålesund for sponsoring a lot of the parts for the construction of the prototype
- Anders Sætersmoen for ordering and supplying the parts needed for the project

Summary

This report concerns the development of a prototype that should autonomously navigate and dock using a Lidar for vision. The purpose of this project is to investigate how to implement such a system, and how viable it can be for autonomous navigation and docking of vessels. A prototype based on a semi-submersible design was developed and equipped with a Lidar and other sensors and equipment to achieve this. The prototype has four thrusters enabling movement in all horizontal directions. The results from testing this prototype show that the system is capable of navigating through smaller areas, where the Lidar has good enough range to find reference surfaces to localize itself. It does however struggle with obstacle avoidance due to a lack of processing power. Another issue is that the Lidar used was not powerful enough to be used outdoors in larger areas due to a lack of range and therefore reference surfaces for localization. Even still the system shows great promise and worked well in a smaller area with enough references for the used Lidar's range.

Contents

Terminology

Steady State Response The output of a system in equilibrium

Transient Response The output of a system not in equilibrium

Covariance A measurement of how different variables vary depending on variation from eachother

Data Packet A unit of data sent across a network

Node A single module unit used by ROS

IPxx Ingress Protection Code, xx is replaced by a number which says something about a components degree of protection from intrusion

Open Loop A controller with no feedback path

Closed Loop A controller with a feedback path

JavaScript A programming language used in web pages.

jQuery A Document Object Model traversal library for JavaScript

Plotly A plotting library for JavaScript

HTML Hypertext Markup Language, a language used to structure web pages

CSS Cascading Style Sheets, a styling language used to style web pages

Python Programming language

Flask a light weight web framework used by Python

RVIZ Visualization software for ROS

Gnd Ground in electronical circuits

DOF Degrees of Freedom, number of configurations for an object

Notation

K_p Proportional term of a PID controller

K_i Integral term of a PID controller

K_d Derivative term of a PID controller

g Earth gravity constant, $9.81 m/s^2$

ρ Density of a given object

ψ Yaw angle

Abbreviations

IEEE Institute of Electronical and Electronic Engineers

MIMO Multiple Input, Multiple Output

PID Proportional, Integral, Derivative

LiDaR Light Detection and Ranging

GPS Global Positioning System

AMCL Adaptive Monte Carlo Localization

SLAM Simultaneous Localization and Mapping

UDP User Datagram Protocol

TCP Transmission Control Protocol

I2C Inter-Integrated Circuit

UART Universal Asynchronous Receiver / Transmitter

IP Internet Protocol

ROS Robot Operating System

GA Genetic Algorithm

IMU Intertial Measurement Unit

CAD Computer-Aided Design

IDE Integrated Development Environment

GUI Graphical User Interface

PWM Pulse-Width Modulation

SIT Studentsamskipnaden i Gjøvik, Ålesund og Trondheim

List of Figures

List of Tables

Chapter 1

Introduction

This report will step by step go through all the stages of development of the prototype for this project. All principles that have been considered will be drawn out, as well as what has been chosen for the final solution and the theory behind this.

1.1 Background

In todays society there is an increased focus on automation and sustainability when talking about our everyday as well as the climate. With a bigger focus on getting people to go over to using public transportation but it being a net gain for all parts involved.

On sunnmøre (South-western part of Møre og Romsdal) the need for transportation by ferry is huge because of all the fjords in the area. If one looks at Statens Vegvesens Ferjedatabank [60] one can see that Fjord1 (the biggest actor in what's called Region Vest [30]) had 31 554 trips in 2019. On these trips there were a total of 401 716 persons aboard on all available trips inside Møre og Romsdal.

To be able to increase the availability of these connections it has to be seen in the aspect of staffing as well as cost in vs. cost out. By making use of autonomous technology one can reduce the cost by decreasing the need for staff aboard, and therefore more easily expanding the connections and increasing the rate of departures as well as extending the operating hours.

When one looks at autonomous ferries, docking is one of the big difficulties when trying to create a safe and sound system. One of the big challenges with a system like this is being able to ensure that the ship does not crash into the dock when arriving.

Earlier research into this field made in cooperation with what was earlier called Rolls Royce Marine (now Kongsberg Maritime), came to the conclusion that using Lidar for such a system had very promising results because of its precision and ability to dynamically map its surrounding areas [53].

Kongsberg Maritime in cooperation with NTNU wants to test a system for autonomous docking that can be scaled to seaborne vessels of any type. This is thought as an alternative system for a bigger more robust system, and will therefore not include any form of GPS to help the vessel find itself in a map or coordinate-system. A system for localizing and mapping the surroundings is therefore wanted.

This project will therefore aim for creating a omni-directional seaborne prototype that will be able to map and localize the environment using a Lidar. The focus will lie in how robust the mapping and localizing will be using a Lidar as well as how impacted it will be by the motion of the ocean.

1.2 Problem formulation

The basic problem for this project is to be able to create some form of prototype that should be able to successfully localize itself in an environment, and from that be able to navigate and dock successfully. This should be done by creating an independent system that functions by using a Lidar. The prototype will simulate a bigger ship by being designed so as to not be impacted by rough seas.

In this project the method of docking used by cargo ships will be the one implemented for

the prototype. This is because of the ease of simulating this on a low scale compared to simulating ferries.

Problems to be addressed

- Design a small-scale boat
- Implement a system for mapping site
- Implement a system for self-localization in a map
- Implement a system for navigation and docking

1.3 Limitations

A limitation set for this project is that it is assumed that the ship is supplied with a map that the vessel can use for localization and navigation from an external source. This map will in our case be generated by the vessel prior to starting system tests.

Because of the limited range of a lidar, this system will focus on navigation inshore where there are several reference-points for the lidar to see.

1.4 Requirements

The requirements for the project is as follows;

- Create a functioning prototype
- Map and Localize the Prototype without using GPS
- Autonomously navigate mapped area
- Autonomously Dock
- Avoid Unmapped Obstacles

1.5 Structure of the Report

The rest of the report is structured as follows.

Chapter 2 - Theoretical Basis: Chapter 2 gives an introduction to the theoretical basis for the prototype and all of its included systems and functions.

Chapter 3 - Preliminary Test Results Chapter 3 details the preliminary test results of early concepts for prototypes and what was learned from these initial tests. It also details early tests of systems without the prototype.

Chapter 4 - Method: Chapter 4 includes a description of the methodology that is used during the project and the software created, as well as all factors that need to be considered when trying to build a seaborne vessel. It also describes the organization of the group and the how the work during the project was to be done.

Chapter 5 - Materials: Chapter 5 is a review of all materials used to produce the prototype and its systems.

Chapter 6 - Testing: Chapter 6 describes the tests that are desired for the project as a whole, and what kind of environment these tests are run in as well as what the goal of the tests are.

Chapter 7 - Result Chapter 7 presents the results of the project. It shows the final prototype and its electrical system, as well as the solution for the software to run the program for the autonomous navigation and docking. The result of all tests are also presented in this chapter.

Chapter 8 - Discussion Chapter 8 includes a critical discussion of the results found in chapter 7 and an evaluation of the prototype compared to a final product. It also discusses the experiences made for the group through the last months.

Chapter 9 - Conclusion Chapter 9 draws a final conclusion for the project as a whole.

Chapter 2

Theoretical Basis

This chapter presents all necessary theory needed to understand the project as a whole. The structure of it is built upon a natural progression through the topics that are used through the project.

2.1 Docking

During a ships docking sequence, there are several factors to consider before actual docking can happen. One important aspect is that these big ships are large and very slow and docking might take several hours.

If one takes a look at the big cargo ships and cruise ships where one often boards or disembarks on its side, the general steps of the docking sequence are as follows:

- Get in the general area of the dock with the boats side lined up with the dock
- Use thrusters to go sideways into dock
- Finalize docking

Ferries use a different method to dock, because they mostly dock with the bow or stern into the dock, to let cars board or disembark.

2.2 Autonomous Robots

The definition of what a robot is is a widely discussed thing in the academic community. But a good definition is a machine that can perform complex tasks and movements, and is programmable [18]. In practice this means a machine that can perform complex tasks on its own with minimal input from humans.

An autonomous robot is therefore a robot that can do all of this, and simultaneously update and orient itself without any form of input from humans.

2.3 Euler angles

Euler angles describes the orientation around the three axis of a body, XYZ. These angles are often used for naval navigation and are named roll, pitch and yaw as shown in figure 2.1.

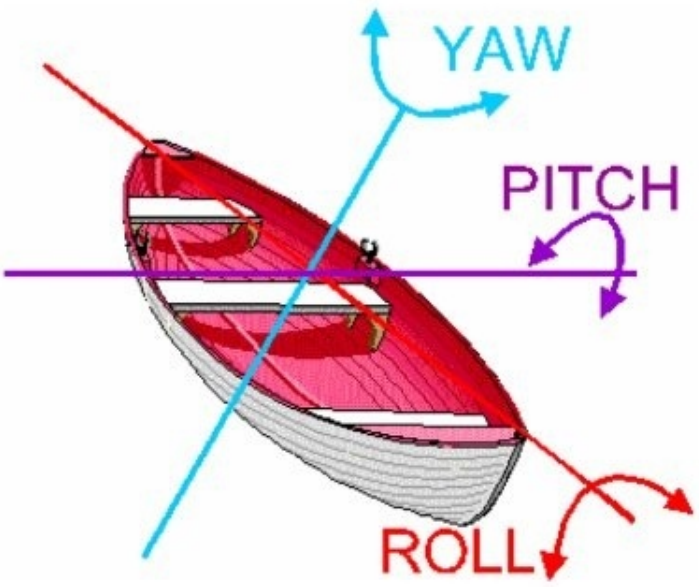


Figure 2.1: Euler Angles illustrated [49]

These angles are often used because of their human readability but suffer from what is called "gimbal lock" which is when the pitch angle approaches +/- 90°, it is prevented from measuring orientation [37].

2.4 Quaternions

Quaternions are an alternate method of measuring orientation in 3D-space. It is often used because quaternions are not affected by "gimbal lock" as described in section 2.3. It is not as intuitive as euler angles, as it consists of 3 complex elements and one real one in a 4-element vector. [37]

2.5 Reference Frames

In robotics reference frames are used to describe the position and orientation of different bodies in relation to each other. Each body has its own coordinate system, and other bodies can therefore be described in this body's coordinate system. [43]

2.6 The Theory of Measurement Uncertainty

When it comes to measurements in repeated values, there will likely be some properties that will affect the measure for each value, these properties can for example be the accuracy of the instrument. This will then create an uncertainty to the total result [36]. Because of this uncertainty, it is important to evaluate the result that is compiled.

The total measurement result can be represented like this:

$$\text{Measurement result} = \text{Measurement average} \pm \text{Standard Error of Mean(SE)} \tag{2.1}$$

where the measurement average is given by:

$$\bar{x} = \frac{\sum_{i=1}^{N} x_i}{N} \tag{2.2}$$

Where:

x_i - Measured value with index i

N - Total number of samples

And the formula for standard error of mean is:

$$\delta \bar{x} = \frac{\delta x}{\sqrt{N}} \tag{2.3}$$

Where:

δx - Standard deviation

N - Total number of samples

finally the formula for standard deviation:

$$\delta x = \sqrt{\frac{\sum_{i=1}^{N}(x_i - \bar{x})^2}{N}} \tag{2.4}$$

Where:

x_i - Measured value with index i

$\bar{x}$ - The average of the population

N - Total number of samples

By representing the measurement result in this way it showcases how trustworthy the result is based on how large the standard error would be.

"Properly reporting an experimental result along with its uncertainty allows other people to make judgments about the quality of the experiment, and it facilitates meaningful comparisons with other similar values or a theoretical prediction." - The University of North Carolina [36].

2.7 Hydrostatics

2.7.1 Archimedes' Principle

Archimedes' principle states that "every floating body displaces its own weight of the liquid in which it floats" [15] and experiences a buoyant force equal to the weight of the displaced fluid. This principle can be expressed as:

$$F_b = \rho \cdot g \cdot V \tag{2.5}$$

Where:

F_b - The buoyant force applied to the object

ρ - The density of the fluid

V - The volume of the displaced fluid

g - The acceleration due to gravity.

2.7.2 Metacentric Height

A ship's metacentre is located at the intersection of vertical lines through the centres of buoyancy in the initial and slightly inclined positions. [61] Metacentric height is the distance between this metacentre and the ship's centre of mass. The position of the metacentre can be determined by the following equation:

$$BM = \frac{I}{V} \tag{2.6}$$

Where:

BM - The distance between the centre of buoyancy and the metacentre

I - The second moment of area at the waterplane

V - the total displaced volume.

2.7.3 Drag

Drag is a resistant force that acts on a body moving through a certain fluid [58]. Drag is dependent on what speed the body moves through this fluid, the area of the body that presents to the fluid as well as the smoothness of the body. It is calculated as such:

$$F_D = \frac{1}{2} C_D \cdot \rho \cdot A \cdot v^2 \tag{2.7}$$

Where:

F_D - Drag Force

C_D - Drag Coefficient

ρ - Density of fluid

A - Area of object facing the fluid

v - speed of the object

2.8 Propulsion

There are several ways to get propulsion on a seaborne vessel. The most efficient way for most types of boats is a form of motor that runs under water, instead of a motor running over the water.

Under water there are primarily two ways of producing propulsion;

- Propeller
- Jet

These two technologies are used in a wide array of different motors for boats. One of the simplest forms of motors used to navigate small boats through waters is a trolling motor.

A trolling motor is often defined as a self-contained electric motor for a boat. The trolling motor includes a motor, propeller as well as controls.

2.9 State Space Modeling

Modeling is the concept of mathematically representing a system. This mathematical representation makes it possible to inspect the behavior of the system when reacted on by a set of inputs [47]. State Space Modeling is one of the possible methods of modeling representation. It is based on a set of linear differential equation in matrix form and is generally represented like:

$$\dot{x} = A \cdot x + B \cdot u \tag{2.8}$$

$$y = C \cdot x + D \cdot u \tag{2.9}$$

where:

x - State Vector

A - State Transition Matrix

B - Input Matrix

C - Output Matrix

D - Feedforward Matrix

u - Input Vector

The advantage of such a representation, is that it is easy to work with MIMO (Multiple-Input Multiple-Output) systems, compared to transfer functions [47]. Below, in figure 2.2, is a block representation of a state space model showcased without the Feedforward matrix.

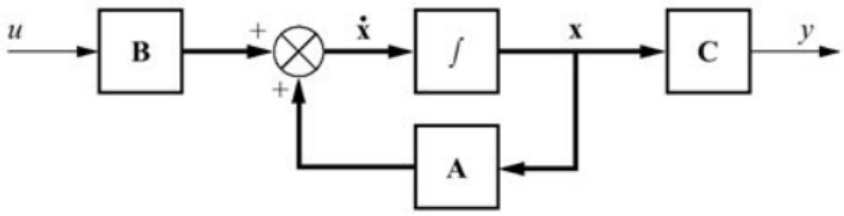


Figure 2.2: Block diagram representation of a state space model, without Feedforward matrix [40]

2.10 Vessel Dynamics

This section covers the theory behind modeling vessel dynamics. This chapter is shortened to the essentials, and a more thorough explanation can be found at [45]

2.10.1 Frames

In order to define the dynamics of a ship's motion it is common to utilize two separate reference / coordinate frames, often called the "Earth-fixed frame" and the "Body-fixed frame". Figure 2.3 illustrates the relationship between these frames.

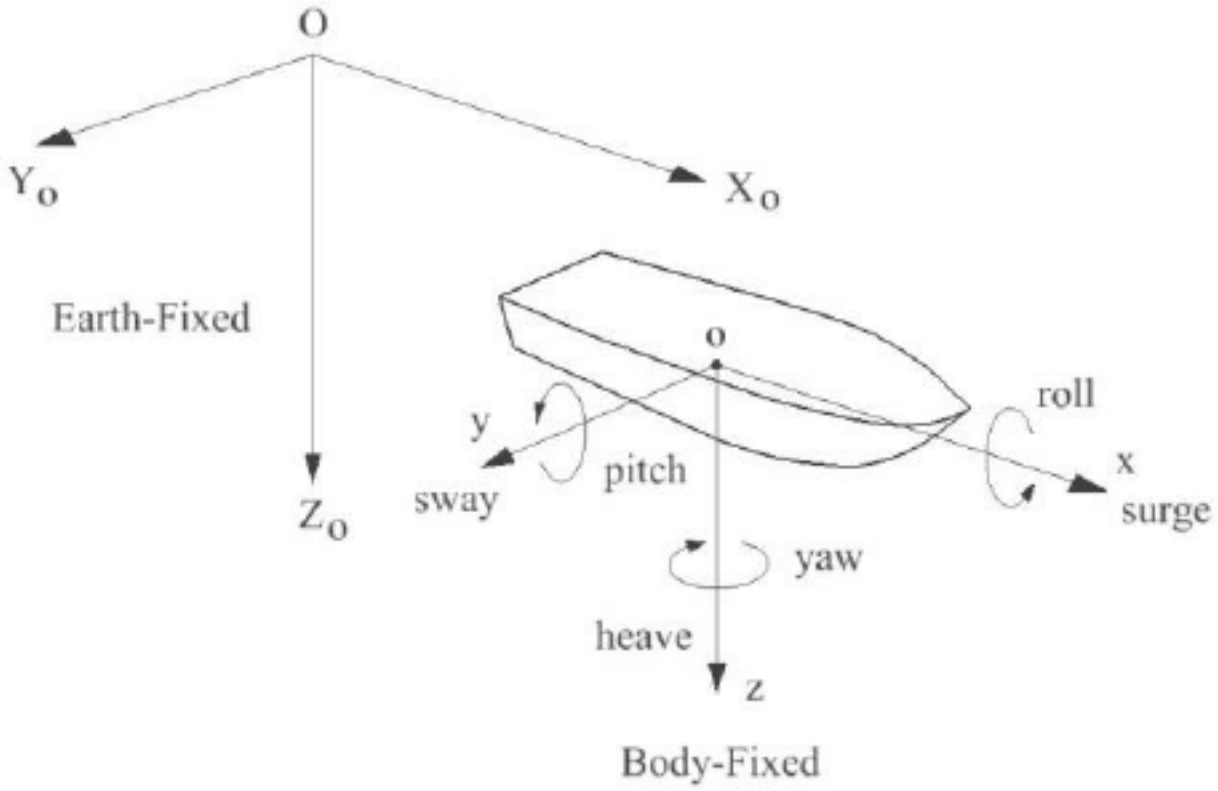


Figure 2.3: Vessel Reference Frames [3]

The earth fixed frame is a local geographical coordinate system and the body fixed frame is attached to the vessel. The position of the vessel is the transformation (translation and orientation) between the earth fixed frame and the body fixed frame and can be expressed as:

$$\eta = [n, e, \psi]^T \tag{2.10}$$

Where:

n, e - the north-east position of the vessel

ψ - the yaw / heading angle relative to north [45]

2.10.2 Modeling Dynamics

For vessel-positioning and heading control it is common to constrain the modeled motions to north, east and yaw. A vessel velocity vector can then be defined as

$$\nu = [u, v, r]^T \tag{2.11}$$

Where:

u - Surge Velocity

v - Sway Velocity

r - Yaw Rate

The system's input vector can be defined as

$$\tau = [X, Y, N]^T \tag{2.12}$$

Where:

X - Force along vessel X-axis

Y - Force along vessel Y-axis

N - Torque around the Z-axis (Yaw Moment)

The various variables and symbols are listed in table 2.1

Symbol	Description	Frame	Unit
n	North position	Earth	m
e	East position	Earth	m
ψ	Heading / yaw angle	Earth	rad
u	Surge speed	Body	m/s
v	Sway speed	Body	m/s
r	Rotation speed	Body	rad/s
X	Force along vessel X-axis	Body	N
Y	Force along vessel Y-axis	Body	N
N	Yaw moment	Body	Nm
η	Position vector		
ν	Velocity vector		
τ	Force vector		

Table 2.1: Description of model variables

A model for the vessel's motion dynamics can then be expressed as [45]:

$$\dot{\eta} = R(\psi)\nu \tag{2.13}$$

$$(M_{RB} + M_A)\dot{\nu} + C_{RB}(\nu)\nu + d(V_{rc}, \gamma_c) = \tau_{control} + \tau_{wind} + \tau_{waves} \tag{2.14}$$

Where:

M_{RB} - Representation of the rigid body mass

C_{RB} - Representation of the Coriolis-centripal effect

And they are defined as

$$M_{RB} = \begin{bmatrix} m & 0 & 0 \\ 0 & m & m \cdot x_g \\ 0 & m \cdot x_g & I_z \end{bmatrix} \tag{2.15}$$

$$C_{RB} = \begin{bmatrix} 0 & 0 & -(x_g r + v) \\ 0 & 0 & mu \\ m(x_g r + v) & -mu & 0 \end{bmatrix} \tag{2.16}$$

When operating velocities are small C_{RB} can be ignored for control design. [45]

When a vessel moves, the water around it experiences a change in momentum. This causes changes in pressure on the vessel's hull and will affect the dynamics of the system [45]. This effect is represented by the matrix M_A which is defined as

$$M_A = \begin{bmatrix} -X_{\dot{u}} & 0 & 0 \\ 0 & -Y_{\dot{v}} & -Y_{\dot{r}} \\ 0 & -Y_{\dot{r}} & -N_{\dot{r}} \end{bmatrix} \tag{2.17}$$

where the matrix coefficients depend on hull shape.

The term $d(V_{rc}, \gamma_c)$ represents the current and damping forces experienced by the vessel. Unless the vessel under consideration is extensively analyzed and tested on small scale models the current coefficients of this term are difficult to estimate with accuracy [45]. It is therefore common to simplify this term to a linear damping term and a bias term.

$$d(V_{rc}, \gamma_c) \approx D\nu - R^T(\psi)b \tag{2.18}$$

where

$$D = D^T = \begin{bmatrix} D_{11} & 0 & 0 \\ 0 & D_{22} & D_{23} \\ 0 & D_{32} & D_{33} \end{bmatrix} \tag{2.19}$$

$$b = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} \tag{2.20}$$

The right hand side of equation 2.14 describes the system inputs. The terms are separated into three force vectors (τ). $\tau_{control}$ is the term for forces applied to the vessel by the actuators, τ_{wind} is the term for forces applied to the vessel by wind, and τ_{waves} is the term for forces applied to the vessel by waves.

2.10.3 Simplified model

Taking the above simplifications in mind, where C_{RB} is ignored due to slow movement, as well as equation 2.18, it is possible to create a simplified model of the vessel.

In order to linearize the model it is necessary to define vessel parallel coordinates, η_p, which is defined as

$$\eta_p = R^T(\psi)\eta \tag{2.21}$$

Under the assumption that the yaw rate is low, and therefore $\dot{R}(\psi) \approx 0$, the system can then be expressed as

$$\dot{\eta}_p = \nu \tag{2.22}$$

$$M\dot{\nu} + D\nu = b_p + \tau_{control} + \tau_{wind} + \tau_{waves} \tag{2.23}$$

where $M = M_{RB} + M_A$ [45]

2.11 PID Regulator

A PID regulator is a regulator system that acts on the difference between a reference signal and a measured signal. This error signal passes through 3 parallel components, namely the Proportional, Integral and Derivative component, as shown in figure 2.4.

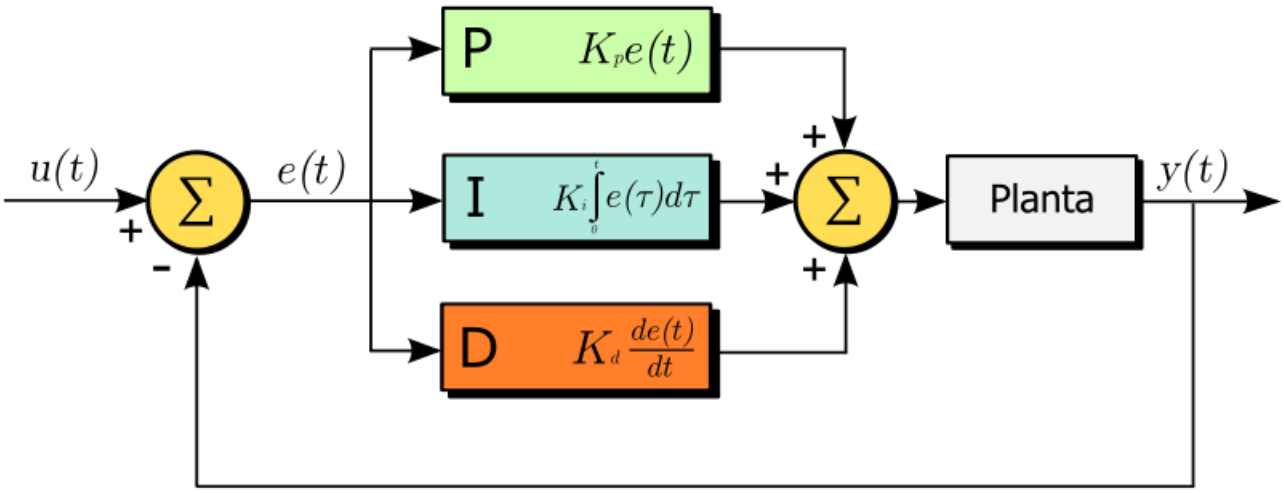


Figure 2.4: PID Regulator Block Diagram [59]

This allows a PID regulator to improve both the system's transient response and steady state response independently. [40]

2.12 Kalman Filter

Kalman filter is a process that is often used in navigation and tracking, since it is based on predicting the states of the system in a future state. This requires the Kalman filter to rely on information about the system, combined with the past estimations computed [8].

The algorithm uses a two step process that is divided into Time Update "predict" and measurement update "correct" as displayed as a cycle in figure 2.5. The Time Update "predict" process, is responsible for predicting the new states and error covariance for the filter ahead of time, based on the last correction. [13].

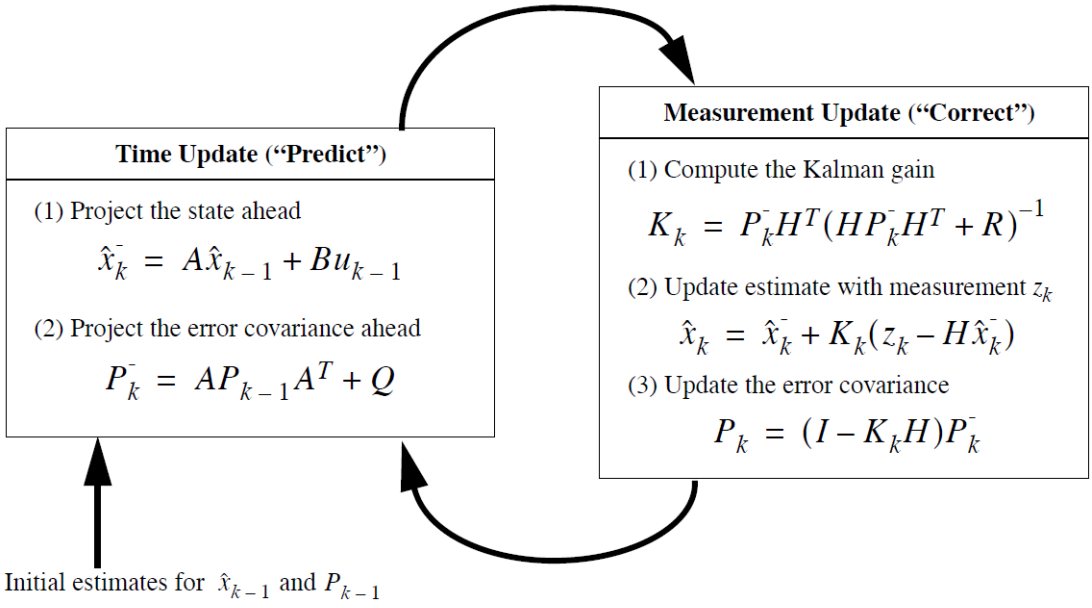


Figure 2.5: A visual representation of the Kalman filter algorithm [13]

Here $\hat{x}_k^-$ is the new predicted state vector ahead of time based on the last estimated states vector, $\hat{x}_k$ fed back from the Measurement Update "correct", and the last input u_{k-1}.

P_k^- is the predicted error covariance, and it points at the error covariance that the filter might experience. Since the system can experience process noise, error covariance is affected by the value of Q, which determines the process noise for the system model [13].

The Measurement Update "correct" feeds back the updated estimates vector $\hat{x}$ with the updated error covariance P_k to further be improved in the Time Update "predict" sequence [13].

Here the first step is to compute the Kalman Gain K_k. The Kalman gain will be affected by the predicted error P_k^- as well as the error of the measurement R, that the model measures.

The Kalman gain is then used to update the estimate state vector $\hat{x}_k$ with the projected state vector combined with the new measurement z_K. Finally the error covariance P_k updated with the Kalman gain and the prior covariance P_k^-. For then to be fed back to the Time Update "pre-

dict" cycle for it to repeat the process again [13].

For a more in-depth introduction to the Kalman filter, see [13].

2.13 Machine Vision

There are several ways for a robot to have vision. One of the possible ways to do this is to use what is called a Lidar.

Light Detection and Ranging (LiDaR) is a form of remote sensing method. It uses light in the form of a pulsed laser, which measures ranges to objects. [42]

The principle for the construction is that it consists of a laser and a sensor together which are mounted on a rotating motor. The laser sends out a pulse which reflects of a surface (an object or surface) and is then received by the sensor which then will calculate a distance based of the time this laser pulse used to return. [42] This concept is shown in figure 2.6

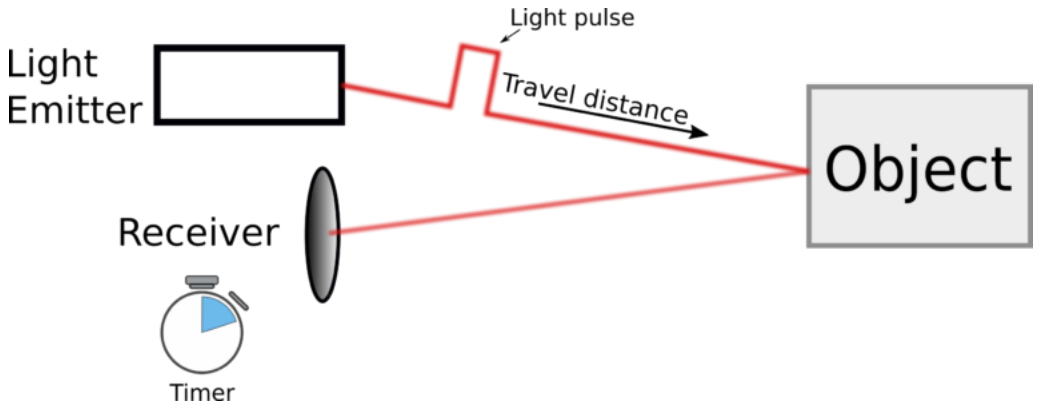


Figure 2.6: Principle of Lidar [44]

2.14 Localization

Localization for a system can be done in two different ways;

- Locally (No relation to world map)
- Globally (Relation with world map)

Depending on what equipment is used, it is possible to create a map and then relate this to the world map or not, or just extract coordinates of robot and relate this to the world map immediately.

Localizing Locally

When localizing locally, different equipment can be used to create a map and find oneself in this map. When doing this there does not need to be a relation to the world map if not explicitly needed.

Localizing Globally

It is also possible to extract the coordinates of oneself using a GPS and accessing a map for the area from the internet.

There is also a possibility to combine the local and global methods to create a bigger certainty of ones position.

2.15 Simultaneous Localization and Mapping (SLAM)

For a robot to be able to navigate without colliding, the robot needs a map to navigate with. With this map, it is necessary for the robot to be able to identify its own position within the map. To find Position, the robot uses a variety of sensors to be able to gather the essential information. A commonly used sensor to find position is GPS, which it uses to localize the robot in the world. But if the GPS-system fails, or is not available, the robot needs to rely on other methods.

Robots can therefore use something called Simultaneous Localization and Mapping, also known as SLAM when GPS is not viable. [38].

Simultaneous Localization and Mapping is the process of gathering sensor data to build a map while simultaneously localize itself inside this map. This makes it possible for a robot to be put in an unknown, unmapped area and it would still be able to navigate trough terrain using the constructed map. This process is used in autonomous navigation. [34].

2.16 Communication Protocols

Communication is important to be able to send information from one client to another. There are several ways to do this communication with different positive sides and negative sides for each type of communication protocol.

2.16.1 User Datagram Protocol

User Datagram Protocol, also known as UDP, is a communication protocol used to transmit data between processes. Compared to many other protocols it is very fast but not as reliable. [39]

The message is sent without checking if the receiver is connected or able to receive, it's a "fire and forget"-kind of communication protocol. The sender sends the packet to the receivers port and IP, and never checks if the message arrives or if there were any problems during sending. [39]

2.16.2 Transmission Control Protocol

Transmission Control Protocol, often refered to as TCP, is a communication protocol used to transmit data between a host and a client. The big difference between UDP and TCP is that TCP is a connection-based transmission protocol. [39]

This means that to send information between host and client the client has to connect to the

host to be able to start the communication.

TCP also makes sure that the information sent is complete and that there are no packets missing from the message sent. This makes it a very reliable communication protocol to use if one is dependant on a reliable communication with minimal loss of information. [39]

2.16.3 I2C

I2C, also known as Inter-integrated Communication, synchronous slave-master protocol intended for short distances between components and microcontrollers. It allows slaves to communicate to a master only using 2 wires; clock and data. To be able to communicate the master and slave has to agree on a specific datarate beforehand. [52]

2.16.4 UART

Universal Asynchronous receiver-transmitter (UART) is a serial communication protocol. It is often used in microcontrollers between the controller and sensors as well as between controllers and computers. [35]

It is quite often used with USB or RS232, using a TTL (Transistor-Transistor Logic) converter. This makes it possible to communicate between small sensors or microcontrollers and computers. [35] Figure 2.7 shows the basic concept of how this works.

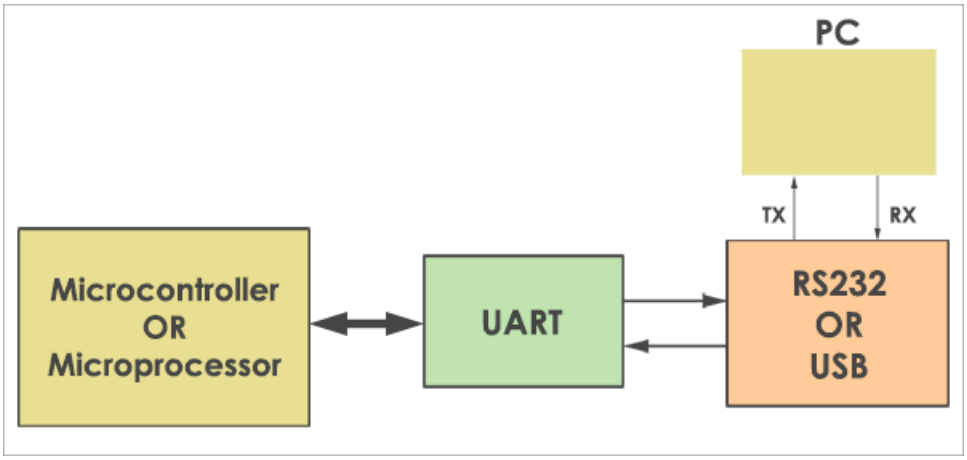


Figure 2.7: How to use USB/RS232 with UART [35]

2.17 Robot Operating System

Robot Operating System (ROS) is a robotics framework designed to make robot control easier. It provides almost all of the elementary functions a robot needs to operate, such as transforms and messaging between processes. [21]

ROS is open source and user made. Any user can add libraries and extra functionality to the operating system. [21] It is also heavily version based, which can make libraries obsolete if they are not continually updated.

ROS is not a real time OS but it uses libraries to access embedded hardware to run code in real time. [21]

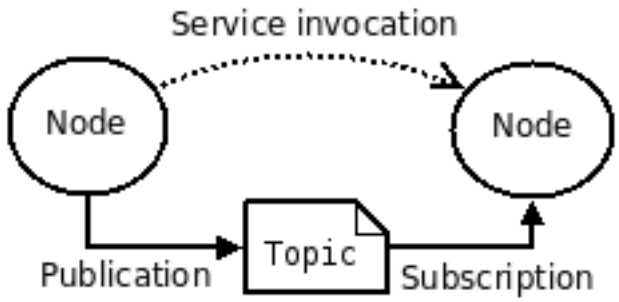


Figure 2.8: Basic concept of ROS-Communication [23]

Nodes is at the core of ROS philosophy. Figure 2.8 shows how these nodes work at a basic level. One node publishes information to a topic, while another might subscribe to this information via the topic the information is published on.

A node represents a process running on the robot. A node executes its code independently and takes action on information received on different topics. [24] Individual nodes do not depend on other specific nodes, only that the information they need is provided on a topic.

2.17.1 ROS Messages

ROS uses topics to send information between its nodes. A topic has a unique name and specifies a message type to use on the topic. This simplifies the communication because nodes have

predefined what type of information they want to receive on a given topic. [22]

Every message type has a specified amount of information that should come in a specific form, and most include a timestamp.

2.17.2 ROS Packets

Different nodes, libraries etc. are stored in what is called Packages. The goal of these packages are to be easy-to-use in any situation and with any other package. [25] There are a lot of pre-made packages that are well-documented found on the ROS-wiki. Packages can also be created by users with ease for any given project using commands such as

```
catkin_create_pkg
```

2.18 Navigation

2.18.1 Costmap

Costmaps are something implemented in ROS through a packet. This packet looks at sensor data or a map and creates an occupancy grid for the navigation of a robot. It looks for obstacles and inflates these obstacles depending on user-specification. [33]

Costmaps are split into two things:

- Global Costmap
- Local Costmap

The global costmap gets the information from a static map. It inflates the cost of the obstacles detailed in this map as shown in figure 2.9.

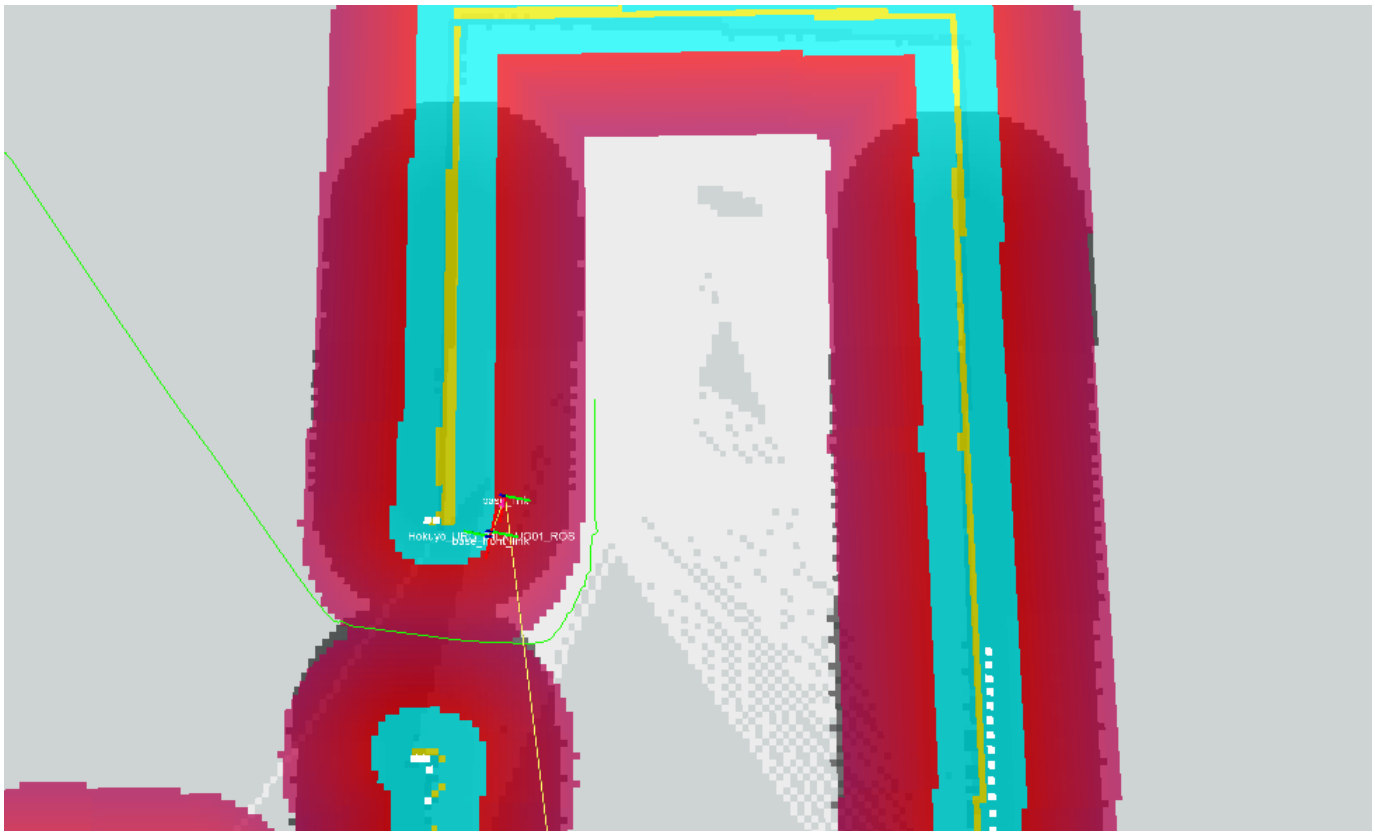


Figure 2.9: Global costmap [27]

The local costmap gets the information from the sensors. It inflates the cost of the obstacles within a given range that the lidar is detecting at the moment as shown in figure 2.10. The red / cyan colors are part of the costmap, and only appears close to the robot.

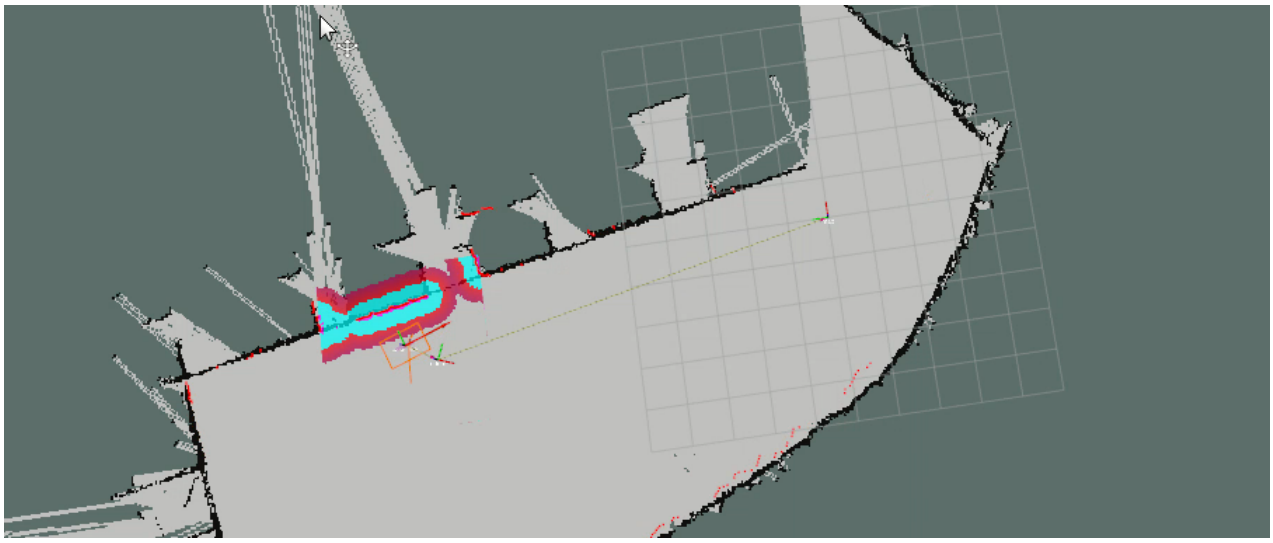


Figure 2.10: Local costmap

What the costmap shows is how it has made a cost around obstacles and objects in the map. The colors show how dangerous it is for a robot to go inside this colored area. The range and inflation of this is something that the user can tweak with parameters.

2.18.2 Pathplanning

A path planner for ROS sends out two different paths;

- Global Plan
- Local Plan

The global plan looks at what has been mapped and creates a path from what information is available from the global costmap. This global plan does not change if new obstacles suddenly appear.

The local plan looks at the global plan and tries to follow it as much as possible. But it also looks at how the robot behaves, and therefore constantly modifies the local plan to try to steer the robot in to the global plan again. [46]

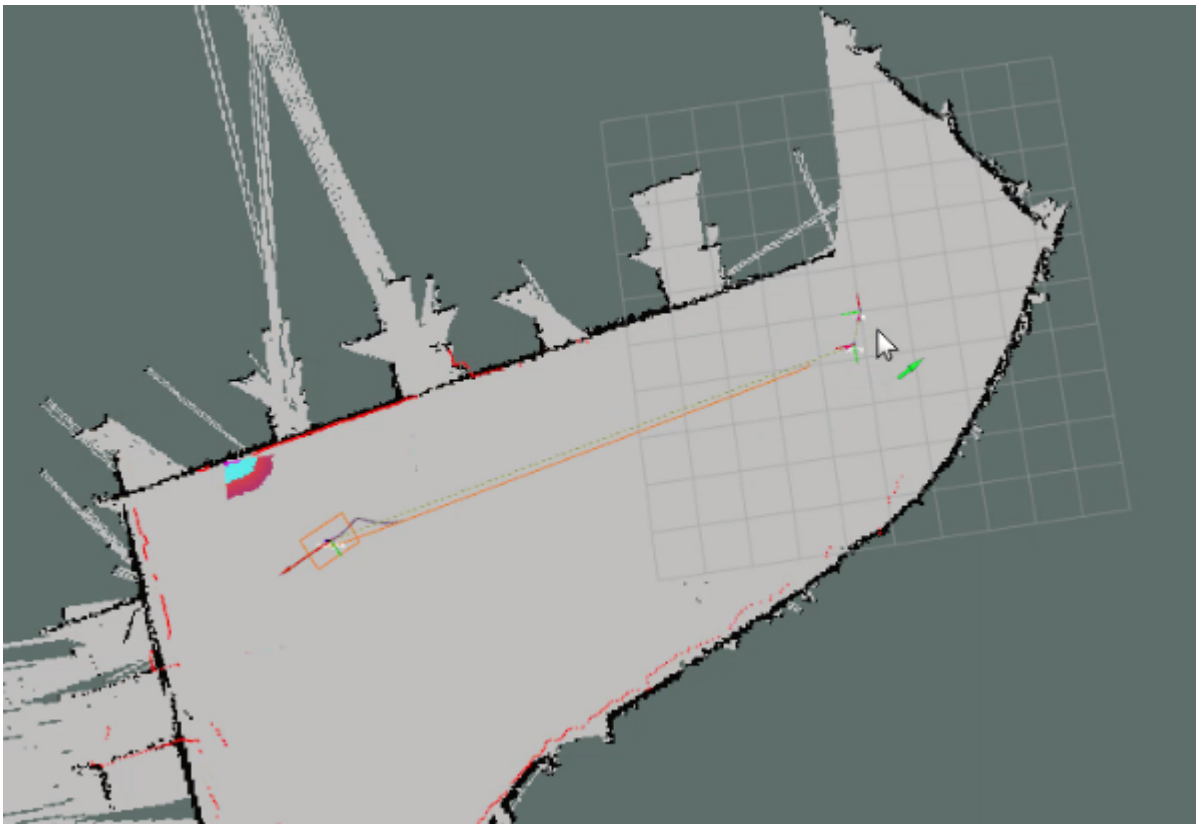


Figure 2.11: Global path in orange and local path in dark purple

As seen in figure 2.11, the orange line is the global path which plans the whole path. The dark purple-color is the local path, and has only planned for a set distance in front of the robot.

If any new obstacles suddenly appear and is detailed in the local costmap, it will also take this into account and try to make a route around this obstacle, while still trying to get back onto the global path.

2.19 Genetic Algorithm

A genetic algorithm, shortened to GA, is an algorithm that imitates natural selection found in nature. In particular the GA emulates two of the components of natural selection, namely genetics and evolution. The genetics component covers genes and chromosomes. These are traits that define an individual in a population. The evolutionary component emulates Darwin's premises, namely that offspring inherit characteristics or traits from their parents, only a fraction of the

offspring survives to adulthood and their survival is dependent on their inherited traits. [14]

A genetic algorithm functions by the following general principles:

- Create an initial population
- Evaluate the fitness of each individual in that population
- Apply genetic operations on the part of the population that survives in order to create offspring
- Check for stop criterion
- Repeat until results are satisfactory

A flow diagram describing this algorithm can be seen in figure 2.12

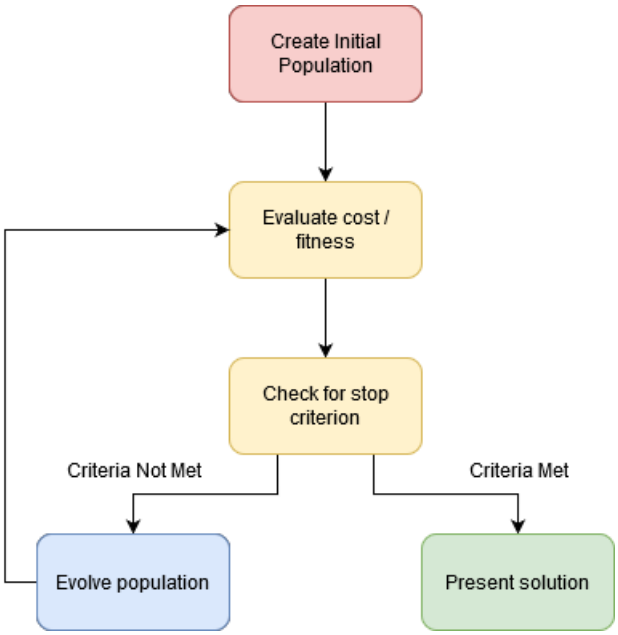


Figure 2.12: Genetic algorithm flow diagram

The initial population consists of a set of chromosomes / individuals. These individuals all have genes, which are a set of values, either binary or continuous. Each individual's performance is evaluated by a user defined fitness / cost function. A set containing the best individuals are then selected for mating, where each individual in that set has their genes merged with

a mate in various ways, depending on implementation, to create new individuals / offspring. These offspring might then be subjected to mutation, meaning that some of the genes in an individual gets assigned a new value. This new population is again evaluated, and the cycle continues until a certain stop criterion is met.[14]

In addition to these steps it is common to implement elitism in the algorithm, which means that the best individual is replicated to the new population. This ensures that the performance of the best individual never decreases.

2.19.1 Cost / Fitness Function

The evaluation of an individual is done using a cost or fitness function. The general difference between a cost and fitness function is that when applying a cost function a lower score is better, while a higher score is desired when using a fitness function.

These functions are what define the problem to be solved and are therefore entirely dependent on application. The core GA algorithm can be used for many different problems, but the cost and fitness functions are tailored to the specific problem.

2.19.2 Pairing Methods

There are several ways of pairing and mating surviving individuals. A simple approach to paring surviving individuals could be to pair the best with the second best, the third best with the fourth best and so on. Another method is to simply select the pairings randomly among the surviving individuals [14]. One could also incorporate weighted random pairing by weighting the individual's rank and / or cost.

2.19.3 Mating Methods

There are many approaches to the mating procedure of a genetic algorithm. Two common ones are the single point crossover method or the blending method.

The single point crossover method is done by selecting a number between 1 and the size of the gene pool of the individual. This is the crossover point. The offspring inherits parent number 1's genes up until that crossover point and inherits parent number 2's genes from that point.

The blending method assigns values to the offspring's genes based on a mix of the parent genes. The values for which could for instance be an average of the parents' genes or a weighted average based on fitness and / or rank.[14]

Chapter 3

Preliminary Test Results

This chapter will present the preliminary test results that formed how the methodology for the rest of the project became. It is mostly focused around the building of the prototype, and what was learned from the iterations that were built and tested.

3.1 Design Reviews

3.1.1 First Iteration

The Design

The first iteration of the prototype was designed with waste tubes with a diameter of 50mm. These tubes were supplemented with 3D-printed T-crosses as well as plugs with a seal to ensure the tubes being waterproof.

The build plate on top was a plexi-glass plate with dimensions 50 cm x 70 cm, with holes thought for mounting electrical components. Corner brackets for mounting this plate to the tubes were modelled and 3D-printed by members of the group. A render of this design can be seen in figure 3.1

The total size of the prototype was 50 cm x 70 cm x 50 cm.

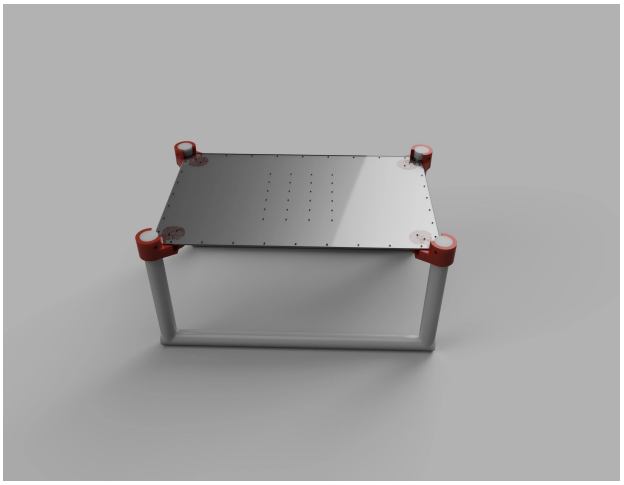


Figure 3.1: 3D-Model of the 1st iteration without electrical components

Testing of Design

During testing of this prototype, it was found that without any form of ballast in the horizontal pipes, the vessel would just flip over the second it hit the water. When running this iteration through the stability calculator created later it showed that this would happen.

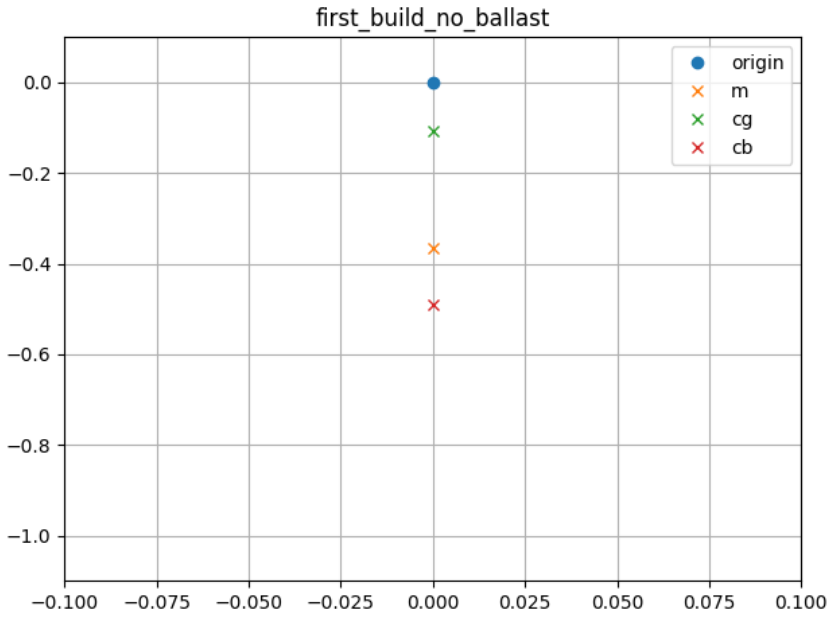


Figure 3.2: Overview of center of gravity, center of buoyancy and metacenter without ballast

When using what is described in section 7.1.3 and reading the graph in figure 3.2 it is possible to see that the center of gravity is higher than the metacenter, and the prototype is therefore

unstable.

```
Parameters:
    Platform width:         0.5 [m]
    Platform length:        0.8 [m]
    Pipe diameter:          0.05 [m]
    Vertical pipe length:   0.5 [m]
    Horizontal pipe length: 0.8 [m]
    Ballast density:        0.0 [kg/m³]
    Additional ballast:     0 [kg]
Centre of gravity:   [0.0, 0.0, -0.10685] (x, y, z)
Centre of buoyancy:  [-0.0, 0.0, -0.49154] (x, y, z)
Total weight:        3.9539999999999997 [kg]
Submerged volume:    0.0038575609756097556 [m³]
Submersion / rate:   0.09115991803734946 [m] / 0.18231983607469893 [ratio]
```

The data over also shows that the submersion rate of the vessel is 18%, which is also too low and the prototype would be too high in the water.

The prototypes horizontal piping was filled with water to get it to lie lower in the water. Doing this made the prototype lie with the base plate right on top of the water, which is too low. It increased its stability, but it was still unstable. This was also run through the stability script calculator.

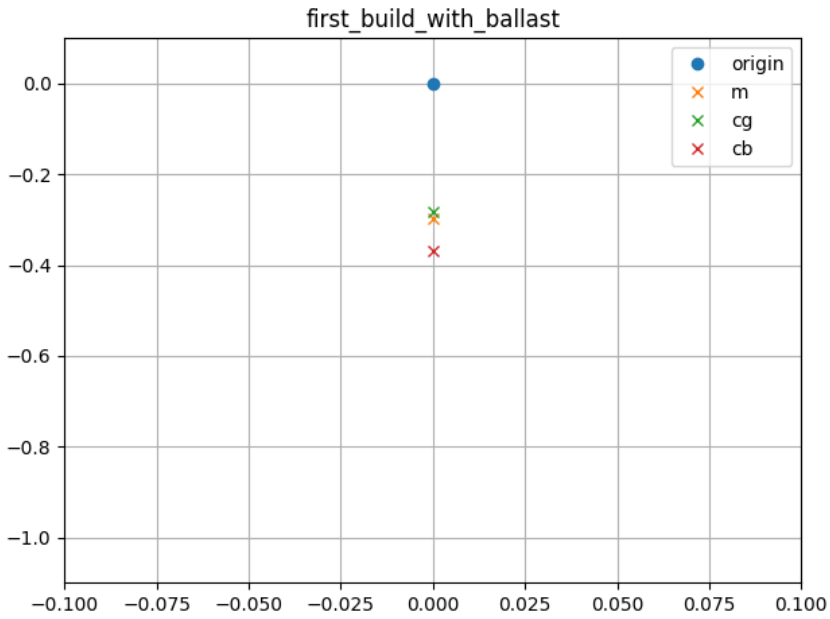


Figure 3.3: Overview of center of gravity, center of buoyancy and metacenter with ballast

As seen in figure 3.3 the prototype is still not stable at rest.

```
Parameters:
    Platform width:         0.5 [m]
    Platform length:        0.8 [m]
    Pipe diameter:          0.05 [m]
    Vertical pipe length:   0.5 [m]
    Horizontal pipe length: 0.8 [m]
    Ballast density:        1000.0 [kg/m³]
    Additional ballast:     0 [kg]
Centre of gravity:   [0.0, 0.0, -0.28092] (x, y, z)
Centre of buoyancy:  [-0.0, 0.0, -0.36853] (x, y, z)
Total weight:        7.095592653589793 [kg]
Submerged volume:    0.006922529418136383 [m³]
Submersion / rate:   0.48140382047637387 [m] / 0.9628076409527477 [ratio]
```

The data also shows that the submersion rate is 96.2% which is way too low in the water. This made the design not viable, and needed to be reworked.

3.1.2 Second Iteration

The Design

The second iteration tried to fix the problems regarding buoyancy and stability that were present in the first iteration. The design tried to fix these problems by increasing the amount of vertical tubes and the concept is displayed in figure 3.4.

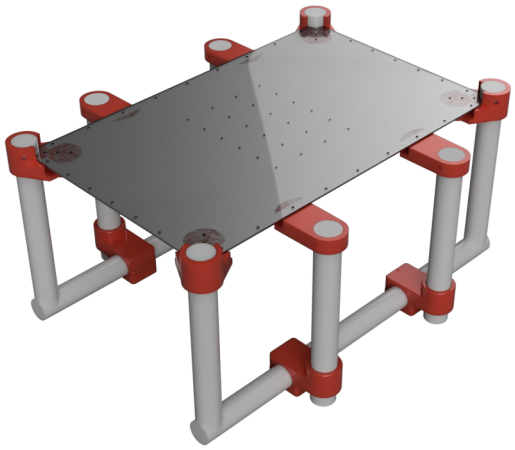


Figure 3.4: 3D-Model of the 2nd iteration without electrical components

Testing of Design

Before starting the construction of this, a ship-design student was consulted to help with some calculations. After acquiring how to calculate the stability and buoyancy of the boat (as described in section 4.5.5) this design was scrapped before the construction even began.

The reasoning for this was that the total weight of all electrical components had been calculated and showed that the amount of buoyancy needed was higher than this design supplied. The reason the weight of components had been calculated so late was because of uncertainty of what motors would be supplied to the prototype.

3.1.3 Third Iteration

The Design

After acquiring the methods to calculate the stability and buoyancy of the prototype, calculations for the amount of weight the electrical components would be were done. After this some test calculations were done for what diameter the tubing should be.

From this some new design possibilities were made. They were all still based on the original design of the semi-submersible but deviated in size and the amount of tubes. There were two primary designs presented;

- Design 1:
  - 2 Horizontal Pipes
  - 4 Vertical Pipes
  - Base plate with a size of 1m x 1.4m
- Design 2:
  - 2 Horizontal Pipes
  - 6 Vertical Pipes
  - Base plate with a size of 0.7m x 1m

Design 1

With a base plate of 1m x 1.4m this design would be a total of 1m x 1.4m x 1m in size. The bigger base plate helps with the stability.

The calculations for this design was run through the stability calculator.

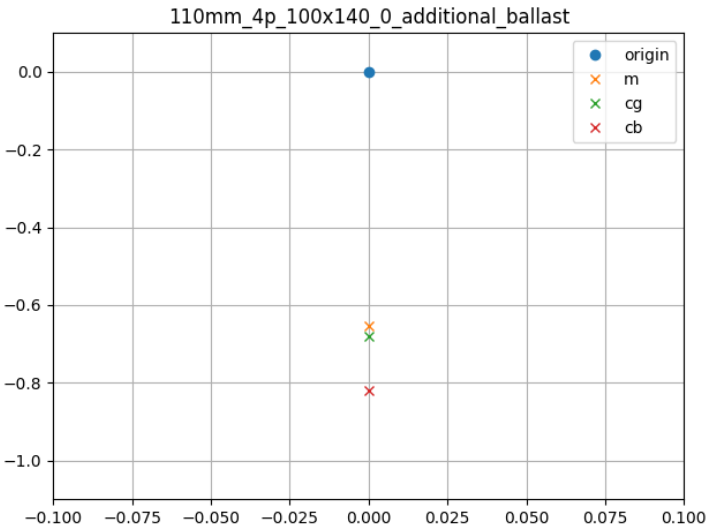


Figure 3.5: Overview of center of gravity, center of buoyancy and metacenter

As can be seen in figure 3.5, the metacenter is higher than the center of gravity, which is good. But this is by a close margin, which gives very little leeway in the case of more weight than calculated for being added to the top plate.

```
Parameters:
    Platform width:          1.0 [m]
    Platform length:         1.4 [m]
    Pipe diameter:           0.11 [m]
    Vertical pipe length:    1 [m]
    Horizontal pipe length: 1.4 [m]
    Ballast density:         1025.0 [kg/m³]
    Additional ballast:      0 [kg]
Centre of gravity:    [0.0, 0.0, -0.67997] (x, y, z)
Centre of buoyancy:   [0.0, 0.0, -0.82013] (x, y, z)
Total weight:         58.18952202030319 [kg]
Submerged volume:     0.056770265385661656 [m³]
Submersion / rate:    0.740819581914543 [m] / 0.740819581914543 [ratio]
```

The submersion rate is also quite nice at 74% the boat would be mostly submerged, but enough space to account for eventual waves that could hit the boat.

Design 2

The second design had the base plate at a size of 0.7m x 1m but increases the amount of vertical tubes to 6. This makes the size of the design 0.7m x 1m x 1m.

This design was also run through the stability calculator and the results can be seen in figure 3.6.

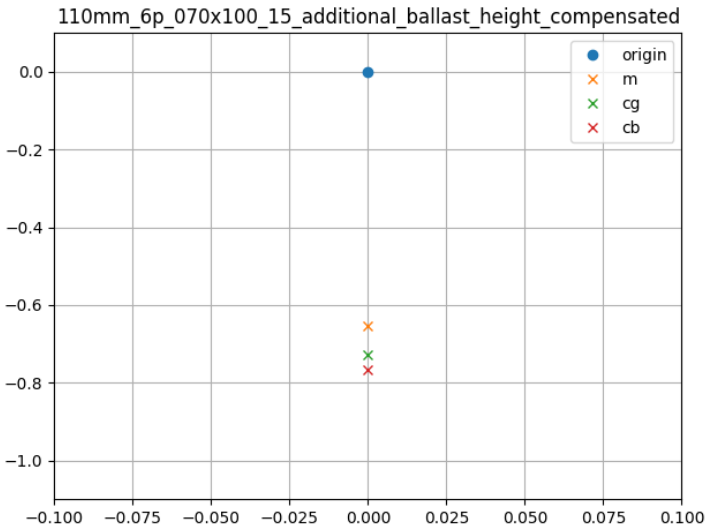


Figure 3.6: Overview of center of gravity, center of buoyancy and metacenter

With 15kg of extra ballast in the vertical pipes, the metacenter height is higher than the center of gravity by a good margin. This means that if the weight would increase by a bit, it would be possible to remove some of the extra ballast in the vertical pipes to account for this.

The data from the calculations are shown under:

```
Parameters:
    Platform width:          0.7 [m]
    Platform length:         1.0 [m]
    Pipe diameter:           0.11 [m]
    Vertical pipe length:    1 [m]
    Horizontal pipe length: 1.0 [m]
```

```
    Ballast density:        1025.0 [kg/m³]
    Additional ballast:     15 [kg]
Centre of gravity:   [0.0, 0.0, -0.72913] (x, y, z)
Centre of buoyancy:  [0.0, 0.0, -0.76769] (x, y, z)
Total weight:        63.311801443073705 [kg]
Submerged volume:    0.06176761116397435 [m³]
Submersion / rate:   0.7148551794535389 [m] / 0.7148551794535389 [ratio]
```

With a submersion rate of 71.5% there is enough space between the base plate and the ocean, but still safe for eventual waves that would've hit the prototype.

Final Choice

The final choice landed on 6 vertical tubes and a base plate with a size of 0.7m x 1.0m. This is because of mainly two factors;

- The height difference between the metacenter and centre of gravity is bigger
- A base-plate of 0.7m x 1.0m is easier to get as the plexi-glass plates come in that size

With this in mind, the construction of the third iteration was done, and the final result is shown in figure 3.7.

Figure 3.7: 3rd iteration without electrical components

The total size of this iteration was 1.3m long, 0.8m wide and 1.1m tall.

Testing of Design

This design was constructed as a whole and the testing of the stability and how it would float in the water was done with it. The calculations showed it would float quite well, and there would be enough space between the upper platform and the water line to not risk the electrical components being flooded.

Initial tests on water showed that this was correct. It floated quite well and was stable when pushed. It did not flip over and the submersion rate was quite good. It was also waterproof and

did not present any critical problems outside of this.

What was discovered was that any type of push to the sides of the prototype made a big impact and the prototype shifted sideways for a long time before falling to rest again. To mitigate this some fins were mounted to the horizontal piping, and tests showed that this helped the amount of impact by small pushes were reduced drastically.

There was still a big problem with it being sensitive to uneven weight distribution, and this problem was exacerbated by the battery being mounted to the top plate. The design was therefore found flawed, and redesigned to mitigate this problem.

3.1.4 New Knowledge Acquired

From these tests a lot of new knowledge was acquired about how the prototype should be designed and exactly why that is so. A lot was learned about how even though the prototype was stable, that uneven weight placement would have a lot of impact for how the boat would behave.

A lot was also learned about how things that might not be visible in the calculations of stability might come forth when putting it into water and testing it, like how the platform would perform with regards to duration of roll and pitch oscillations, and how the platform would respond to uneven weight distribution.

3.2 Localization and Navigation

The initial tests of localization and navigation were done before the prototype was up and running. These first tests of localization were mostly to figure out how to get the odometry right, as a lot of information was required for the localization algorithms to be able to keep track of the movement.

These tests were first done by holding a bracket with the Lidar and an IMU mounted together and guiding it through a room. Later an RC-Car was acquired and the Lidar and IMU mounted

to this as shown in figure 3.9.

One of the big challenges was regarding velocity and translation. The first tests were trying to integrate the values from the accelerometer to try to get velocity and translation from this. These early tests showed that this was way to unstable when double integrating the accelerometer and caused a lot of drifting in the map shown in figure 3.8. This instability was because of the presence of quite a lot of noise on the IMU-signals.

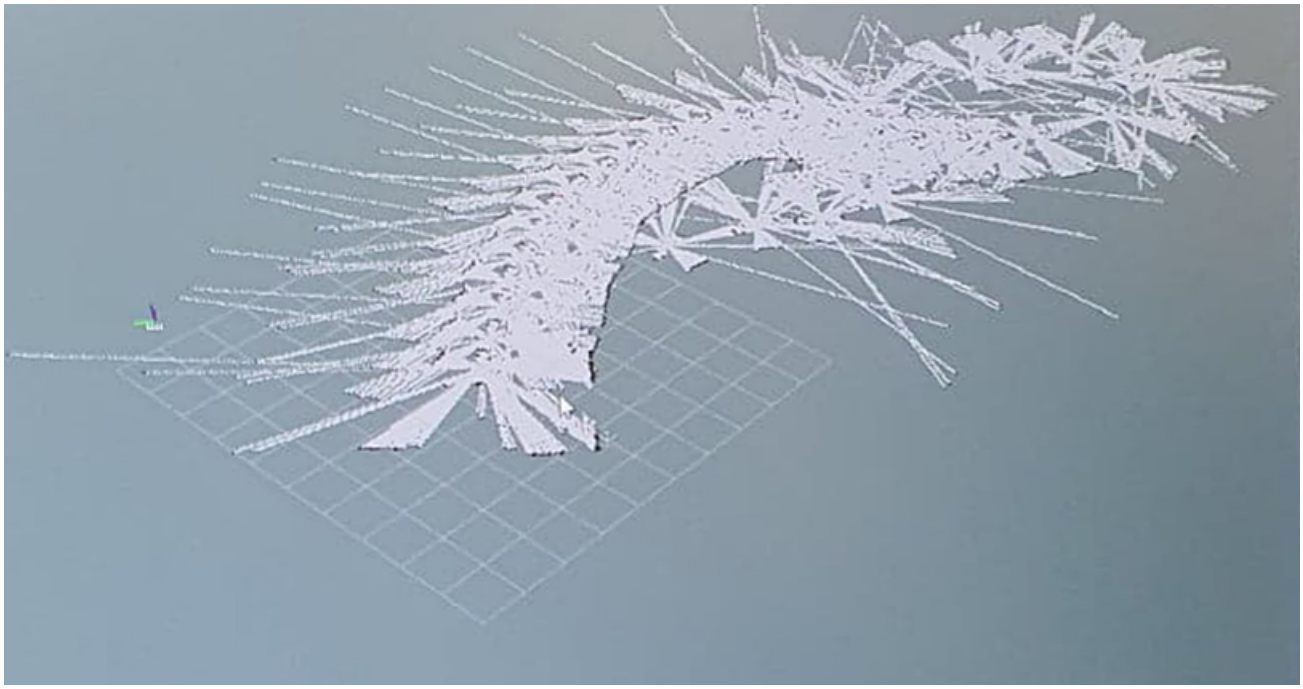


Figure 3.8: Drifting in map caused by double integrating accelerometer data

When this failed, a package named laser_scan_matcher was found that gives out translation and orientation using the lidar as a source. When this was implemented, all information that was needed was supplied to the localization-algorithms.

Initial testing of the navigation stack were done in the living room of one of the members using a small RC-car. Because of this car not being omni-directional like the prototype is, the lateral shifts were not tested. The car is shown in figure 3.9.

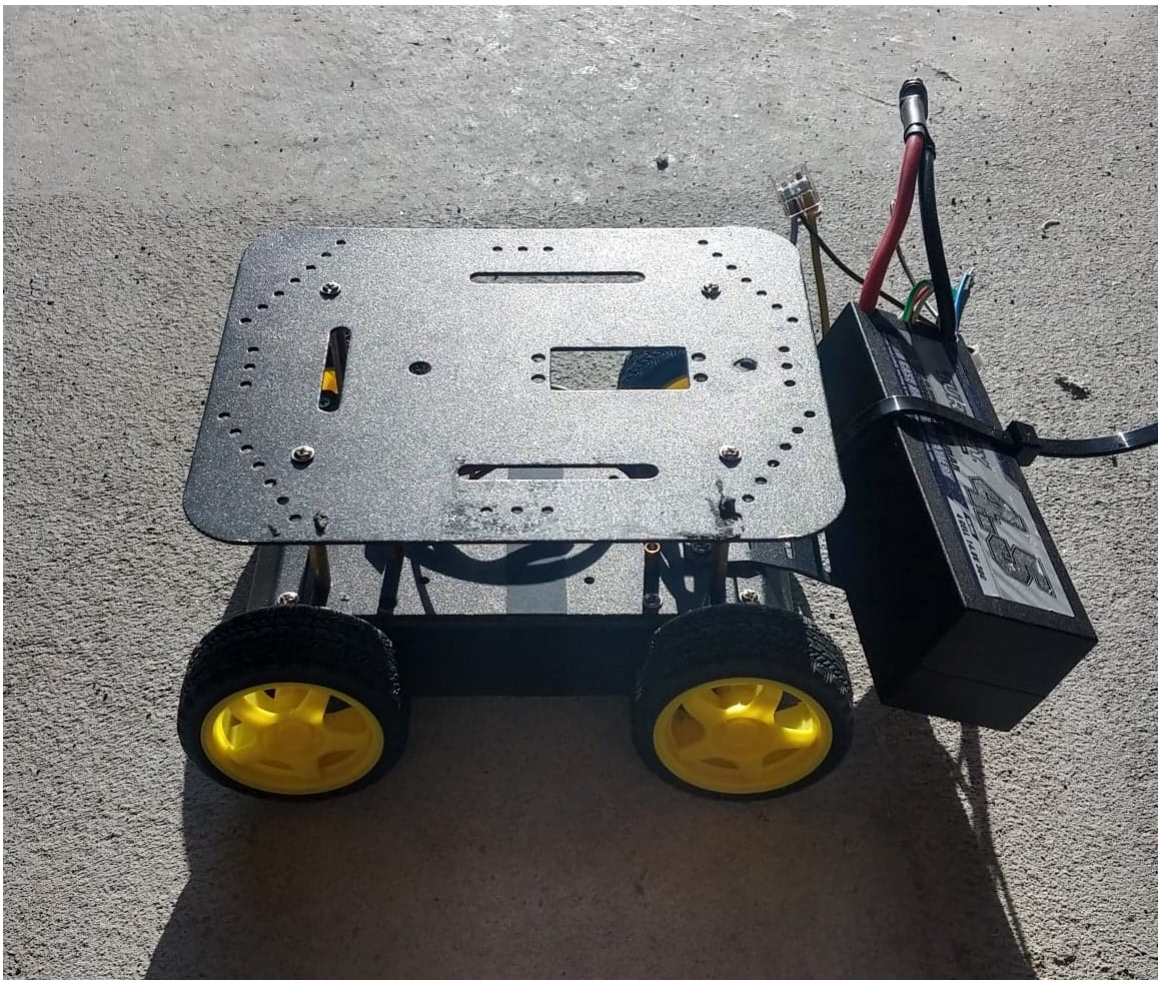


Figure 3.9: RC-Car used during testing

During these tests the local planner used for testing was dwa_local_planner. This planner is one of the less process-heavy local planners compared to others. It also had the possibility of enabling planning for omni-directional robots, so it seemed fitting.

General tests of sending coordinates and checking how the navigation went were done. These tests showed that the Raspberry Pi 4 had huge problems managing the amount of processing power needed to process further than 0.5 seconds forth in time.

This created other problems such as it getting within centimeters of the goal then starting to rotate in place not being able to get the last stretch. At this point the testing was discontinued as the prototype was up and running.

3.2.1 New Knowledge Acquired

These early tests gave the group a lot of experience in how to effectively tweak the localization to a point where it was stable, and what parameters had a big influence on the different factors. Some experience was gained in the navigation stack as well, but not as much as the localization-part, this was because before the navigation stack was tuned properly the prototype was finished and ready for testing.

Chapter 4

Method

This chapter provides an explanation of how different parts and problems of the project can be handled. It also provides an overview of how the project and group was organized in an attempt to make the execution of the project as efficient as possible.

4.1 Project Organization

The group quickly organized certain positions for the members to fill, so there was a hierarchy if need be at a later point. But there was an agreement on trying to resolve issues or disagreements with everyone, instead of letting one person decide.

The positions created were;

- Group Leader
- Secretary
- Chief of Deviations

This made it possible to automatically give people the responsibility in case of certain situations. The Group Leader was to be the deciding factor if the group could not reach an agreement in the case of some issue. The secretary was to write a report for each weekly progress meeting

held as well as the meetings with the supervisors.

In the case of something happening that had an impact on the project as a whole, the Chief of Deviations was the person to process this and call a meeting to discuss possible solutions.

If there were bigger decisions to be made, a meeting was held to get better time to discuss the possible drawbacks or positive sides of different decisions.

Meetings were held every week for the group itself, with the reports from these meetings being sent to the supervisors. Meetings with the supervisors were to be held every second week.

Tasks were given out to the members that felt most fitting or that they had experience in. There was a focus on trying to keep the amount of work quite equal to be sure that one person didn't become stuck with the biggest amount of work.

Progress on the work was monitored and updated using a Gantt-diagram, where all tasks were created and given a time-frame for finishing. This diagram can be seen in appendix A.3.

Documents, code as well as diaries and the timesheet were all saved in the cloud, using the Microsoft Teams-app. This made it quite easy to share documents, figures as well as movies/pictures with each other when necessary.

The main area of work was to be at NTNU Ålesund, using the data labs for all office-work, and the labs downstairs for the practical work. When it came to testing the prototype, the area outside Sunnmøre Museum was to be used, as there was quite an open space there to work freely at.

4.2 Extraordinary Situation (Corona)

During the thesis an extraordinary situation happened that no-one would have expected. The corona-virus pandemic came to the country, and most of the city was shut down until further notice. There were signs that this could happen just a few days before, so some preparation was able to be done for this situation.

The groups response to the message that NTNU would shut down until further notice was to gather all the parts of the prototype as well as all parts and components needed generally and load them up in the cars. It was agreed upon to take all of this home to the garage of the group leader, Håkon B. Waldum.

From there, a risk assessment had to be done as this was a situation with some serious implications for what could happen if it were not taken seriously. During this risk assessment a new plan for how to be able to work properly on the project was decided upon.

The risk assessment shows the thought process behind how this methodology was decided upon, as well as why this was decided upon. It was decided to start working more separately where possible, and reduce the amount of time the group had to meet in person.

One other precaution that was taken was that when the group had to meet in person, there were hygiene-precautions that would be taken. All members were to clean their hands properly more often, and sharing food, beverages etc. was not acceptable.

After talks with the supervisor, Ottar L. Osen, it was also decided to start having the meetings weekly instead of bi-weekly. This to ensure that the group had everything they needed, as well as to be sure that the work went as expected and didn't start lagging behind.

Because of the situation NTNU also sent out a notice that groups that started lagging behind had the possibility of applying for an extension on the deadline for the thesis. In a discussion

with the supervisor, Ottar L. Osen, an agreement was made that the work progressed as expected for the time being, and an eventual application for extension on the deadline was not needed, but that this was something to keep in mind during the work.

4.2.1 Initial Working Plan vs. Actual Working Plan (Corona)

The original working plan was to work mostly at school with both the "office-work" (programming, report etc.) as well as the physical construction of the prototype. And for the testing of the prototype the plan was to use the area outside Sunnmøre Museum to test it.

All of this changed after the corona-situation happened, as NTNU closed. Because of this office-work was now to be done alone at home, and the physical construction of the prototype was to happen in the garage of one of the members.

For the testing-part it was decided to use the SIT-boathouse to be able to store the prototype there as well, as transporting the prototype meant dismantling it because of its total size.

A lot changed for the working plan, but it didn't stop the progression too much, as all the members together had mostly everything needed (both tools and space) to keep the progression up.

4.3 Software

There are several softwares utilized in this project, listed in table 4.1.

Program	**Description**
PyCharm	*Pycharm* is an IDE used for writing Python-code
Matlab	*Matlab* is a software used to create filters and mathematical models for the project
Arduino IDE	*Arduino IDE* is an IDE used for writing Arduino-code

Fusion 360	*Fusion 360* is a CAD software used to model the parts that were manufactured by the group.
Cura	*Cura* is a 3D-Printing slicer software.
Tracker	*Tracker* is a video analysis software that is used to track an object frame by frame to compute a position-time graph
AutoCad Electrical	*AutoCad Electrical* is a CAD software used to draw electrical diagrams.
Basic Micro Motion Studio	*Basic Micro Motion Studio* is a software used to configure and monitor Roboclaw-units.

Table 4.1: Overview of software utilized in the project

4.4 Concept Studies

When choosing a concept for the prototype there are several factors to take into account. Stability and buoyancy has to be accounted for to ensure that the prototype will stay afloat without any problems.

One popular design for simple boats for the kind of testing purposes as in this project is the semi-submersible concept. This is the same concept that oil platforms are built on. This concept reduces the effect that the sea will have on the boat, in that the amount of surface that the waves can impact is low, while also keeping the center of gravity low. This concept can be seen in figure 4.1.

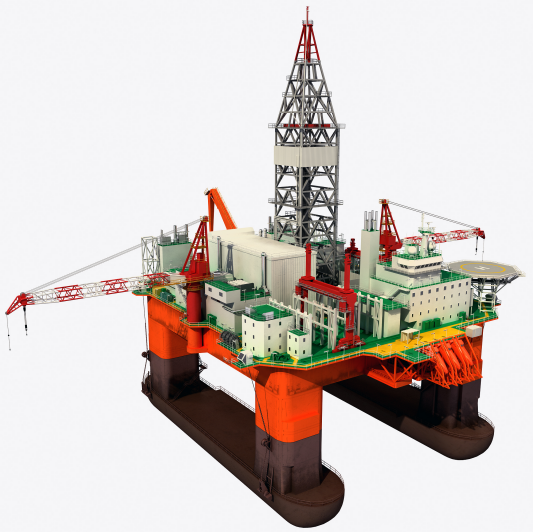


Figure 4.1: A small-scale Oil Platform [1]

4.5 Design

This section will describe the steps taken in order to come up with a design for the prototype.

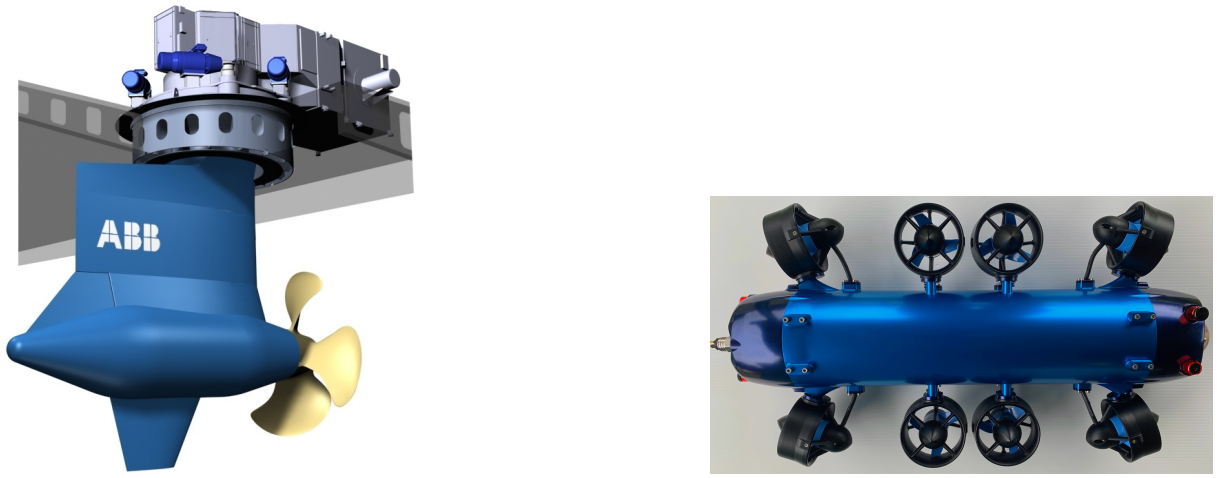
4.5.1 Concept

The concept for the prototype is based on a semi-submersible platform in order to limit the impact of small waves by the dock. This is an important factor as a larger ship does not get affected by these waves as a scaled down model would. The semi-submersible configuration, although less stable, will be less affected by said waves and therefore better emulate a larger ship. [48]

4.5.2 Propulsion

To be able to move a boat of this size, a certain amount of power is needed. The propulsion method has several ways to implement this.

One of the ways is to use underwater thrusters, which is a simple under water motor with a propeller that gives thrust in a given direction. These thrusters can be statically mounted, or



(a) Rotating Thruster [29]

(b) Statically Mounted Thruster [28]

Figure 4.2: Comparison of Thrusters mounted on rotating surface and statically mounted

they can be mounted on a surface that makes it able to be rotated as shown in figure 4.2.

There are positive sides to making these thrusters able to rotate, but this also creates a need for a more complex system to regulate the positioning of a boat. But on the other hand if they are statically mounted, this will possibly reduce the effectiveness of a given thruster.

4.5.3 Material Selection

For constructing waterborne vessels there are several materials that are popular in use because of their robustness as well as their ability to be water tight. Some metals as well as quite a lot of types of plastics are often used in these cases. Plastics are often cheap and metals are often very robust and can survive water with no big difficulties.

For metals aluminum is very popular to use, specifically because of its resistance to corrosion as well as its strength-to-weight ratio [51]. It is also a good choice because of its non-magnetic properties which is positive when mounting big electrical systems inside these hulls.

For small-scale operation plastics are a good choice because of the cheap price for components made from plastics, and the ease of creating the needed components in a small enough size. One thing that has to be mentioned is the growing problem of micro-plastics in the ocean,

which has several negative effects on the climate and aquamarine life [41], which is a reason to reconsider using plastics for waterborne vessels.

4.5.4 3D-Modelling

For a prototype it is often hard to find parts to fit exactly what is wanted. To supplement this 3D-modelling and 3D-printing can be a great tool to use. One thing to note is that 3D-printing takes a lot of time, and requires a certain amount of material not easily gotten from stores in Norway.

4.5.5 Stability

With the concept described above in mind, it is necessary to do calculations for stability of the platform in order to determine which components can be used to solve the design. For these calculations a ship-design student, as well as Håvard Lien was consulted. In order to determine the platform's stability it is important to calculate the metacentric height as described in section 2.7.2 as the distance between the vessel's centre of mass and the metacentre.
The centre of mass between particles can be calculated as

$$C_G = \frac{1}{\sum_{n=1}^{N}(m_n)} \cdot \sum_{n=1}^{N}(m_n \cdot v_n) \tag{4.1}$$

where:
C_G - Centre of mass
N - Number of particles
m - Mass
v - Location vector

A way to calculate the centre of mass for the prototype could be to use equation 4.1 and treat individual components of the prototype as particles in a 3-dimensional grid.

To determine the metacentric height, one can use equation 2.6. In order to use this equation, the centre of buoyancy has to be known, as well as the second moment of area at the water

plane and the displaced volume.

Determining the centre of buoyancy can be broken into two steps. The first step is to calculate how far the vessel will be submerged. This can be done using Archimedes' principle (equation 2.5). This equation will yield the displaced volume required in order to negate the weight of the vessel. This requirement can then be used in order to determine how far the vessel will be submerged before the buoyant forces negates the forces from gravity.

The second step is to calculate the centre of the displaced volume. This will represent the point at which the buoyant forces act on the vessel and will be referred to as the centre of buoyancy. The centre of buoyancy can be calculated the same way as the centre of mass, as shown in equation 4.1. Instead of mass, this equation will consider displaced volume. The equation becomes

$$C_B = \frac{1}{\sum_{n=1}^{N}(V_n)} \cdot \sum_{n=1}^{N}(V_n \cdot v_n) \tag{4.2}$$

where:

C_B - Centre of buoyancy

N - Number of particles

V - Displaced volume

v - Location vector

Determining the second moment of area can be done using Steiner's theorem since the area of the cross section of the vessel does not lie on the axis of rotation, which states:

$$I = I_0 + A \cdot y^2 \tag{4.3}$$

where:

I - Second moment of area around rotation axis

I_0 - Second moment of area for the shape when a parallel rotation axis passes through the centre of area

A - Area of the shape

y - Distance between rotation axis and parallel rotation axis

With these equations it is now possible to calculate the centre of gravity, centre of buoyancy and metacentre. These values can be used to evaluate the vessel's stability.

4.6 Data

During the development of the prototype a lot of data logging was done to be able to evaluate the performance of all the systems. A data logging script for collecting all this data was developed in order to achieve this goal. The script is developed in a way that it was easy to add new data if there was a need for it.

The script subscribes on ROS Messages to continually listen to information sent between the different nodes in the system. All the information is saved to the harddrive at fixed interval to save on memory and make sure the data doesn't get lost during a crash.

The relevant data would be all of the data from the IMU (Gyroscope, Accelerometer, Magnetometer) the position of the prototype, the velocity commands sent, the commands sent to the motor controller as well as the global path.

All of this data can be used to continually improve upon the prototype in different forms, as well as detect problem areas.

4.6.1 Processing data

To process a lot of the data that is acquired, there are several softwares which can efficiently plot and process it. Matlab is one of the strongest mathematical programs available to students through the university, and enables creating own functions which will streamline the process of processing the data from different tests and runs.

4.6.2 Using Tracker to process certain data

The video analysis software Tracker can be used to compute a theoretical position-time graph from some of the test runs of the vessel. This test can be done by filming the boat moving in one direction multiple times, with the camera filming in a fixed position. This should be done multiple times to gather a set of data that can later be analyzed further.

By uploading the film of the vessel moving, and selecting the frames where the vessel starts and stops moving. Tracker can then track the vessels position, frame by frame, with Trackers own auto tracker function. To be able to use this function, there is some criteria that needs to be fulfilled.

The first criteria is that a start position needs to be defined, usually where the initial speed is zero. This can be done by inserting the coordinate system in Tracker onto the position where the vessel first starts moving. Another important criteria is to define a fixed measured length for tracker to define a measure scale in the frames. To define a fixed length, the "calibration stick" can be used to define a measured length, in this case the length of the vessel itself can be used. Finally, there needs to be specified a mass, or an object to track. By selecting "create" and the vessel, the vessel will be selected as the mass. A complete setup can be observed in figure 4.3

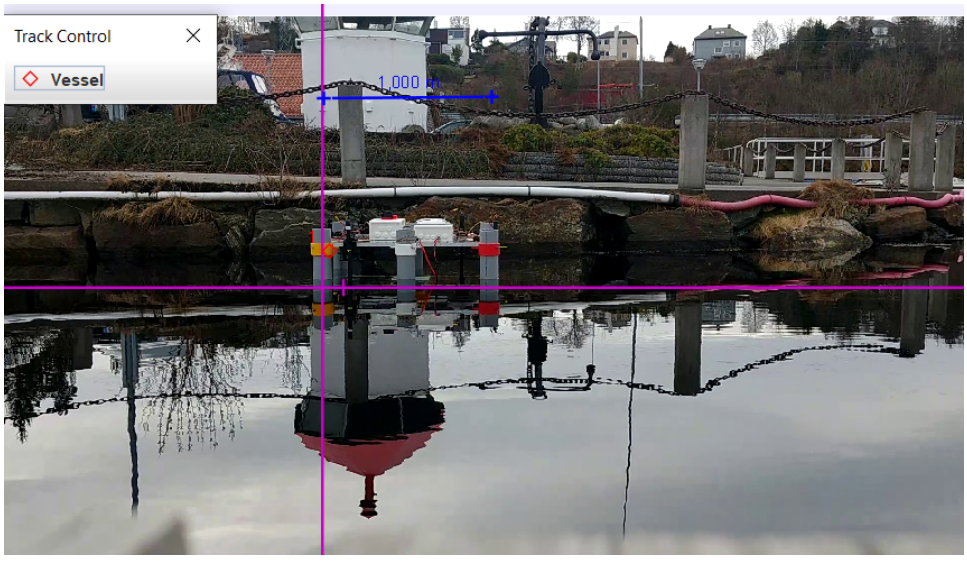


Figure 4.3: A representation of how the initial criteria can be placed.

Then when the auto tracker function is done tracking the vessel from the start to the stop position. Tracker will then return a plotted x, t -graph that represent the position over time for the tracked mass as can be inspected in figure 4.4. This plot can be analyzed to see how far the vessel has moved over a period of time, based on the displacement of the vessel in frame to frame.

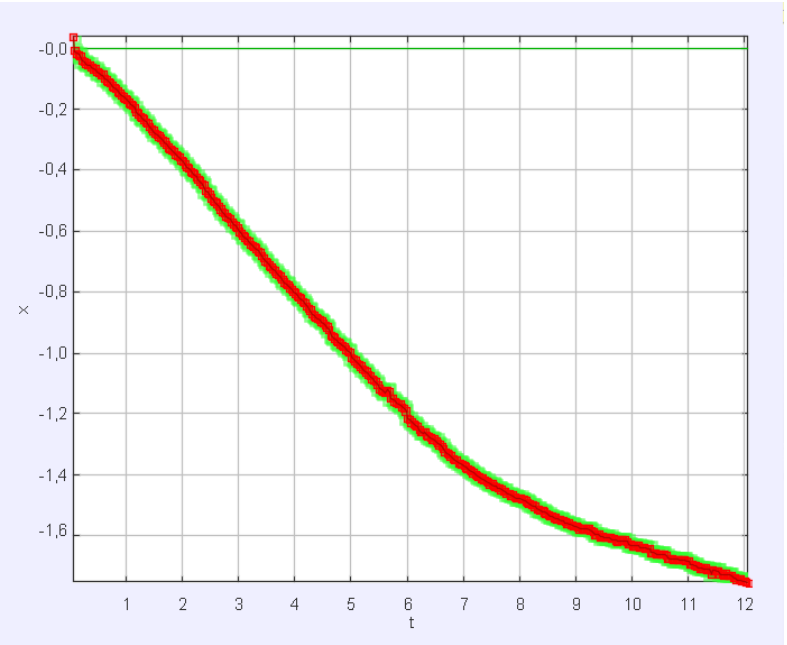


Figure 4.4: A representation of the x,t-graph Tracker returns

The returned position, time -graph can further be analyzed with Trackers own "analyze" function. This functions allows for the plotted graph to be fitted with other line equations, for example linear or parabola equations. By selecting the Parabola equation and then the "Autofit" options, Tracker will try to match the plotted function with the following quadratic equation:

$$A \cdot x^2 + B \cdot x + C = 0 \tag{4.4}$$

Where Tracker puts in the value for A, B and C, that makes the quadratic equation match the plotted curve best as can be seen in figure 4.5.

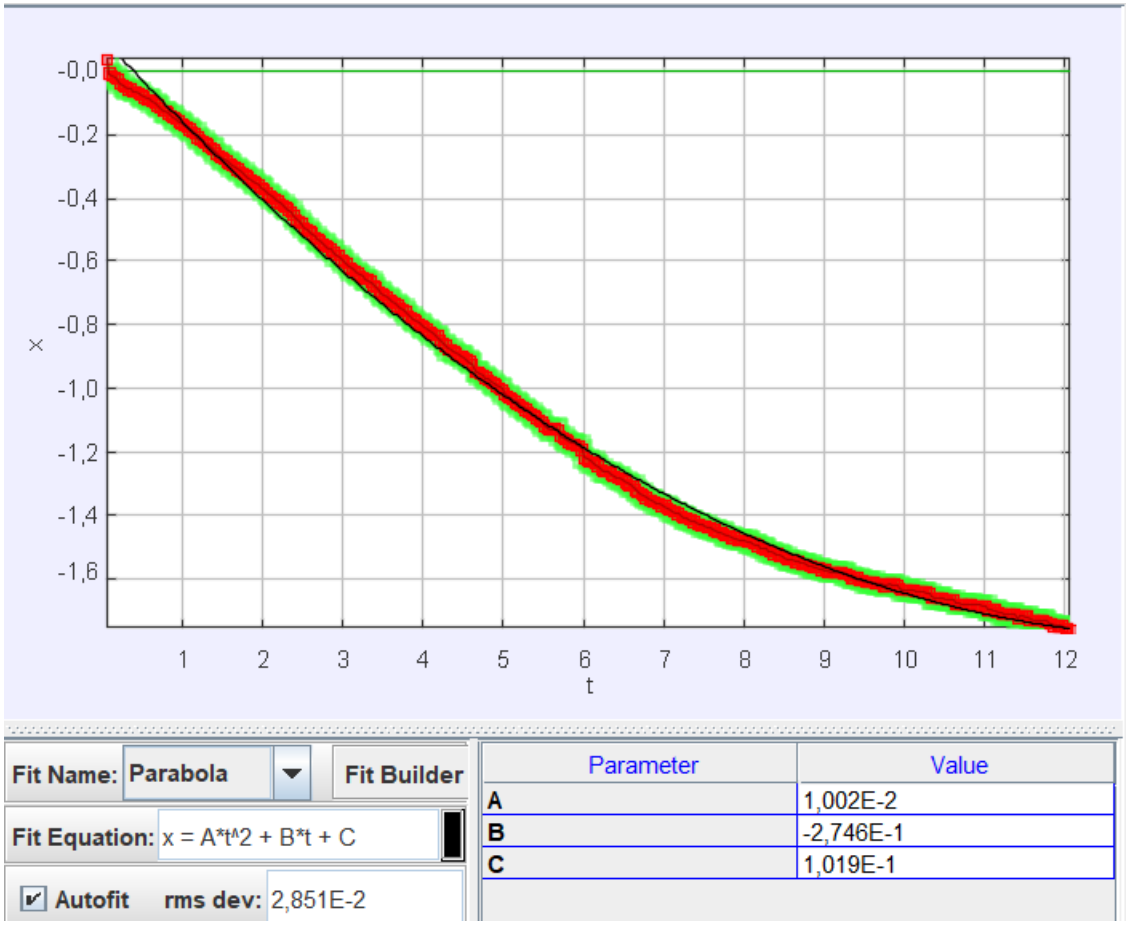


Figure 4.5: A representation of how Tracker tries to fit the curves

These values can further be used to calculate the acceleration and velocity for the vessel. Since, from kinematics, there is a quadratic equation for an object in motion:

$$\frac{1}{2} \cdot a \cdot t^2 + v \cdot t + s = 0 \tag{4.5}$$

Therefore the acceleration of the vessel can be found with:

$$a = 2 \cdot A \tag{4.6}$$

Where:

a - Acceleration

A - Computed value from Tracker

And the velocity:

$$v = B \tag{4.7}$$

Where:

v - Velocity

B - Computed value from Tracker

4.7 System Identification

4.7.1 Modeling

As described in section 2.10 a ship can be modeled by the equation 2.23. The system modeling was implemented in Matlab, and in order to describe the model it needs to be on the form described by 2.8 and 2.9, repeated in 4.8 and 4.9

$$\dot{x} = Ax + Bu \tag{4.8}$$

$$y = Cx + Du \tag{4.9}$$

Rearranging equation 2.23 to this form results in the model equation:

$$\dot{\nu} = -D \cdot M^{-1}\nu + M^{-1}\tau \tag{4.10}$$

where τ denotes all terms on the right hand side of equation 2.23 as these are the force inputs on the system.

Substituting the matrixes in the terms of 4.10 with A and B gives the following.

$$A = -D \cdot M^{-1} = \begin{bmatrix} A11 & 0 & 0 \\ 0 & A22 & A23 \\ 0 & A32 & A33 \end{bmatrix} \tag{4.11}$$

$$B = M^{-1} = \begin{bmatrix} B11 & 0 & 0 \\ 0 & B22 & B23 \\ 0 & B32 & B33 \end{bmatrix} \tag{4.12}$$

Note that the shape of the final A and B matrix is the same as the original D and M matrixes. The model can now be arranged in Matlab for use in system identification with GA. The states of this system are given as v, see table 2.1 for a description of these states.

4.7.2 GA

Using equation 4.8 and 4.9, a software like Matlab can simulate the response of this system for a given input. By then defining a suitable cost function, it may be possible to make the GA search for a system model by simulating different combinations of parameters for the A and B matrixes.

Defining the GA there are several choices for pairing, mating, and mutation. Examples of different methods are described in section 2.19 where a possible combination could be a GA with random pairing, single point crossover mating and random mutations. The population size must also be specified, as well as the number of genes in each individual. There is also a choice between a continuous or binary GA, which are also described in section 2.19. To identify a system model by the method outlined above, a continuous GA could be implemented as it can work with “Real” values without complex encoding and decoding.

As for the GA parameters, this specific implementation is based on an algorithm published by Haupt & Haupt (2003) [32], and utilizes single point crossover mating, random pairing and a weighted random chance of mutation. The mutation rate of this implementation is set to 0.2, or 20%. This means that 20% of the total number of genes will be mutated on each iteration. Which genes are mutated is selected randomly. The reason for setting this mutation rate so high is that

the single point crossover mating method does not add new information to the population, and thus all new gene information must come from random mutations. The population size was set to 12, meaning that 12 solutions would be evaluated simultaneously.

In order to reduce the search space of the GA, it is possible to calculate the M_{RB} matrix with measurements and estimates, as well as setting the M_A matrix to zero, as this would have no impact on the shape of the final M matrix. This means that the values used in the D matrix does not represent the true values of damping, as they will have different values depending on which numbers were actually located in M_A. The individuals of the GA population can then be set up with 6 genes. These genes could correspond to the 5 coefficients of the D matrix as well as one of the parameters of the M_{RB} matrix.

The I_z term of the M_{RB} matrix was estimated using the formula for inertia of a solid cuboid.

$$I_z = \frac{1}{12} m(w^2 + l^2) \tag{4.13}$$

Where:
m - Mass
w - width of the cuboid
l - length of the cuboid

The prototype is not a cuboid, so the sixth GA gene can also be incorporated to fine tune the value of I_z by modifying equation 4.13 to equation 4.14, where G_6 represents the sixth gene from the GA.

$$I_z = \frac{1}{G_6} m(w^2 + l^2) \tag{4.14}$$

Instead of letting the GA have genes for all 18 variables contained in two 3x3 matrixes, the GA now only has 6 variables to search for possible solutions.

4.7.3 Cost Function

A suitable cost function for system identification could be a function that constructs a state space model with parameter values given by the GA individuals in its population. The state space representation takes the from $\dot{x} = Ax + Bu$, where A and B are calculated as described in equation 4.11 and 4.12 and the cost function can replace the values of D and M_{RB} by the genes contained in an individual. This system can then be simulated for a specific input. The state response of this simulation can be matched to a measured state response for that particular input.

The cost function used to identify the system model, was implemented by designing a cost function which produces the system model described in 4.10 with parameters given from GA genes. The function also takes in a set of estimated velocity responses and input values measured and recorded on earlier test runs. The system models are then simulated with the recorded input, and the simulated state response is compared to the estimated state response for the same input. The cost of each solution is calculated as the discrete area between the measured state response and the simulated state response curves. It is assumed that the testing was done without input from wind and waves, meaning that the only input to the system is the recorded force applied from the motors.

4.7.4 Data Acquisition

To acquire suitable data for system identification it is necessary to perform a controlled test on the physical platform and recording the system's response for that input as well as recording the input itself. As the system model described by 4.10 is a multiple input, multiple output (MIMO) system, the input sequence should be carefully planned to test the effects of individual inputs on different outputs. A possible test sequence could be to perform a step on each input one by one, waiting for the system to reach a steady state before letting the input and response go to zero.

Depending on how the data is recorded some manipulation might be required, such as making sure all measurements are done according to a common reference, as well as synchronizing

timestamps for the different datapoints.

The effects of external forces like wind and waves affecting the ship should also be taken into account. If there are no ways to measure the wind and waves with on board sensors, a drift test where the system is given no input from the motors could be performed in advance. This test could record how much the platform moved due to other forces as no motor input is given. This data could be used for calculating the wind and wave forces, or it is possible to simply wait for the drift response to become negligible before running the actual data acquisition test.

4.7.5 State Estimation

The recorded data for this project is given as the vessel's local position in a map, on the form "Position X, Position Y, Angle (Yaw relative to x-axis in the map)" and will be referred to as "local position vector". These measurements need to be altered in order to estimate states (ν) for the vessel as these states are surge, sway and yaw rate as seen from the body frame (vessel). Continuing forward, the states will be referred to as "body velocities". In order to estimate the body velocities the measured position needs to be converted from the map reference frame to the body reference frame. This was done by describing all local position vectors as separate coordinate systems (The coordinate system for the vessel at that given point in time).

The first step was to translate and rotate all sampled local position vectors such that the first sample is located at (0, 0) and has an angle of 0 (aligned with the x-axis). By doing this the x-axis represents "forward" as seen from the vessel at the first sample.

The second step of the state estimation is to realign all the coordinate systems such that they all have the same orientation as the first sample. This will make the x-axis represent the vessel's forward translation. It is important to note that this has to be done in a very specific way. The samples need to be iterated through chronologically. Each subsequent sample has to be rotated around the sample currently under consideration, where the rotation angle is the negative of the current sample's orientation / angle relative to the x-axis. This will move the subsequent points in the xy grid and also change the orientation of these points. When this iteration is complete the

coordinate systems should all have their axes oriented the same way. The x-axis now represents the forward translation of the vessel, while the y-axis represents the lateral translation of the vessel. This can now be used in order to determine the states of the vessel by calculating the velocity from translation.

4.8 System Design

To create a complex system to control and navigate a robot a base system is needed to make it cohesive. Such a system should give the possibility of easy communication between modules and supply easy-to-use modules.

ROS as described in section 2.17 is such a system. It has packages that supply basic functionality one would want in a robot, and enables the possibility of fusing these pre-made packages with self-made packages.

Using a system such as ROS makes it very easy to create new packages to communicate with the pre-made packages, as well as communicate between them. Because ROS has such a well-documented wiki-page, a lot of problems that might come forth will already be solved by other people as well.

4.9 Sensors

To be able to get the orientation of the prototype, data from an IMU needed to be extracted. Further more the data needed to be filtered because of problems with noise as discussed in section 3.2. All the data (Gyroscope, Accelerometer & Magnetometer) needed to be combined in a filter to extract the orientation in quaternion.

This is because this is the most optimal way to represent angles in 3D-space as discussed in section 2.4.

Data from the lidar also needs to be extracted to be used with the mapping and localization-algorithms. As most lidars come with standard libraries for communication this is mostly plug-and-play into a microcontroller.

4.10 Motor Controller

To control the motors a form of feedback controller is needed. One of the most popular feedback controllers is the PID-controller as described in section 2.11. It allows for a system to keep a constant velocity as it compensates for enviromental factors like wind because it looks at the feedback from the system itself.

A Kalman filter can be used for the feedback loop of the controller. The Kalman filter uses a model of the system in order to predict the system states, as well as using measurements from sensors, as described in 2.12. Implementing this in a feedback loop can compensate for noisy sensor readings if the model used for prediction is accurate enough.

4.11 Mapping

To produce a map for the surrounding area where the prototype will navigate, there exists a package in ROS called hector_mapping. This is a package that utilizes the SLAM-approach to create a map and simultaneously localize itself as discussed in section 2.15.

This package can be used without any form of odometry, but is made significantly better by supplying odometry information in the form of orientation.

There are a lot of parameters to fine tune for this package to make the mapping stable. This has to do with the update-frequency of the mapping as well as data about the lidar used for the mapping, and its limitations.

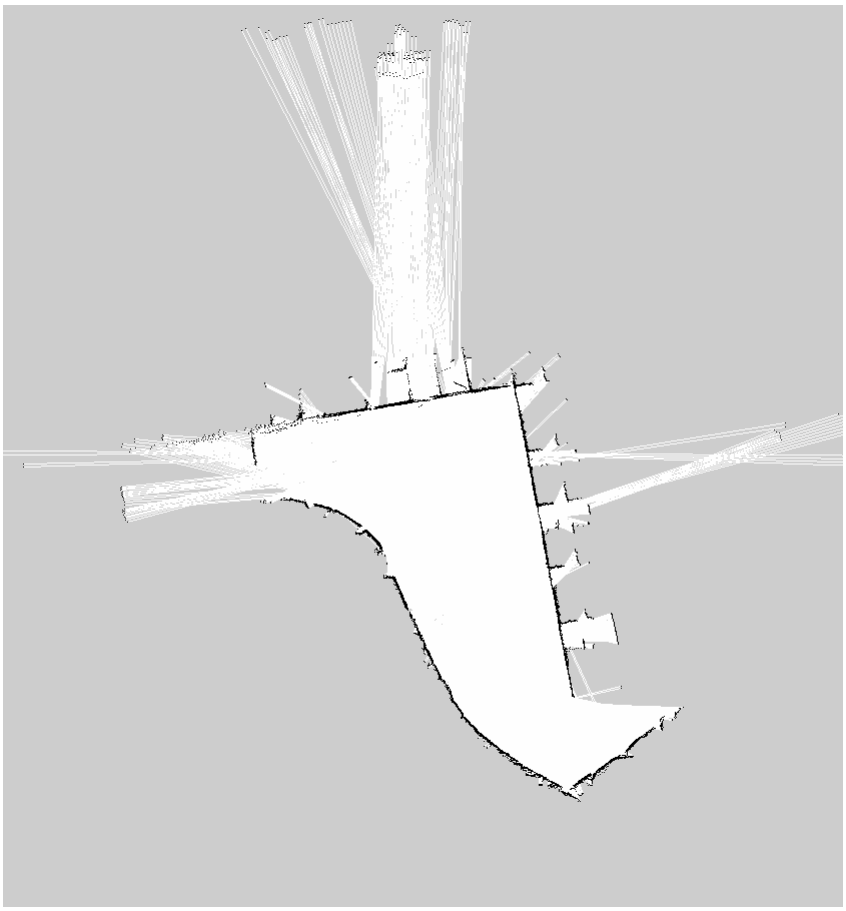


Figure 4.6: A map created by hector_mapping

The mapping-algorithm creates a map and marks what is free space and what is obstacles in the form of some kind of surface detected. Such a map is shown in figure 4.6. In this map white is detected free-space, black is detected obstacles and gray is unmapped area.

4.12 Localization

To localize a robot in a given map, there are several possibilities. In ROS hector_mapping does the mapping as well as localization at once using the SLAM-approach as discussed in the section above.

One other approach to localization is the AMCL-method. This method requires a static map and an initial pose to be able to localize itself within the map. This method works on probability,

it creates a set of possible positions and from the information given gives the most likely position of the robot. To increase this certainty information such as translation as well as orientation is required.

As seen in figure 4.7 AMCL is uncertain in its position, as represented by the large purple bubble and its orientation as represented by the yellow cone.

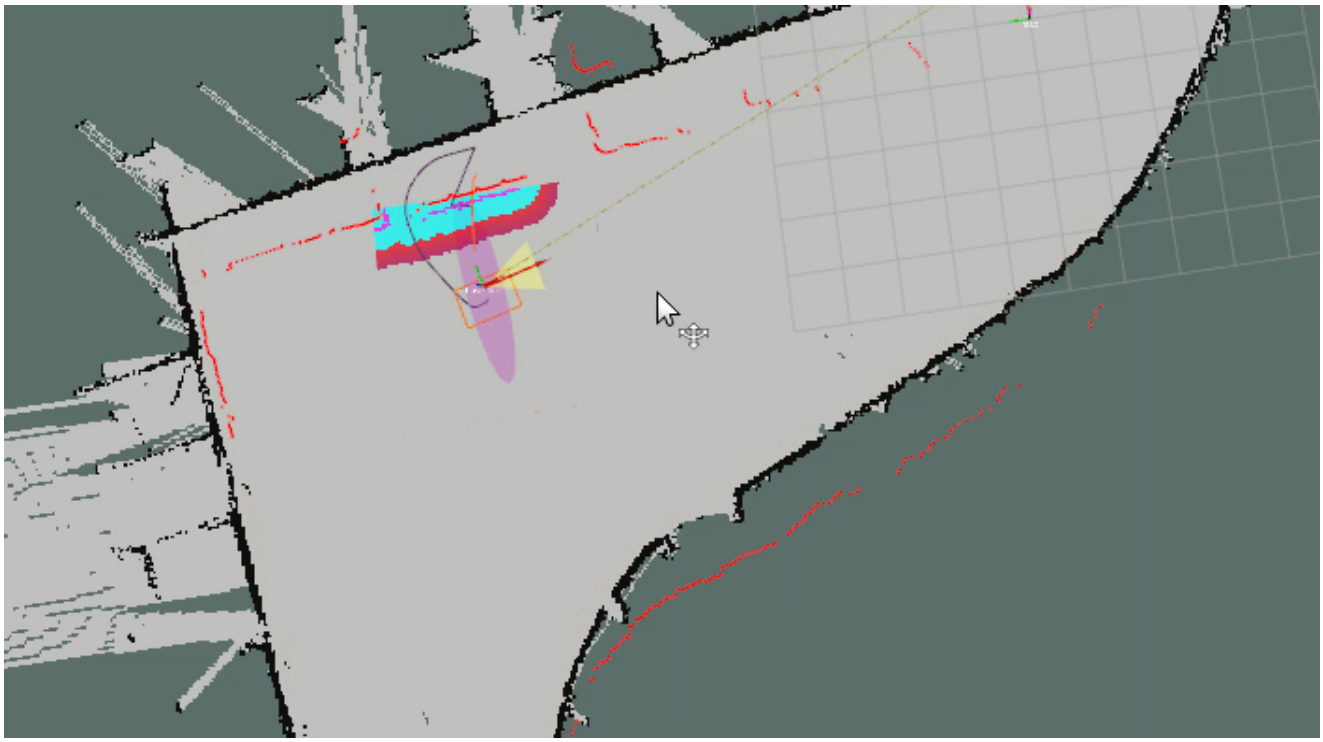
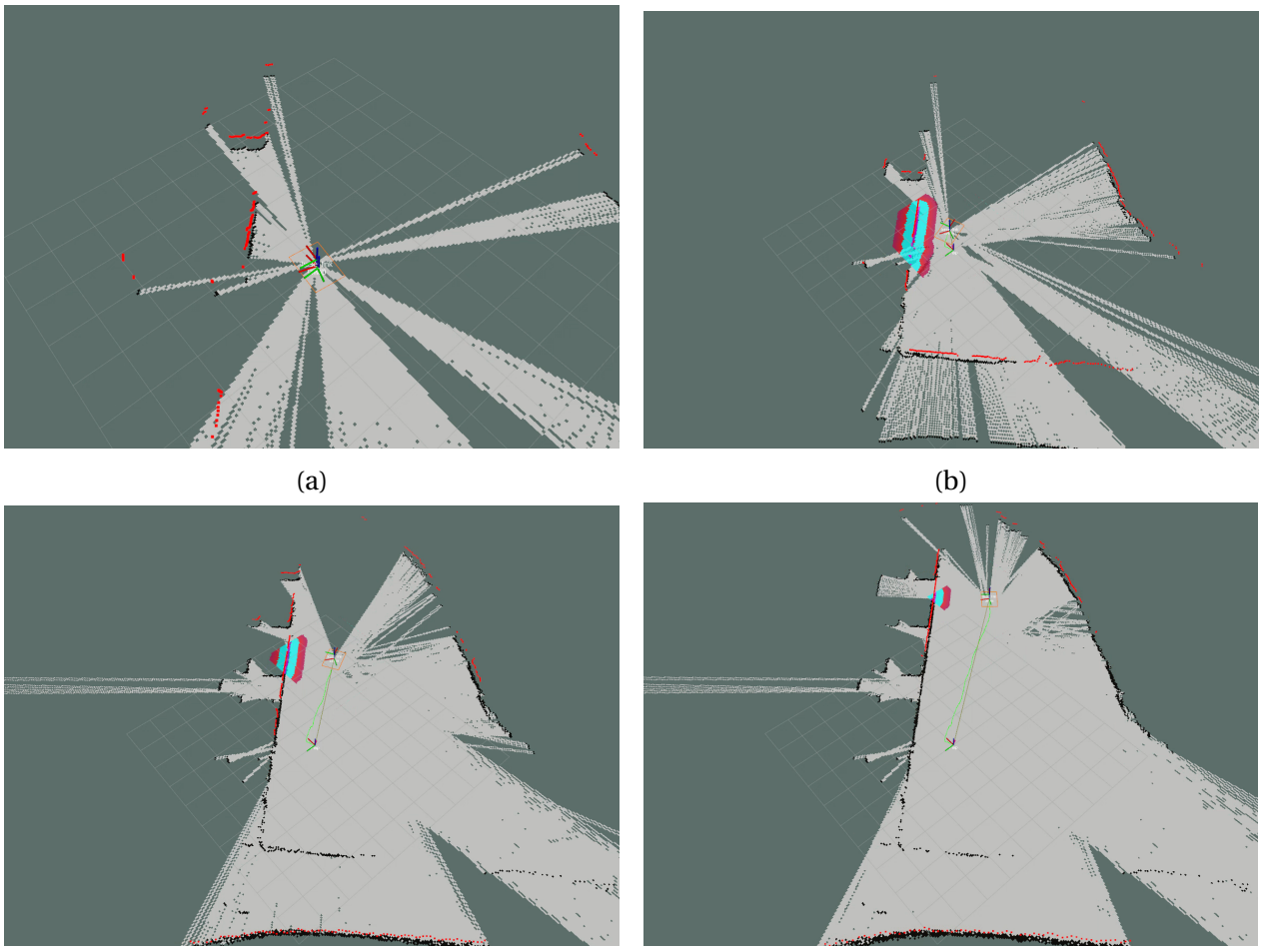


Figure 4.7: AMCL in Static Map

The big difference between these two localization methods is that SLAM makes and updates its own map as it goes as shown in figure 4.8, where AMCL uses a static map to localize itself inside.



(a) (b)

(c) (d)

Figure 4.8: Mapping and localizing in unknown-area using SLAM

A positive side to SLAM is that unmapped area can be navigated as the map updates itself, but the drawback is that since every time the algorithm is started, the coordinates for any given position will change with what the start position was when mapping was begun.

AMCL can not navigate unmapped area, but since the map is static it has the possibility of navigating to the same position every time a given coordinate is sent to it.

4.13 Navigation

In ROS there are several different libraries to enable navigation through areas that are mapped. These libraries all excel at different things, some being better than others at specific things, but lacking at other aspects.

The most popular one used that comes installed with the package move_base is dwa_local_planner. This planner has the possibility of enabling a mode for omni-directional robots, and has obstacle avoidance built into it.

Another popular one is teb_local_planner often used for omni-directional robots, as it often uses all directions of the robot more. It has most of the same functionality as DWA in that it also has obstacle avoidance.

These planners make it possible to create a path for a robot, and they have their own set of parameters that can be adjusted to compensate for what kind of robot needs navigation, as well as what kind of enviroment the navigation is to happen in.

The obstacle avoidance of these local planners come in the form of reading from the local costmaps to identify and avoid obstacles that pose a threat to the robot itself as described in section 2.18.2 and shown in figure 4.9.

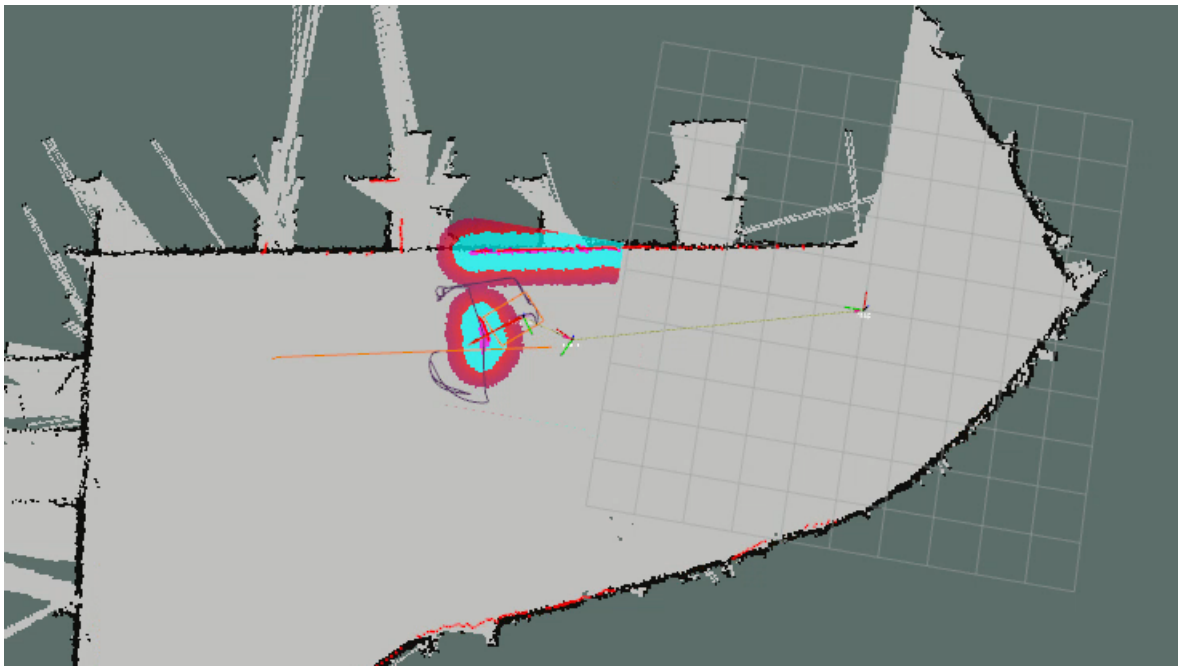


Figure 4.9: Unmapped obstacle detected right in front of the robot

The navigation stack sends out velocity commands to the robots control system, that says how it should move to keep on the path given by the local planner.

4.14 GUI

The construction of a Graphical User Interface introduces the possibility of multiple methods. These methods can for example be a web-based or a program-based GUI. In a web-based GUI the GUI utilizes a web browser as its engine. This makes it possible to use HTML, JavaScript as well as CSS to create the pages, and then use Python for the back end to operate the GUI. To build such a GUI requires the use of a web framework, these frameworks can for example be Flask or Django.

Whether its a web-based or program-based GUI, they still follow the same principles of what make a good GUI. An important part of the construction is the layout itself. It's easy to create a cluttered GUI by including all possible data without thinking about the layout or placement within the GUI itself. The layout of the GUI should be intuitive and easy to understand.

To organize a GUI it's important to showcase the most necessary information to the user in a convenient way. A method for doing this is to place information that belong together close to each other. By doing this one creates a system that is easy to follow and makes it more user friendly.

Color selection is also important when creating and organizing the GUI, as it can be used to enhance the user experience. It might be smart to use a consistent color theme throughout the GUI and stay consistent. One can for example use complementary colors, or for example contrasting colors when one wants the users attention. This can for example be red on a gray background to display a warning.

A web-based GUI opens up for a more flexible platform. Since a web-based GUI can run from a single server, it can be accessed from any device that is capable of running a chromium-based web browser. By having a single server that runs the GUI, it makes it unnecessary to download and install the GUI on a new device. Avoiding this leads to a more flexible GUI that can easily be altered without causing problems on the user-side.

Chapter 5

Materials

This chapter will introduce you to what materials and components is used for the physical part of the project. For the electrical parts it will detail what their use is as well as what power consumption can be expected from them. For the construction itself it will detail shortly what their use is.

5.1 Electrical system

This prototype will consist of several electrical components to be able to build a system good enough to do what it needs.

5.1.1 Computer for Processing

The PC that will run most of the processing is a MSI laptop with an Intel i77820HK 4.2GHz processor, 16GB DDR4 RAM and a NVIDIA GTX 1080 graphics card. This laptop runs the newest Ubuntu distro directly (as in not via Virtual Machine).

5.1.2 Components

The components the prototype will consist of are:

- RPLidar A3

- Thrusters
- Gyroscope
- Accelerometer
- Magnetometer
- Raspberry Pi 4
- Arduino Nano
- Buck Converters
- Motor Controller
- Internet Router
- Battery
- Fuses
- Power Switch

Each of these components will have an impact on the amount of current required.

RPLidar A3

The Lidar which will be used is a RPLidar A3 shown in figure 5.1. It has a 25 meter radius, where it can take up to 16000 samples per second and rotate with a frequency between 15 and 20 Hz [54].

Figure 5.1: Rplidar A3 [4]

Its current consumption is impacted by several factors, such as what mode it is in, how fast does it rotate and what kind of resolution is in use. It requires a supply voltage between 4.9V and 5.5V [54], and it will be supplied with 5V. During startup the current consumption can peak at around 2500mA because it needs to charge the input capacitor. After startup the current consumption will be around 600mA [54]. Therefore the current consumption will be calculated as 600mA.

Thrusters

Thrusters are required to move the prototype through the water. The type being used is Haswing Osapian 20lb, a simple electrical trolley motor shown in figure 5.2.

Figure 5.2: Haswing Osapian 20lb

This is a motor that requires 12VDC to work properly, and has a current consumption of around 14A to 17A from what documentation is available through one of the websites selling the engine [20].

Gyroscope

The Gyroscope in use for this project is a part of a bigger IMU that has several different modules, the IMU itself is called "Adafruit 10-DOF IMU Breakout". The Gyroscope itself is of the type L3GD20. This is a 3-axis gyroscope which gives its measurements in radians per second [56].

The gyroscope requires between 2.4VDC and 3.6VDC to work properly, and will draw upto 6.1mA [56]. With this low current the amount is negligable.

Accelerometer

An accelerometer is also used to be able to say something about the prototypes acceleration in its movements. The accelerometer used is a part of the same IMU as the gyroscope (The Adafruit 10-DOF IMU Breakout) and is of the type LSM303DLHC.

This accelerometer requires between 2.16V and 3.6V to work properly, and draws upto 110 µA [55]. With this low of a draw, the current is negligable.

Magnetometer

A magnetometer is used to combine with other sensors to monitor the orientation of the prototype. This magnetometer is a part of the accelerometer mentioned over, and is therefore of the type LS303DLHC.

The voltage and current draw, is therefore the same as the accelerometer.

Raspberry Pi 4

To be able to control the system of the prototype a Raspberry Pi 4 will be used (shown in figure 5.3). This is an open-source single board computer that is easy to use. With access to WiFi, bluetooth and ethernet there are several possible ways to communicate with it, which makes it quite practical in several cases.

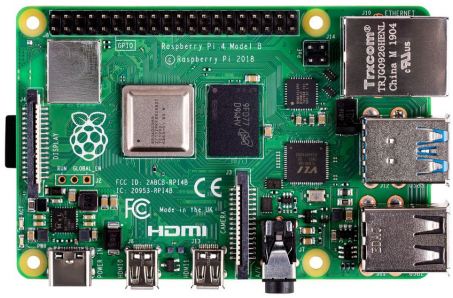


Figure 5.3: Raspberry Pi 4 [19]

It has a quad core Cortex-A72 (ARM v8) 64bit SoC-processor which runs at 1.5GHz and has

4GB RAM [26]. It requires 5V to be able to run properly, and will draw between 2.5A and 3A depending on what components are connected to it [26]. From experiments found online, it will draw around 575mA [2] when idle.

Arduino Nano

An Arduino Nano as shown in figure 5.4 will be used to read the data from the IMU. This is because of the availability of standard libraries for Arduino from Adafruit. It requires 5V to run properly, and has a current draw of 19mA without any external components connected to it [5].

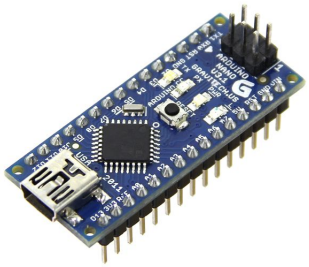


Figure 5.4: Arduino Nano [57]

Buck Converters

There is a need to regulate the voltage levels to the different modules, to do this Buck Converters will be used.

The ones chosen has an input voltage from 5V to 40V, and output voltages of 1.2V to 36V [50]. By using this it is possible to draw upto 12A, but it is reccommended to not exceed a current draw of 8A [50]. From earlier experiences it will be ensured that the current draw will be kept under 8A, because the credibility of ebay-documentation is not always high.

The documentation says that the efficiency rating of this component is up to 95%, but earlier experiences are that they mostly fall around 90% efficiency.

Motor Controllers

The motor controller used for the thrusters is of the type RoboClaw as shown in figure 5.5. This is a 2 output motor controller capable of controlling 2 DC Motors of up to 30A each. The Roboclaw has a few different communication interfaces like PWM, Serial as well as USB. [6]

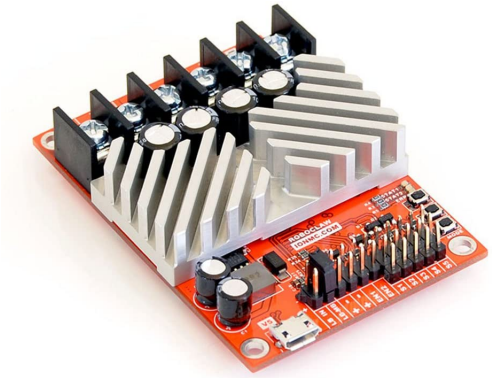


Figure 5.5: Roboclaw [16]

In this project the USB-interface was chosen because of its ease of use and the ability to read data from the roboclaw like battery voltage.

Battery

The type chosen is a battery that has the battery acid bound in a slow-moving gel [11]. Since the battery acid has been bound in this gel, the battery can be mounted at an angle of 90° without any chance of any leakages. This also gives the advantage of it not producing any gases during charging, which makes it possible to encapsulate it if need be [11].

5.2 Construction

The choice of material for the prototype was heavily affected by the price of the different materials. Therefore plastics and simple waste pipes were chosen, since these are components that are easily available in any hardware store.

These parts were supplemented by modeling and using 3D-printed parts to mount some of these parts together. This was all done to try to keep the price as well as the weight of the

construction low.

5.2.1 Parts

- Waste tubes
- Aluminium Profiling
- 3D-Printed Parts
- Plexi Glass
- Scews and nuts + washers
- Ballast
- Junction Boxes

Waste Tubes

The tubing for the prototype is waste tubes gotten from a regular hardware store. These tubes are robust, cheap as well as light.

Aluminium Square Profiling

The profiling used to stabilize the prototypes base plate as well as between the vertical tubes is aluminium profiling of the size 20mm x 20mm.

3D-Printed Parts

Where standard parts are not possible to get, the plan is to supplement with 3D-printed parts. This includes parts such as corner construction for the base plate, as well as the link-lock system between the vertical piping. The .stl-files for these parts are in the appendixes.

Plexi Glass

The base plate for the top as well as the bottom is made by plexi glass. These come in a set size of 0.7m x 1m and are easy to modify by using a laser cutter. The plate is modified to fit with the vertical tubes as well as holes to mount everything together with ease.

Screws, nuts and Washers

All screws, nuts and washers are stainless so as to not rust after quite some time in the water. This is to ensure the integrity of the construction over time.

Ballast

The extra ballast needed to make the prototype sit lower in the water is sand put into small bags. This makes it easy to reduce as well as increase the weight of the prototype as need be.

Junction Boxes

Three different junction boxes are used on the prototype. One of the types is a IP67 junction box [12]. The other two types are IP66 junction boxes of different sizes [9] [10]

Chapter 6

Testing

This chapter will introduce you to all the tests that are going to be run for the project. These tests include both software tests as well as physical tests for the prototype. For these tests the purpose will be detailed, as well as the test setup and why these tests are interesting to run.

6.1 Design

The design was tested during development and inherited a lot of its tests from the early design reviews described in Chapter 3. These tests were general early tests to see if the design was a viable design that could be used for the project.

6.1.1 Buoyancy Testing

The buoyancy of the prototype was tested by releasing it slowly into water at the SIT boathouse and observing how buoyant it was.

By adding extra weight in the form of bags of sand, the prototype was tested for buoyancy with a simulation of the weight that would be added by electrical components if the prototype passed the test.

6.1.2 Stability Testing

To get a rough estimate of the stability the platform was tested by pushing it around in the water so its behavior could be observed. The platform was pushed to apply roll, pitch and yaw-force independently and the time it took to stabilize itself was observed.

6.1.3 Waterproof Testing

For the boat itself, the waterproof testing was done while testing the buoyancy. Because of no expensive equipment being mounted to the boat initially, it was safe to test how waterproof the different aspects of the tubing was.

The electrical box for the battery was waterproof-tested at several stages. At first submerged in a sink filled with water for some hours without any modification done to it. Then modified with sealant around all edges and possible leakage-points and tested with battery and oil on land.

As a final test it was submerged in water mounted to the boat when starting to test the rest of the system.

6.1.4 Weight Distribution Testing

During the weight distribution testing the prototype was tested to see what kind of impact weight on the top plate would have for the angling of the platform. It was tested using small 1kg bags of sand to simulate component placement.

6.2 Initial System Tests

After mounting all of the electrical components on the prototype, several tests were run while in the garage before putting the prototype into water.

A test script was made to be able to extract data from all data sources (IMU, motor-signals, Lidar) as well as being able to control the motors by using a wired Xbox-controller. This script was ran while the prototype was in the garage, and everything was tested properly before it was sent out on the water.

This was to ensure that nothing would go wrong while on the water, as the damage that could come from a system failure while on water could be catastrophic for the prototype and thesis as a whole.

Test Setup

For the test the thrusters were just mounted on a work bench with enough space to ensure the propellers did not collide. The electrical system was on top of the work bench, connected to everything, and the battery was on the floor in its electrical box. Figure 6.1 shows the setup for this test.

Figure 6.1: Setup for Initial System Test

In this test the manual control and responsiveness of the thrusters were tested. The test also was to see that every system communicated like it should, and that the thrusters would shut off if the PC was ever disconnected from the Raspberry Pi.

6.3 Initial Tests on Water

After ensuring that launching the prototype into water would be safe, several tests were ran initially. The initial tests were used to see how the prototype would handle in water, how the motors would make the prototype tilt during acceleration, deceleration and constant speed, etc.

One of the things that were to be checked as well was how waterproof the casing for the battery was at the depth it was at.

Test Setup

For the test the prototype was slowly submerged into the water to float for itself for a little time to see that it was stable and fine. Then each direction that was possible to run in was ran manually with the manual control.

The maximum amount of power to the motor controller was found during this test, by testing how much tilt the motors applied to the prototype during the different amount of power supplied to the motors.

For how waterproof the casing for the battery was it was just observed as the prototype handled out in the water.

All of these tests were done with one member of the group in the water along side the prototype to ensure that nothing could go terribly wrong during these early tests.

6.4 System Responses

When all small errors were fixed, and the prototype was safe to put on water for a longer amount of time, general system responses were tested with every electrical component mounted to it. A series of tests were planned out and described neatly before they were done, so there was a good plan for how to do them. The tests were as follows;

6.4.1 Bollard Pull

The Bollard Pull is a test where one checks how much pull force one has from the motors. This was an important test to do to be able to map out the power curve of the engines mounted to the prototype.

The test is run by mounting a rope to the prototype, and linking this up to a spring-based weight to be able to measure how many kg the prototype pulls at certain motor levels.

6.4.2 Rotation Test

A simple rotation test was needed to be able to map out what kind of speed and acceleration the prototype had during the rotation. The test was run by placing the vessel in open water and applying maximum rotation velocity for two full rotations. The sensor data for this run was recorded and used to calculate the rotation velocity.

6.4.3 Speed test

In the same vein as the rotation test, a test for forward and lateral speed and acceleration was needed to be able to accurately map out the limits of the prototype forwards and sideways. This test was run in three ways;

- Running max speed for a set length 3 times, taking the time manually
- Starting the motors, reaching max speed and then letting it die down by itself, logging all info from IMU's and sensors
- Recording video of prototype running forwards, analyzing data via Tracker-software

6.5 Testing of Navigation Stack

With the prototype the different local planners were tested to see which of them would work best for the intended goals. In these tests things like the usage of the omni-directional possibilities were checked, as well as the obstacle avoidance and how accurate the local planner was.

Test Setup

For these tests of the navigation stack the prototype was set into water and sent different coordinates either via RVIZ using the send 2D-coordinate-button or by sending them via a script created to send a coordinate + an orientation.

When sending these coordinates or poses it was observed how the navigation stack wanted the prototype to achieve getting to this goal. Specifically to ensure that the lateral motion-

possibilities of the prototype were made use of, it was tested how willingly the navigation stack used this.

The obstacle avoidance of the navigation stacks were also tested by having a person in the water creating a dynamic obstacle and seeing how the navigation stack handled these challenges of unmapped obstacles as seen in figure 6.2.

Figure 6.2: Testing of obstacle avoidance in Navigation Stack

6.6 Performance Tests

When everything was running smoothly, performance tests were to be run. These tests would show how well the prototype would do the tasks it was meant to do. These tests are mainly performed in the area seen in figure 6.3.

(a) View 1

(b) View 2

Figure 6.3: Overview of main test area

6.6.1 Pathfollowing Test

This test was designed in order to determine how well the prototype would follow a planned path and how it would maneuver in order to reach its goal. It was done in three different test configurations.

First the prototype was given a straight line path in the same direction that the platform was facing. These were the ideal conditions as all the platform had to do was to drive in a straight line.

Second, the prototype was given a path opposite to the direction it was facing. This meant that the platform had to change its orientation before heading towards the goal.

Third, the prototype was given a path in the same direction as the platform's heading, but the goal orientation was rotated 180 degrees from the direction of travel.

6.6.2 Drift Test

This test was designed to examine how far the prototype would drift due to wind, waves and current when no inputs were given. The prototype was placed in the middle of the test area and

then no inputs were given for 50 seconds.

6.6.3 Obstacle Avoidance

This test was designed to examine how well the prototype could handle unmapped objects located along its path to the goal destination. This test was done in two stages.

The first test was done with an obstacle placed in the middle of the planned path of the prototype. The second test was configured the same way, but the obstacle was offset slightly to one side of the planned path. When the obstacle was placed the platform was allowed to attempt to navigate to its goal.

6.6.4 Docking Test

This test was designed to test how the platform handled the docking sequence and which maneuvers were chosen in order to get to the end destination and was performed in two parts. First the prototype would go to a pre-dock position and orientation. When arriving at this pre-dock destination, the platform would get a new destination which was the final docking position.

6.6.5 Open Loop Observer Tests

This test was performed to see the difference in performance when using a kalman filter with sensor feedback versus a pure model prediction feedback. To compare these, the previously defined path following and drift tests were performed.

6.6.6 Mapping and Localization in Bigger Enviroment

Specifically for testing mapping and localizing in bigger enviroment, another part of the marina was used as seen in figure 6.4.

(a) View 1

(b) View 2

(c) View 3 (Prototype circled)

(d) View 4

Figure 6.4: Overview of test area for mapping and localizing in bigger enviroment

This test was to see how the system would cope with less reference points for the lidar and if it was possible to map and navigate such a big area.

Chapter 7

Result

During this chapter the final solution for the prototype and the result of all the tests done are presented. The chapter will chronologically go through the building of the prototype and why all the different choices are made, before going through the tests made after the prototype was finished and ready for testing.

7.1 Physical Design

7.1.1 Design Choice

The final design of the prototype ended up being quite a lot like the third iteration of the prototype and is shown in figure 7.1 There are three big differences:

- The Horizontal Tubes are removed
- A plate is mounted towards the bottom of the prototype to serve as a mounting piece for the battery
- The vertical tubes were made shorter

The plate towards the bottom serves as a mounting place for the battery, but also as a stabilizer for the vertical tubes to reduce the amount of warping that can occur during movement of the prototype.

Figure 7.1: Fourth iteration of the prototype

By mounting the battery towards the bottom of the prototype instead of the top, the amount of weight towards the top is drastically reduced, and should lower the centre of gravity and therefore make it more stable.

One of the other changes made is also that the vertical tubes were made shorter to get the top of the platform closer to the centre of rotation and reduce the moment arm created when weight is distributed on the top of the plate. This reduced the impact uneven weight-distribution had.

7.1.2 Dimensioning

Construction Components

In this category all components used to build the shell of the prototype will be included. This includes (but is not limited to) things like the aluminium frame, the corner constructions as well

as the plate. The weight and amount are shown in table 7.1.

Component	Weight (kg)	Amount	Total weight (kg)
Aluminium Frame	2.349	1	2.349
Plate	2.92	2	5.84
Tube (1m)	1.44	6	8.64
Corner constructions	0.065	6	0.26
Screw/Nut/Disc	0.009	30	0.270
Link / Locks	0.225	6	1.35
Stabilizer Bracket (Center)	0.019	5	0.095
Stabilizer Bracket (Ends)	0.048	8	0.384
Angle Brackets	0.016	8	0.128
Lidar Bracket	0.067	1	0.067
Total Weight:			**19.383kg**

Table 7.1: Overview of weight of construction

From table 7.1 one can see that the weight of the construction will be roughly 19.5kg.

Components for electrical system

The parts included in this category are components that will be used for the electrical system on the prototype. This includes (but is not limited to) the lidar, battery and the control system. Some of the components have their weight given in the documentation, some of the parts had to be weighed by using a kitchen scale. Weight and amount of the parts are shown in table 7.2.

Components	Weight (kg)	Amount	Total weight (kg)
Raspberry Pi 4	0.046 [31]	1	0.046
Arduino Nano	0.007 [5]	1	0.007
Battery	7.18 [11]	1	7.18

Buck / Boost Converters	0.100	2	0.100
Thrusters	2.0	4	8.0
Electrical Box	0.302	1	0.302
T350 El. Box	0.630 [10]	1	0.630
T250 El. Box	0.465 [9]	2	0.930
Gyro/Acc/Mag.	0.004	1	0.004
Lidar	0.19 [54]	1	0.19
Roboclaw	0.07	2	0.14
Internet Router	0.240 [17]	1	0.24
Total Weight:			**17.869kg**

Table 7.2: Overview of the weight of the electrical components

Here one can see that the total weight of the electrical components will be roughly 18kg.

Total weight

After summing up these weights, one can see that the total weight of the prototype will be roughly 37.5kg. This will then have to be used to dimension the tubing to see what kind of floating ability that is needed. It will also be necessary to calculate how much ballast will be in the horizontal tubes on the bottom of the construction.

7.1.3 Stability calculation script

A parametric script was designed in order to calculate the centre of gravity, centre of buoyancy and metacentre of the platform as described in section 4.5.5. The script is based on defining several "Component" objects which describe the component's weight, volume, location and whether it is unsubmerged, partially submerged or totally submerged. The component objects are assumed to have an even weight distribution and it's location is the component's centre of gravity described by a 3D vector in a 3D grid.

In order to calculate the centre of gravity the script iterates through all component objects and calculates a weighted sum of all position vectors by using equation 4.1, repeated in 7.1 for

ease of reading:

$$C_G = \frac{1}{\sum_{n=1}^{N}(m_n)} \cdot \sum_{n=1}^{N}(m_n \cdot v_n) \tag{7.1}$$

To find the centre of buoyancy the script uses the method outlined in section 4.5.5

In order to calculate the submersion distance of the platform more easily, the calculation makes a few assumptions. Firstly, every component object marked as totally submerged are assumed to be totally submerged, no matter the weight of the construction and their volumes are used to offset the total amount of volume necessary to displace in order to support the platform's weight. Additionally, every component marked as partially submerged are considered to be submerged at an equal rate, meaning the total volume to displace are shared between these components. This assumption was made since the partially submerged components were defined as the vertical pipes.

With these simplifications the script first calculates the volume to displace from the total mass of the construction. It then subtracts the volume of each component marked as totally submerged from this value. Next it finds the total volume of each component marked as partially submerged and calculates the submersion rate of these components as a percentage. This percentage is then scaled by a value (in this case the length of the vertical pipes), which gives how far these pipes must be submerged in order to support the platform's weight.

The centre of buoyancy is then calculated in the same way as the centre of gravity, but instead of looking at the positions of each individual component's centre of gravity, the script uses the totally and partially submerged components' centre of buoyancy, as described in equation 4.2, repeated in 7.2.

$$C_B = \frac{1}{\sum_{n=1}^{N}(V_n)} \cdot \sum_{n=1}^{N}(V_n \cdot v_n) \tag{7.2}$$

The platform's metacentric height is found by using equation 2.6.

In order to use this equation it is also necessary to calculate the I term. This term can in this

specific case be described using equation 4.3, modified to 7.3

$$I = N \cdot (I_o + A \cdot y^2) \tag{7.3}$$

where

$$I_o = \frac{\pi}{4} \cdot r^4 \tag{7.4}$$

$$A = \pi \cdot r^2 \tag{7.5}$$

I - Second moment of area.

N - Number of vertical poles.

y - The distance form the mid line of the platform to the center of each pole.

I_o - The second moment of area of a circle

r - Radius of vertical pipes

The results from these calculations are then displayed and can be used in order to determine stability.

7.1.4 Simulations

The numbers known were put into the stability calculator-script to see that this prototype was viable. The results of the script showed that this was fine, and the prototype would be stable.

It should be noted that this simulation is just a guide line, and is not an exact measurement of the prototype in any way. It shows the basic gist of it, and was evaluated to see if there is enough leeway in case there needs to be more components which adds weight to the prototype.

The figures 7.2, 7.3 and 7.4 show that with a weight of 40kg this iteration is stable with a margin big enough to handle more weight if that should happen. They are labeled "Front", "Side" and "Top" views. By this it means that the plot displays the location of the points as seen

from the front, side and top of the prototype respectively.

```
    Parameters:
    Platform width:          0.7 [m]
    Platform length:         1.0 [m]
    Pipe diameter:           0.11 [m]
    Vertical pipe length:    0.7 [m]
    Horizontal pipe length: 0 [m]
    Additional ballast:      10 [kg]
Centre of gravity:    [0.0, 0.0, -0.3996] (x, y, z)
Centre of buoyancy:   [0.0, 0.0, -0.47439] (x, y, z)
BM vector:            [0.0, 0.0, 0.18211] (x, y, z)
Total weight:         39.55799999999999 [kg]
Submerged volume:     0.038593170731707316 [m³]
Submersion / rate:    0.5028118720162471 [m] / 0.7183026743089245 [ratio]
```

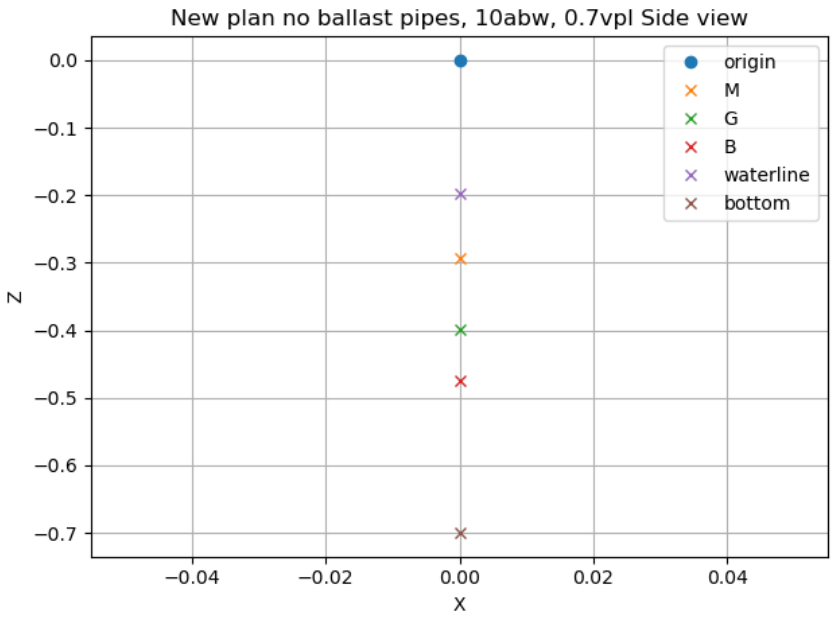


Figure 7.2: Calculations of Final Prototype, Side View

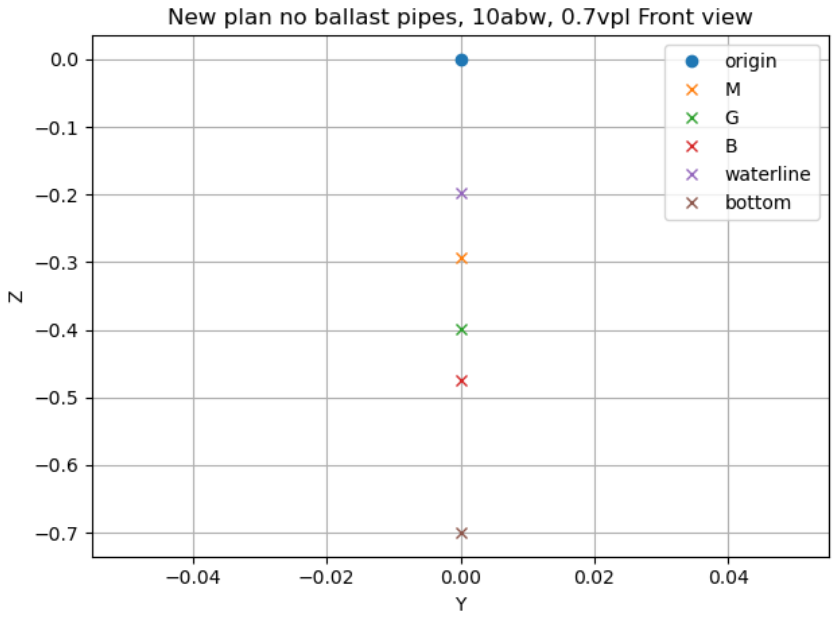


Figure 7.3: Calculations of Final Prototype, Front View

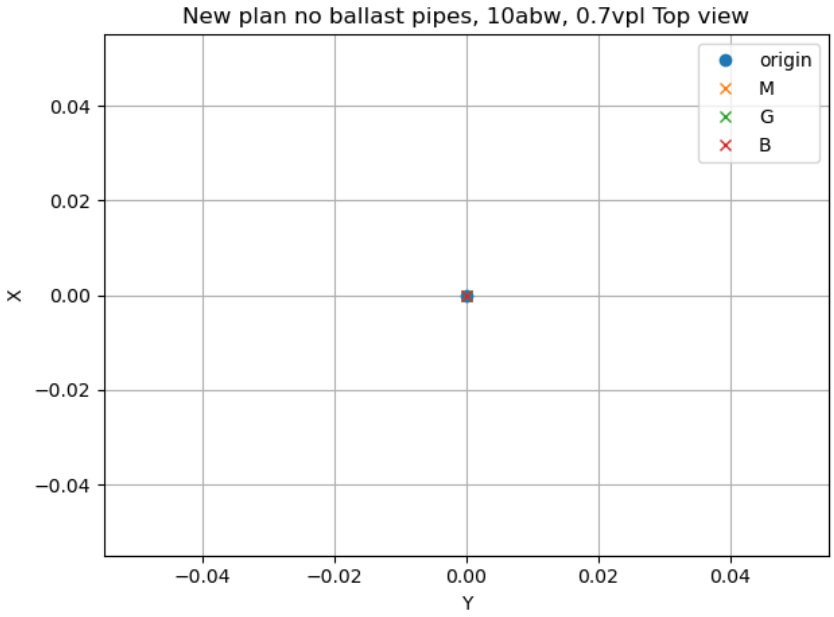


Figure 7.4: Calculations of Final Prototype, Top View

7.1.5 Motor Placement

The placement of the motors ended up being two for forward thrust and two for turning / lateral shifting. The placements and thrust directions are shown in figure 7.5.

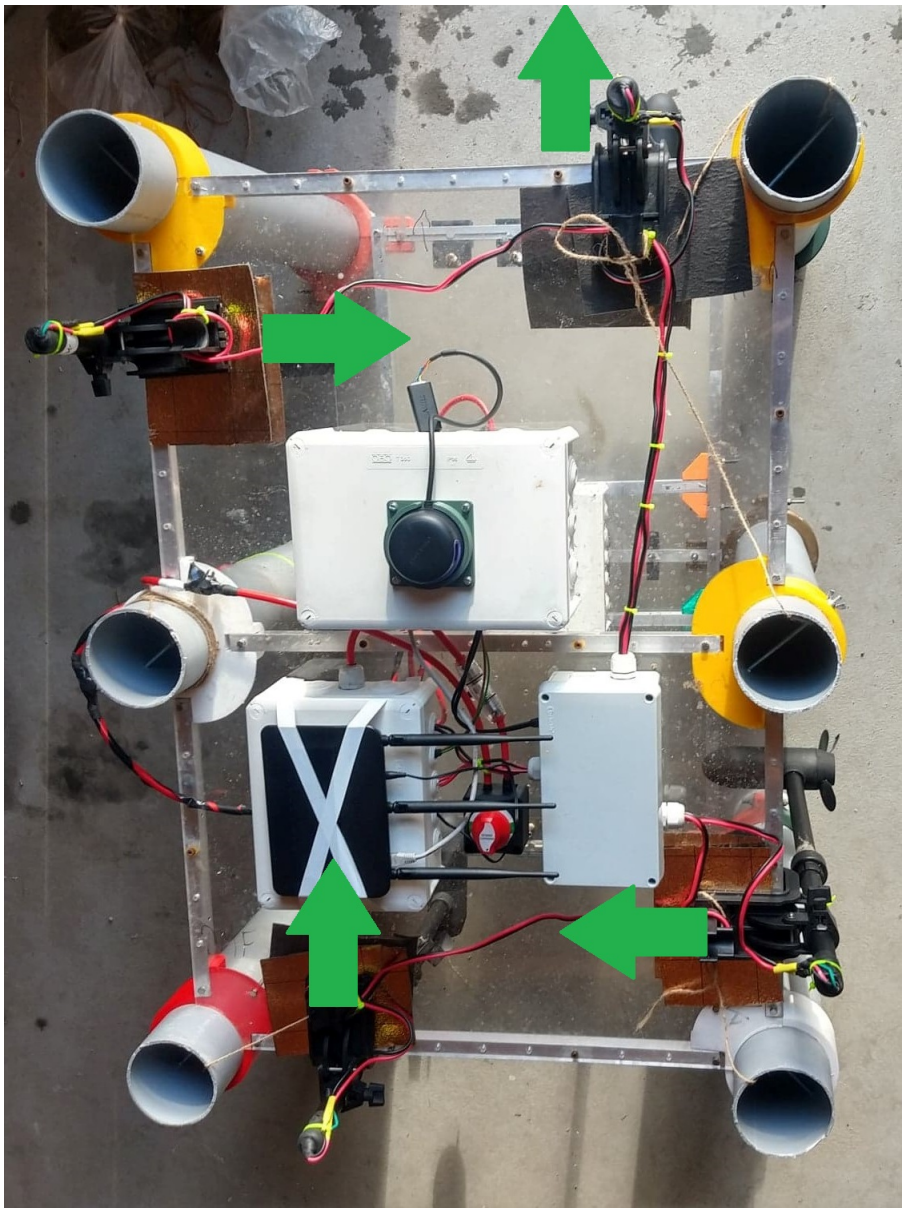


Figure 7.5: View of motor placement and thrust direction on prototype

The motors were mounted statically instead of placing them on rotating surfaces to reduce the complexity of the system to control the motors. Doing this reduced the chance of something going wrong and made it easier to control.

The forward motors were placed diagonal to each other to reduce the amount of pitch motion produced from running forward or backwards, compared to both motors being placed together on one side. The broadside motors enabled the possibility of turning in place, as well as

give the possibility of shifting laterally.

7.2 Electrical Design

Components	**Current Draw (mA) Max / Idle**	**Amount**	**Summed Current (mA)**
Raspberry Pi 4	3000 / 575	1	3000 / 575
Arduino Nano	19	1	19
Thruster	17000 / 0	4	68000 / 0
Gyroscope	6.1	1	6.1
Accelerometer / Magnetometer	0.11	1	0.11
RPLidar A3	600	1	600
WiFi Router	500	1	500
Roboclaw 2x30A	30 [7]	2	60
Total Current Draw:			**72144.21 / 1779.21**

Table 7.3: Overview of maximum current draw from components

Table 7.3 provides an overview of power consumption for the electrical components. Calculating an accurate current draw for the system is hard, as the current draw of the thrusters are unknown. But a power draw in idle-mode (no thrusters running) is possible to calculate within a certain margin of error.

The components that have an idle draw are;

- IMU (Acc, Gyro, Mag)
- Raspberry Pi 4
- Arduino Nano
- RPLidar A3

- Internet Router
- Roboclaw

Some of these components go through buck/boost converters, and thus will draw more power because of the efficiency rating it has and are listed below.

- IMU (Acc, Gyro, Mag)
- Raspberry Pi 4
- Arduino Nano
- RPLidar A3
- Internet Router

To calculate the total amount of current from these components, Kirchoffs Current Law is used

$$\sum_{k=1}^{n} I_k = 0$$

The total current is:

$$I_{tot} = 6.1mA + 0.11mA + 575mA + 19mA + 600mA + 500mA = 1700.21mA \tag{7.6}$$

From this one has to calculate how much current will be drawn from the battery to be able to output this with an efficiency rating of 90%.

$$I_{out} = 1700.21mA \tag{7.7}$$

$$I_{out} = 0.9 \cdot I_{in} \tag{7.8}$$

$$I_{in} = \frac{I_{out}}{0.9} = \frac{1700.21mA}{0.9} = 1889.12mA \tag{7.9}$$

The amount of current the Buck/Boost-converters will draw is $1889mA$ at idle with no thrusters running. Adding the small current drawn from the Roboclaws will then total the current draw at $1949.12mA$. The lifetime of the battery with everything just running like this will then be:

$$Lifetime = \frac{20Ah}{1.949} = 10.26h = 10h\,15m\,36s \tag{7.10}$$

An idle lifetime of 10h and 15 minutes is quite fine for this prototype. Considering how infrequent the prototype will run the thrusters and at varying degrees of power it is hard to say anything more precise about what battery life time one can expect in general.

From experience the longest day of testing included having the prototype on water for around 5-6 hours without any problem. This included infrequent running of the motors for most of the time. It started at full battery power and at the end it was as low as 10.0V-10.4V.

7.3 Software

The robot consists of different modules responsible for the different tasks needed to be done in order to achieve autonomous control over the vessel. For this robot it was chosen to divide the program in to three main parts;

- Sensors / Sensor-processing
- Localization / Navigation
- Motor Control

The program also contains a data logger to be able to recreate and analyze the robots behaviour and a data server used by the GUI was created to be able to monitor the system. The data server was implemented as a UDP server which transmitted data the GUI required.

7.3.1 System overview

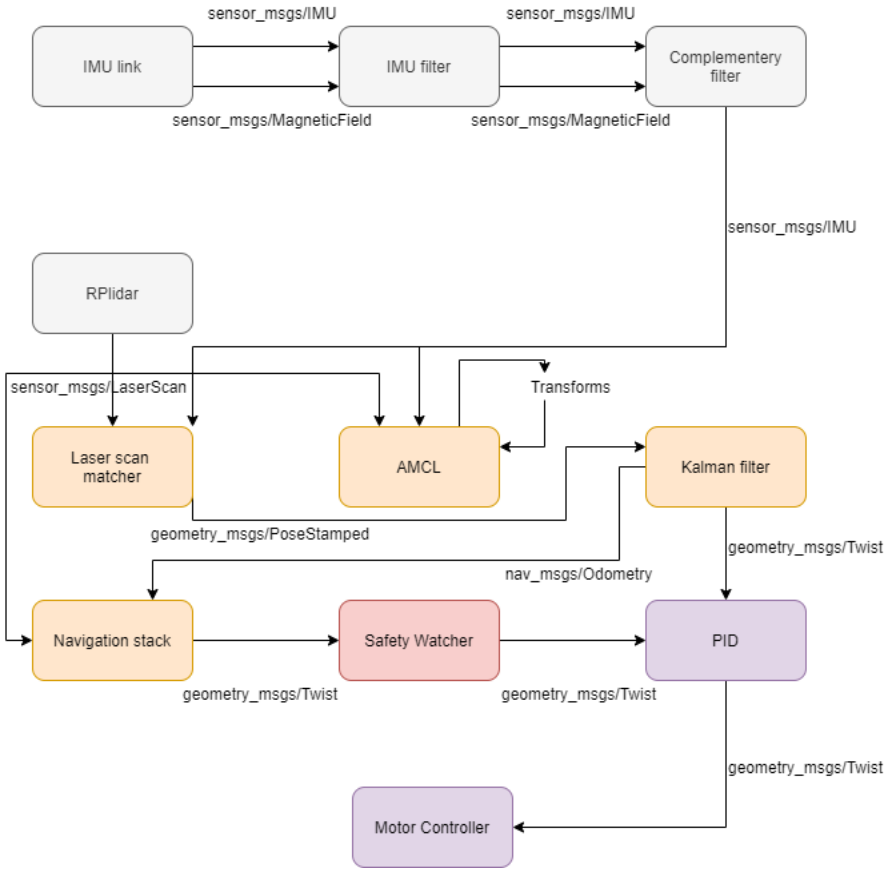


Figure 7.6: Figure displaying system setup and links between modules

Figure 7.6 shows how the software is put together and what nodes send what information. It is color-coded so one can see what part of the system the different nodes belong to.

- White - Sensors and Sensor Processing
- Yellow - Localization and Navigation
- Purple - Motor Control

There is one node in red, which is the safety watcher described in 7.3.4. This has its own color as it doesn't inherently belong to one part of the system.

7.3.2 Sensors and Sensor Processing

The sensor module consists of four nodes;

- IMU-Link
- IMU Filter
- IMU Complementary Filter
- RPLidar

IMU-Link and RPLidar are directly responsible for gathering data about the robots surroundings and orientation in the real world. IMU Filter and IMU Complementary Filter processes data for the IMU so it is useable for the rest of the system. The RPLidar delivers already processed data and does not need any extra steps to make the data useable for the rest of the system.

IMU Link

The IMU Link node establishes a connection with the arduino responsible for reading the 10DOF-IMU used for getting the orientation data. Gyroscope, magnetometer and accelerometer data is sent up to the node as raw unprocessed data using UART over USB for the arduino to the IMU Link-node. This data is parsed in to different ROS-messages representing this type of data. Namely ROS IMU-message containing the accelerometer and gyroscope-data, and a ROS Magnetic Field-message containing the magnetometer-data. IMU Link publishes these two messages using the ROS Messaging System.

RPLidar

This node connects to the RPLidar A3 on the vessel using a UART to Serial Bridge. The node is part of the software following the RPLidar from the start and does all the processing out of the box. It reads in data from the lidar and packages it as a ROS Laser Scan-message. It is then sent along using the ROS Messaging-system.

IMU filter

The IMU Filter-node does mainly two things; it removes a static error in the readings from the IMU and filters some of the sensors for the IMU after removing this static error.

After the static error is removed from all sensors, the accelerometer and the magnetometer is filtered. The accelerometer is filtered using a lowpass filter to remove unwanted noise, the same is true for the magnetometer.

After the values are filtered they are sent to the complementary filter using the ROS Messaging-system.

IMU Complementary Filter

This node is a premade node used for combining data from an IMU in to orientation-data. It takes in filtered IMU data and magnetometer data provided by the IMU Filter-node as ROS IMU and ROS Magnetic Field-messages. It then combines all this data to produce orientation data, which is given relative to the earths magnetic field. This was found to not be perfect and had a tendency to drift slowly in yaw. But this was counteracted by SLAM / laser_scan_matcher because it also gives out an orientation.

This is then sent along using the ROS Messaging-system to the rest of the program.

7.3.3 Navigation and Localization

The navigation module consists of 3 main parts;

- The Odometry
- The Mapping / Positioning
- The Guidance / Navigation

Odometry

The odometry on the robot is a secondary source of translation and orientation input used to help the localization-module localize the robot. The most common source is wheel encoders, or some other way of detecting the robots movement relative to the surface it moves on. Because of the low accuracy of measuring movement relative to water the robot instead uses the input from the complementary filtered IMU in combination with a laser scan matcher to measure its translation and orientation relative to its start point (When the node is started).

This worked well but another source of odometry that is not dependent on the lidar would have been preferred. This would enable the platform to not lose its positioning when the lidar has no surfaces to refer to.

Localization

To localize the robot in its surroundings Adaptive Monte Carlo Localization (AMCL) is used. This node takes odometry data from the scan matcher to get an estimate of the robots position. The node combines this data with inputs from the RPLidar to generate a position relative to a known map of the area.

Navigation

To be able to make the robot move and dynamically avoid obstacles the ROS Navigation Stack is used. This node uses the position from the localization in combination with the RPLidar to generate a path through the environment and dynamically avoid new obstacles. Given a wanted location the robot should move to, the node generates a path to this location and dynamically updates it as the robot moves avoiding any new obstacle in the path. The node outputs a desired velocity the robot should achieve to move towards the new position.

7.3.4 Motor Control

Controller

To make the prototype run at a consistent speed three PID-Controllers were implemented. One for each degree of freedom;

- Forward / Backward
- Lateral Right / Left
- Turning

This controller takes in the desired velocity from either the navigation stack or the XBox-controller and the estimated velocity from the kalman filter-node, using this to generate a control signal for the motors.

Motor Translator / Watchdog

The watchdog for the motor controller stands as a middle man between the nodes sending commands to the motors (Navigation Stack / Xbox-controller) and decides which operating mode the robot should use.

It also keeps control if the raspberry pi loses connection to the PC. If this happens, the watchdog will time out and start sending 0-speed commands to the motors. This is to ensure that the prototype does not have the ability to run haywire.

7.3.5 GUI

To interact with the prototype vessel a graphical user interface was used. This user interface was built as a web based framework. This framework was created with Flask, which is a web based framework for Python, and it acted as a front and back end for the GUI. Here Flask returned the web pages that together created the GUI. These web pages was structured with HTML, styled with CSS and lastly JavaScript gave it the ability to interact with the pages.

Since the GUI was built with Flask and had multiple web pages, it was created a main template page for some standardization. Whatever this template contained would show up on all web pages throughout the GUI. This template consisted of a navigation bar and a footer. The navigation bar featured buttons to navigate between the pages of the GUI, as well as the official logo for the project, as showed in figure 7.7. Whilst the footer contained an emergency button, that made it able to stop the boat regardless of what page the user was on and a "contact" button, both seen in figure 7.8. The "contact" button would display the contact information from all the members of the project, in case questions arose, showcased in figure 7.9.

Figure 7.7: The navigation bar that make up the top of the GUI

Figure 7.8: The footer bar that makes up the bottom of the GUI

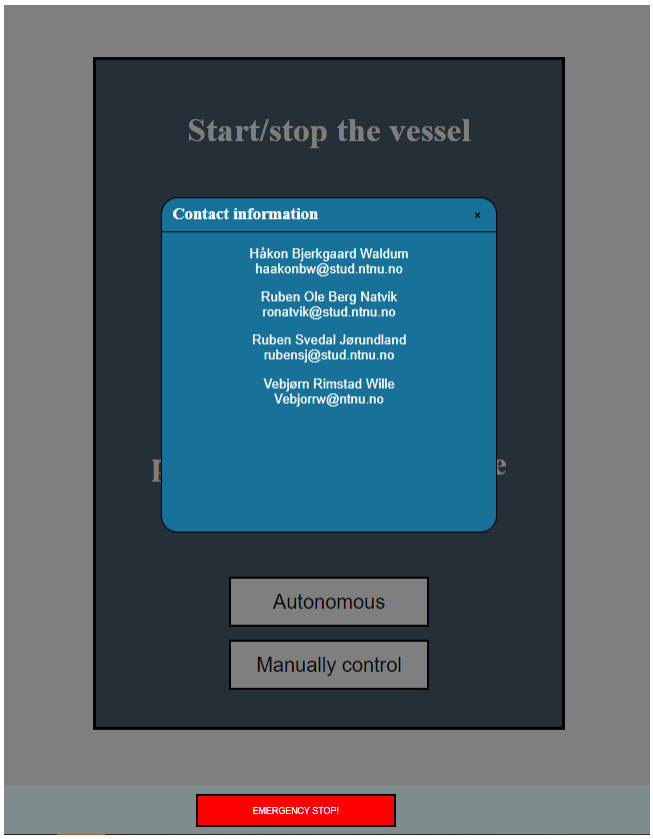


Figure 7.9: The Modal that pops up when the contact button is pressed

The three pages that made up the user interface was the "Home" page, "Navigation" page and the "Advanced" page. The "Home" page function was to send commands to either set the vessel in run or idle mode, along with the option to change between autonomous or manual control, as showed in figure 7.10. These options were presented with four buttons, that when pressed would send a specific constructed message to ROS via TCP communication. However, due to time constrains, the communication on the vessel was not implemented, and therefore the messages was never used to interact with the vessel and the GUI was used purely as a monitoring system.

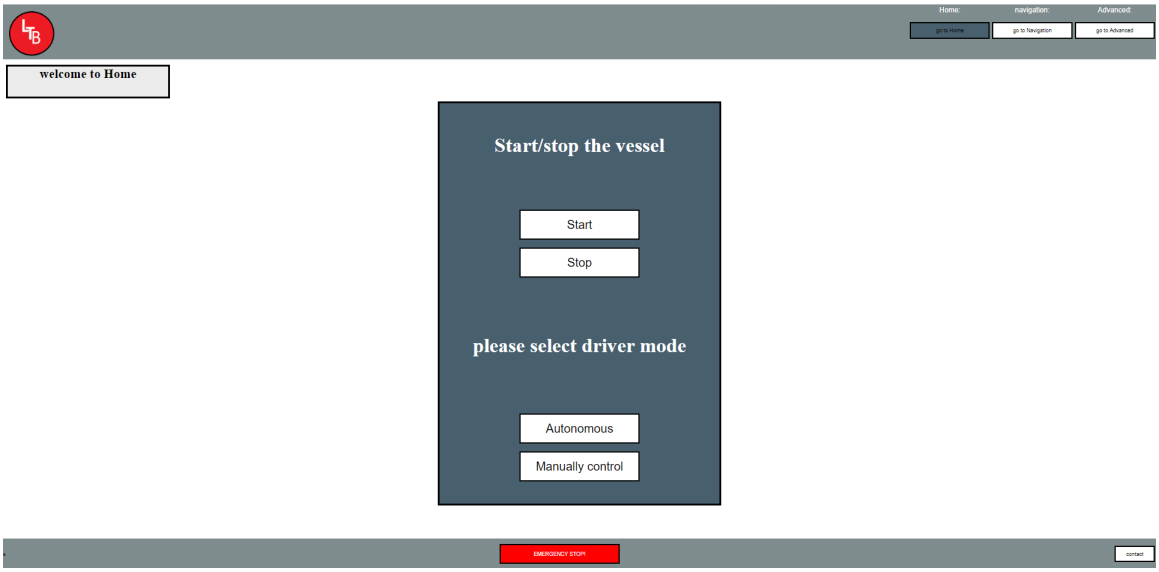


Figure 7.10: The Home page of the GUI

The Navigation page of the GUI was designed to contain all the essential navigation information for the vessel, and can be seen in figure 7.11. The navigation essentials were the behavior of the motors based on the input, the raw Lidar data, the speed of the vessel in x, y and ψ direction and the scanned map of the surroundings, with the boats position marked. Since this information would be sent rapidly, all information was sent via UDP communication from ROS. Lost UDP packets would not affect the data since new packets would arrive in time to have no effect on the information. This information then was parsed and displayed with the JavaScript library Plotly.js.

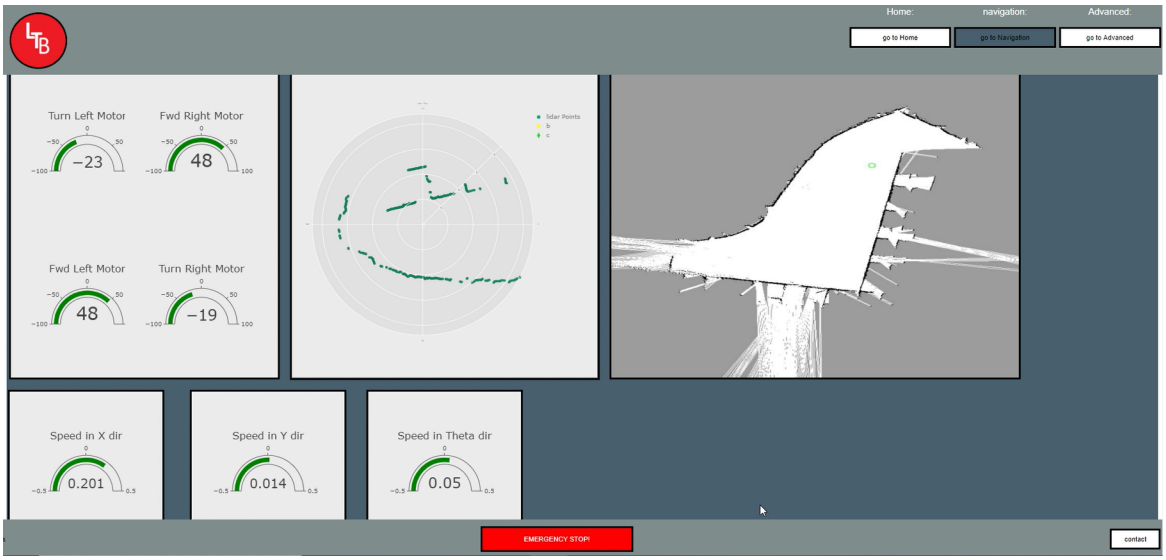


Figure 7.11: The navigation page in a working scenario

The motor input gauges was configured in such way that they represent the actual position of the motors on the vessel inside the GUI. The intent with this configuration was to create a more intuitive view of the motion of the vessel compared to the reaction from the motor input gauges. Since the value that the motor gauges received were the same as the input sent to the RoboClaw motor controller, the motor gauges had configured threshold values from -100 to 100, where -100 was full reverse and 100 was full forward for each motor.

The raw Lidar scan was included to observe what the Lidar sensor retrieved and compare it with the retrieved map given by ROS. This could also be used for debugging purposes by more intuitively observing if any errors occurred from the optical Lidar, as the Lidar might be susceptible to measurement errors.

The scanned map containing the surroundings and the location of the vessel, was included as it served the purpose of showing the vessels location relative to land.

The Advanced section of the GUI was designed contain all technical information of the system. These types of information would be displaying battery life, ampere usage and controller values from the PID controller. Due to time constraints this implementation was cut from the

GUI since it lacked the necessity to complete the vessel. Figure 7.12 showcases the structure that this page would have had.

Figure 7.12: The structure of the Advanced page ready for more information

To prevent the web pages from refreshing every time new data was received from UDP, the GUI used a method to dynamically update the old variables with new data. This dynamic method involved the use of jQuery, a JavaScript library, to compute an interval function that updated the new data dynamically. The data was updated by making a Json call to the flask server every 500ms, the new Json data got parsed and plotted again with Plotly.js .

7.4 System Identification

The velocity dynamics model takes the shape $\dot{v} = Av + B\tau$, where

$$A = -D \cdot M_{RB}^{-1} = \begin{bmatrix} A11 & 0 & 0 \\ 0 & A22 & A23 \\ 0 & A32 & A33 \end{bmatrix} \tag{7.11}$$

$$B = M_{RB}^{-1} = \begin{bmatrix} B11 & 0 & 0 \\ 0 & B22 & B23 \\ 0 & B32 & B33 \end{bmatrix} \tag{7.12}$$

$$\nu = \begin{bmatrix} u \\ v \\ r \end{bmatrix} \tag{7.13}$$

$$\tau = \begin{bmatrix} X \\ Y \\ N \end{bmatrix} \tag{7.14}$$

and the states u, v, r represent the vessel surge, sway and angular velocity respectively. The model from T. I. Fossen also includes an equation describing the relationship between ν and the position vector η. As this system model is used for velocity control, this relationship is not necessary in the case of this project.

As described in 4.7 a GA was used in order to determine the parameters for A and B. The best solution the GA could find had the following genes:

$$G_{best} = \begin{bmatrix} 56.2646 & 55.1343 & 2.6548 & 1.9900 & 19.9970 & 4.7235 \end{bmatrix} \tag{7.15}$$

These genes were used in the D and M_{RB} matrixes which were then calculated as the following:

$$D = \begin{bmatrix} 56.2646 & 0 & 0 \\ 0 & 55.1343 & 2.6548 \\ 0 & 1.9900 & 19.9970 \end{bmatrix} \tag{7.16}$$

$$M_{RB} = \begin{bmatrix} 60 & 0 & 0 \\ 0 & 60 & 0 \\ 0 & 0 & 18.9266 \end{bmatrix} \tag{7.17}$$

Solving for A and B then gives the state space model

$$\dot{\nu} = \begin{bmatrix} -0.9377 & 0 & 0 \\ 0 & -0.9189 & -0.1403 \\ 0 & -0.03317 & -1.057 \end{bmatrix} \nu + \begin{bmatrix} 0.01667 & 0 & 0 \\ 0 & 0.01667 & 0 \\ 0 & 0 & 0.05284 \end{bmatrix} \tau \tag{7.18}$$

$$y = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \nu \tag{7.19}$$

This model had the results displayed in figure 7.13:

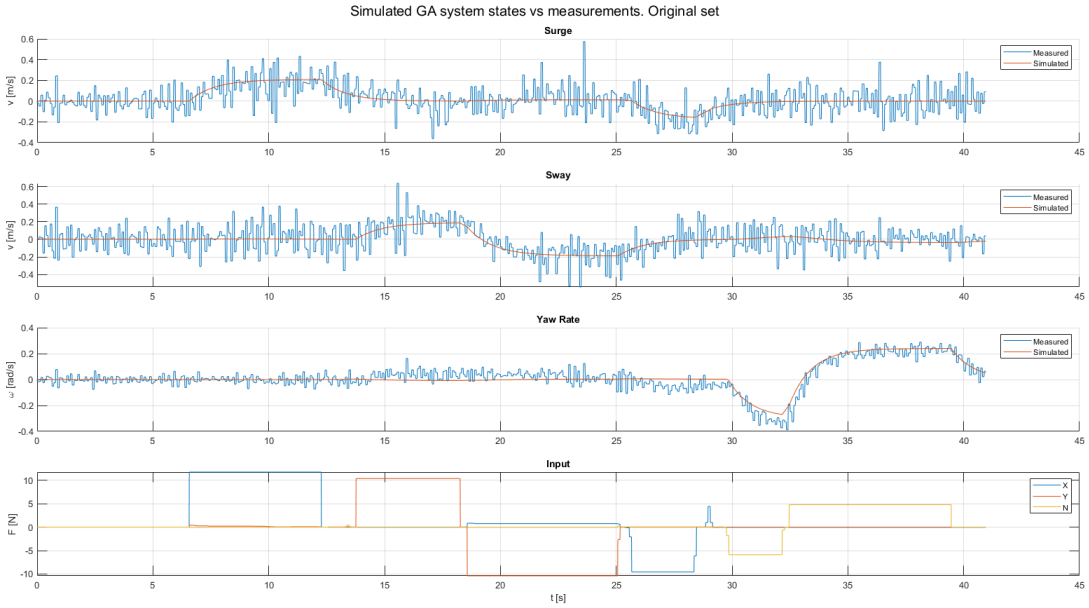


Figure 7.13: Plot from Matlab displaying the simulated system states found with GA, compared to the estimated states from measurements done during system identification test run.

The top three plots show the vessel states estimated from measurements in blue and the simulated GA states in red. The bottom plot displays the recorded input for the input vector τ.

This result looked promising and the model was tested with different motion recordings to determine if the system responded accurately for inputs different from the set that was used for

the cost function to match up against. Figure 7.14 and 7.15 displays the result from these simulations.

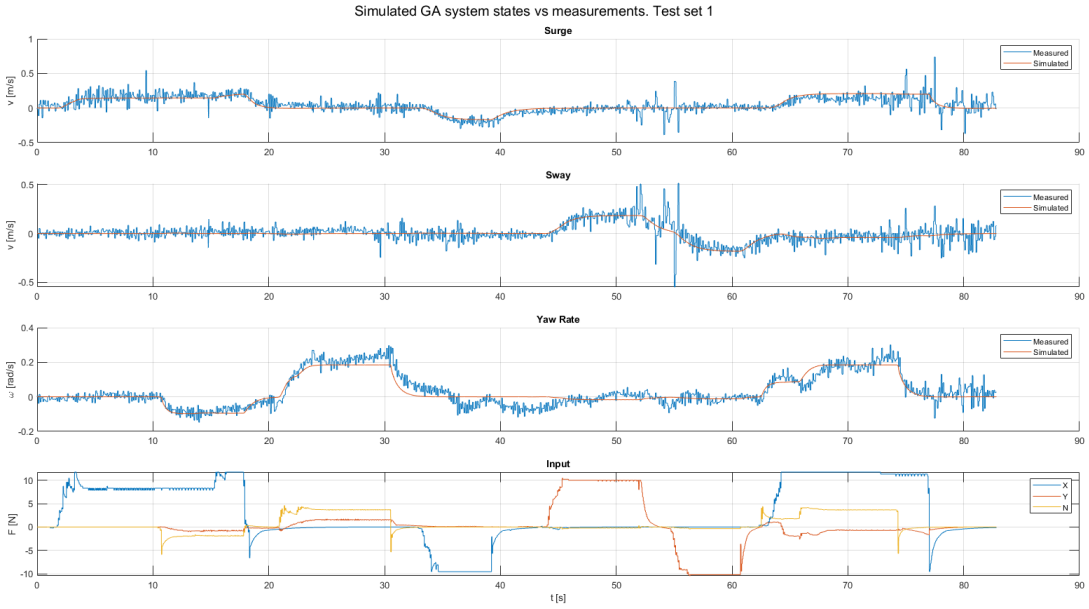


Figure 7.14: Plot from Matlab displaying the simulated system states found with GA, compared to the estimated states from measurements of a different test run.

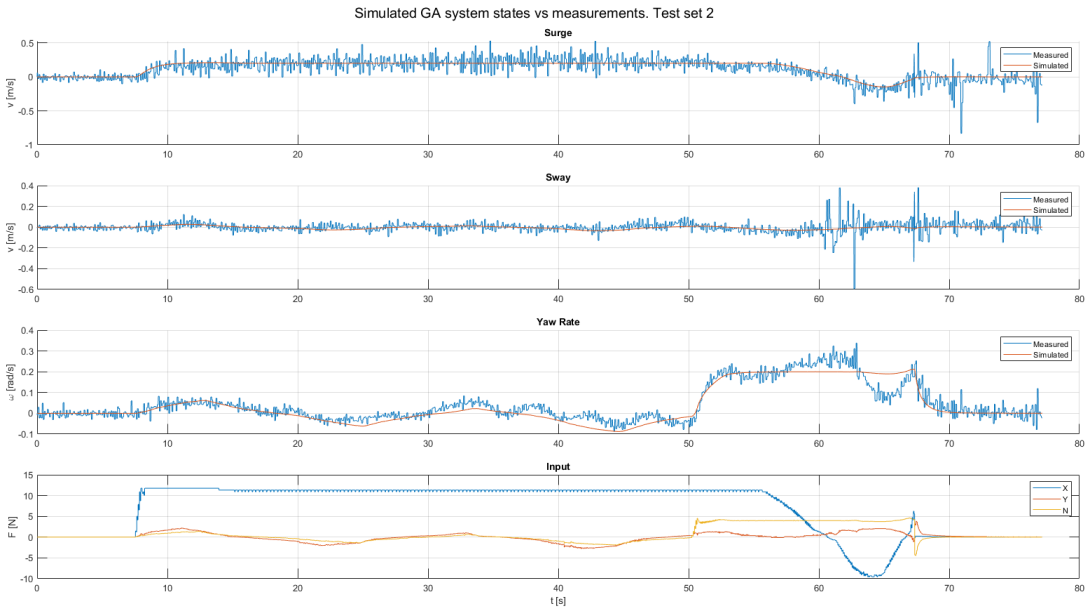


Figure 7.15: Plot from Matlab displaying the simulated system states found with GA, compared to the estimated states from measurements of a different test run.

As can be seen from these figures, the system describes the states relatively well. The simulated response does not match the estimated states perfectly, which is expected as these recordings were not made after ensuring low external force input like the original set was. This error was factored in the Kalman filter as process noise.

7.4.1 Kalman filtering

The system model was incorporated into a Kalman filter for the feedback loop of the prototype's PID regulator. It was implemented in Python as a ROS Node that listened for output from the PID regulator and readings from the Laser Scan Matcher. As the device used for measuring the vessel's velocities was the Lidar, the data from this had to be transformed into body velocities, which meant derivation of position data. This introduced a lot of noise in the measurements. In an attempt to reduce this noise the scan matcher was sampled five times before averaging these measurements. This average was then sent to the Kalman filter at a rate of 2Hz, since the scan matcher publishes odometry with a rate of 10Hz, and 5 samples were averaged before yielding the measurement to the Kalman filter. The Kalman filter's predict cycle was run at a rate of 20Hz,

as this was the publishing rate of the PID regulator, which meant that for each measurement the Kalman filter ran 10 prediction cycles.

Figure 7.16 shows the output of the Kalman filter for one of the path following tests that was performed as part of the final solution. The blue 'x' marks represents the filtered measurements while the red line represents the estimated states from the Kalman filter. As can be seen from this figure the Kalman filter provides an adequately stable and accurate value for the estimated system states.

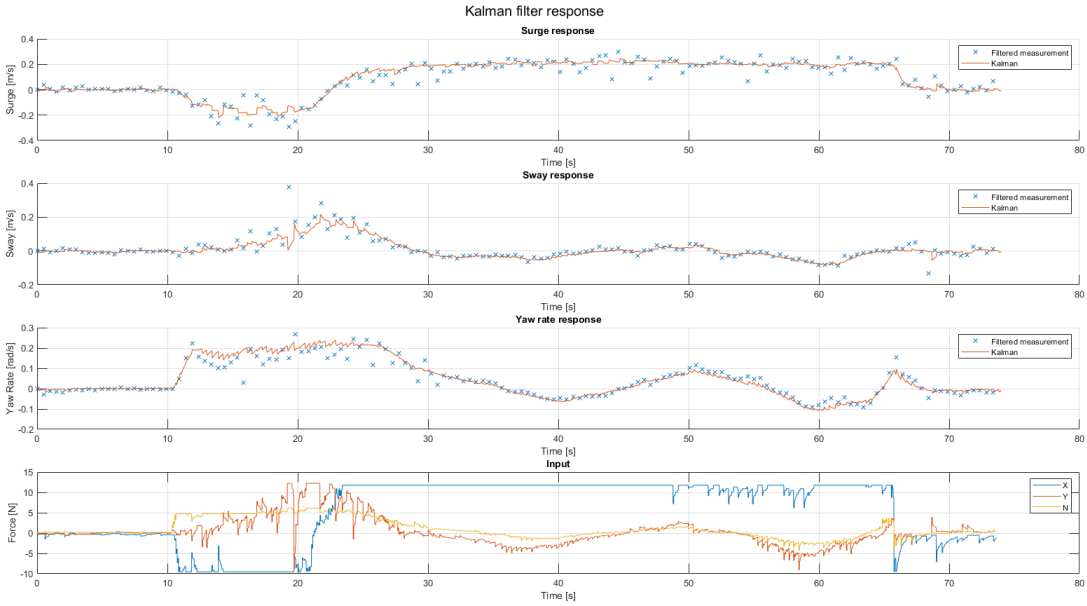


Figure 7.16: Output from the Kalman filter

7.5 Construction Test Results

The construction test results do not present any meaningful data in the form of data extracted from sensors, but more experiences and the group witnessing the prototypes behavior in the water.

7.5.1 Buoyancy Testing

The result of the buoyancy testing was quite fine. Without any electrical parts mounted to the prototype it floated quite high in the water, but was stable. Increasing the amount of ballast and simulating the electrical equipment in the form of bags of sand showed that it would place itself nicely in the water with enough room between the base plate and the water.

7.5.2 Stability Testing

With the battery placed under water, the stability of the boat was fine as well. The centre of gravity was pushed low by taking the heaviest component and placing it almost at the bottom of the prototype.

In an earlier design described in section 3.1.3 there was trouble with huge impact on roll/pitch when pushing it. This was mitigated with fins in the previous iteration. In this iteration when mounting the battery under water on a base plate, this base plate had much of the same effect as the fins and reduced the impact on roll/pitch.

7.5.3 Waterproof Testing

The result of the waterproof testing on the boat showed that one of the vertical tubes were not waterproof. This was a result of a cut not being straight, and was later fixed when the vertical tubes were shortened and the plugs were changed to plugs with gaskets.

The electrical box for the battery showed in the initial tests that it was not waterproof. Sealant was added to all points that could cause a leak, and the battery was added into the box, and the rest of the box filled with oil.

The box was then observed again after 24 hours, and some small leaks were found and more sealant was added. After this the boat was released on water, and the box was found to not be completely tight still. It was taken onto land, the box was opened and emptied, and all possible cracks were filled with sealant and the box resealed and filled with oil.

After this the tests showed that the box was waterproof.

7.5.4 Weight Distribution Testing

The result of the weight distribution test showed that it was not as sensitive to uneven weight distribution of the components. And the reduced weight of the components on top of the platform since the battery was placed under water helped with this as well.

7.6 Initial Tests

7.6.1 Initial System Tests

When the initial tests of the electrical system and the software was run, everything seemed to work quite OK. There were small problems with thrusters running in the wrong direction, but this was fixed quite easily with a polarity change in the software.

After implementing a safety watcher to ensure that the thrusters would not run amock in the case of something disconnecting, a huge lag started occurring for the manual control system. This was found to be because of the safety watcher taking up too much of the cycle time.

This was fixed by making it check with a rate of 10Hz, and sleeping in the time in between. This fixed the lag and made the system responsive again. The Lidar and IMU were also checked and seemed to respond well and gave the data that was expected.

This initial test went quite well and only presented small problems that were easily fixed in a matter of hours, making it possible to test on water quite quickly after.

7.6.2 Initial Tests on Water

Initial tests on water with all electrical components attached to the prototype showed that the max speed that was possible to use would be around 50% of max speed. If the speed exceeded

this, the prototype would start tilting too much forward, and the Lidar would lose track of where it was.

A problem with the tilting was also the danger of getting too much sea water on the top part of the prototype, and then the danger of water actually coming inside some of the boxes with electrical equipment. Therefore a limit of 50% max speed was set in the software.

7.7 System Response Test Results

7.7.1 Bollard Pull

The results of the Bollard Pull are presented in table 7.4:

Roboclaw Signal	Weight (kg)
20	0.17
30	0.45
40	0.76
50	1.20
60	1.70
70	2.20
80	2.70
90	3.40
100	3.90
110	4.50
120	5.00

Table 7.4: Bollard Pull Results

When comparing these values with the signal sent to the engines, a power curve can be produced for the thrusters as presented in figure 7.17.

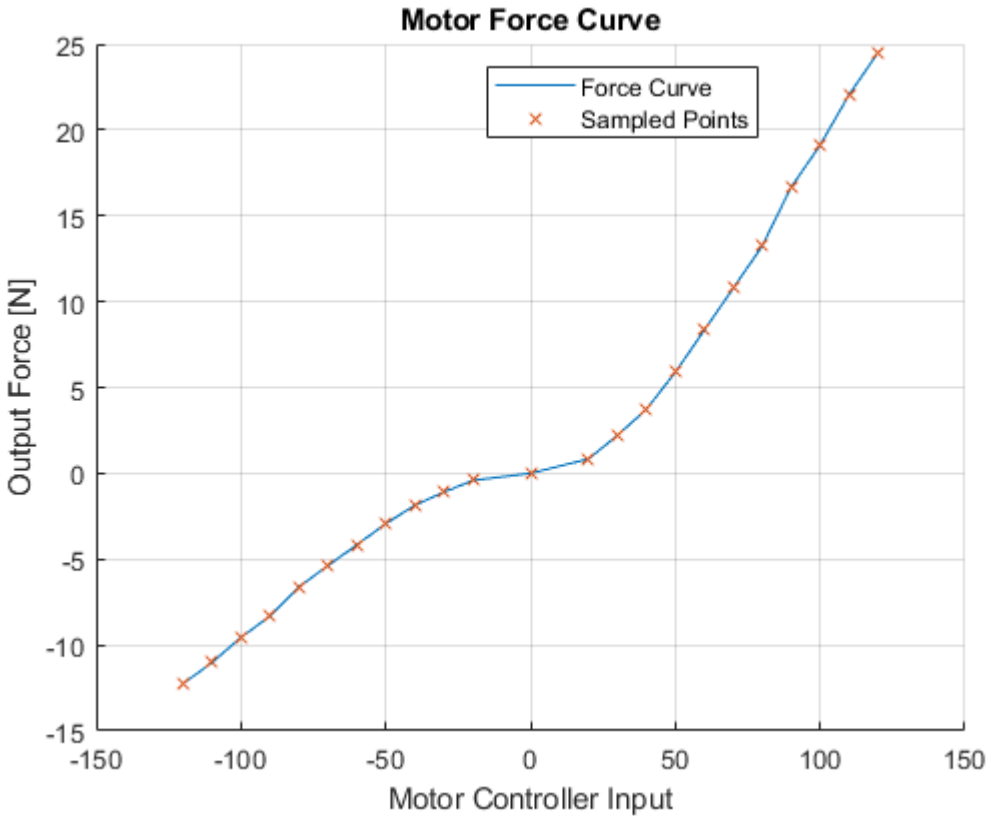


Figure 7.17: Output force for input PWM signal

It should be noted that from testing it was found that the motors were not as effective in reverse as they were when running forward. It was found that for the motors to produce the same amount of force they had to run 1.3 times as fast as if running forward. The negative side of the power curve is therefore scaled with this in mind, and is not from actual tests.

7.7.2 Rotation test

The results of the rotation tests were as displayed in figure 7.18. The red line at yaw rate 0.26 represents the estimated steady state velocity for the vessel's yaw rate. The spikes seen at approximately time 38s and 51s is a product of the yaw being represented as euler angles, which are represented in the interval $-\pi$ to π. These spikes are then the product of the platform rotating beyond the limits of the interval, causing spikes in velocity.

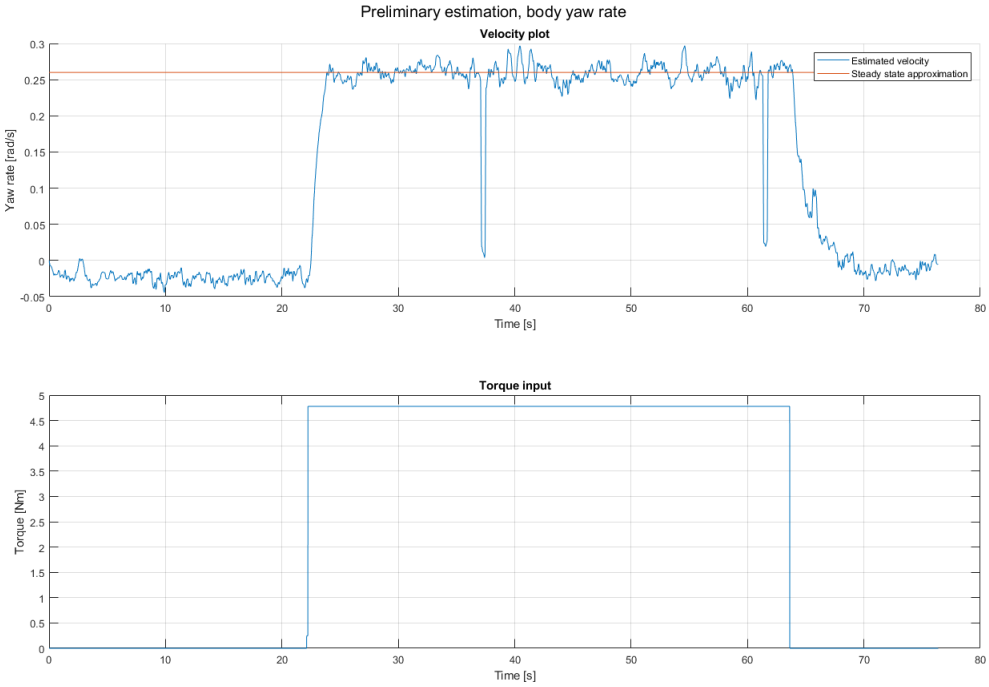


Figure 7.18: Rotation speed of the platform

7.7.3 Speed tests

The results of the forward speed tests were as follows;

Manual timing

The prototype drove a recorded length of 13m a total of 3 times, while each lap got timed. The result of this test can be found in table 7.5.

Round	Time
1	50.43s
2	53.67s
3	50.90s
Average Time	51.67s

Table 7.5: Forward Speed Test Lap Times

By taking the total length and dividing that by the average time from table 7.5, the final result is displayed in equation 7.20

$$v = \frac{13m}{51.67s} = 0,251m/s \tag{7.20}$$

Tracker Speed Test

During the forward speed tests it was decided to film the vessel while it moved in the forward direction. These film sequences, five in total, got loaded onto Tracker for further video analysis. The video analysis of the vessel moving forward was done with the Tracker procedures, described in method section 4.6.2. This process was repeated for all five film sequences. Tracker then returned five position- time- graphs, displayed in 7.19,7.20, 7.21, 7.22 and 7.23.

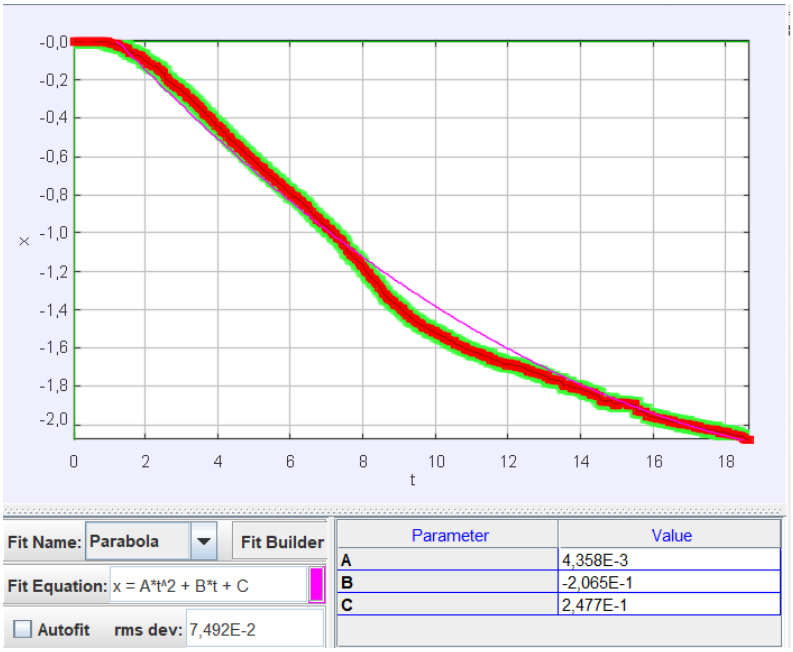


Figure 7.19: position- time- graph from the first forward run of the Vessel

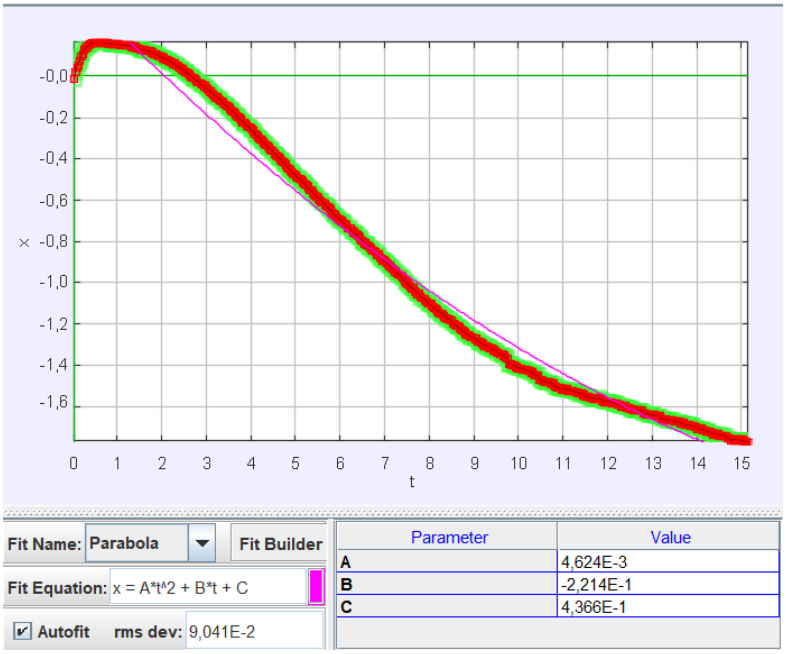


Figure 7.20: position- time- graph from the second forward run of the Vessel

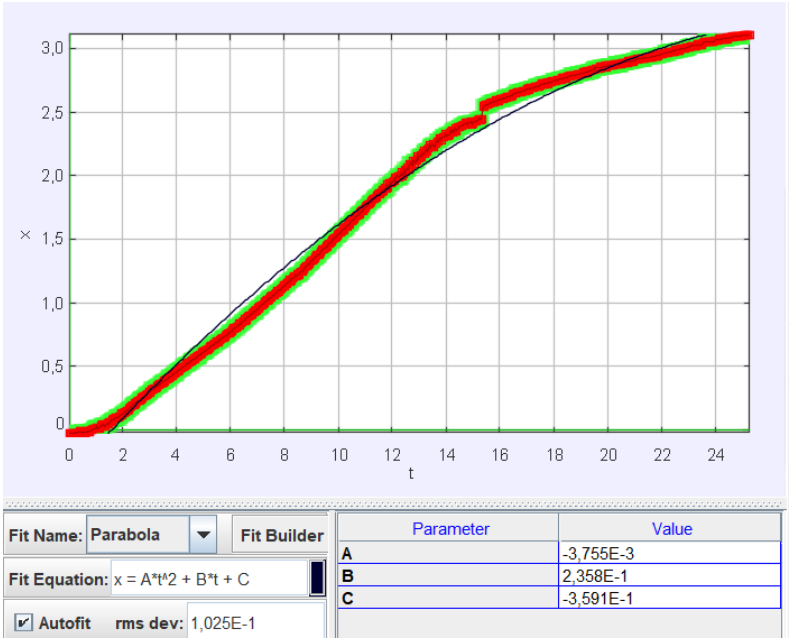


Figure 7.21: position- time- graph from the third forward run of the Vessel

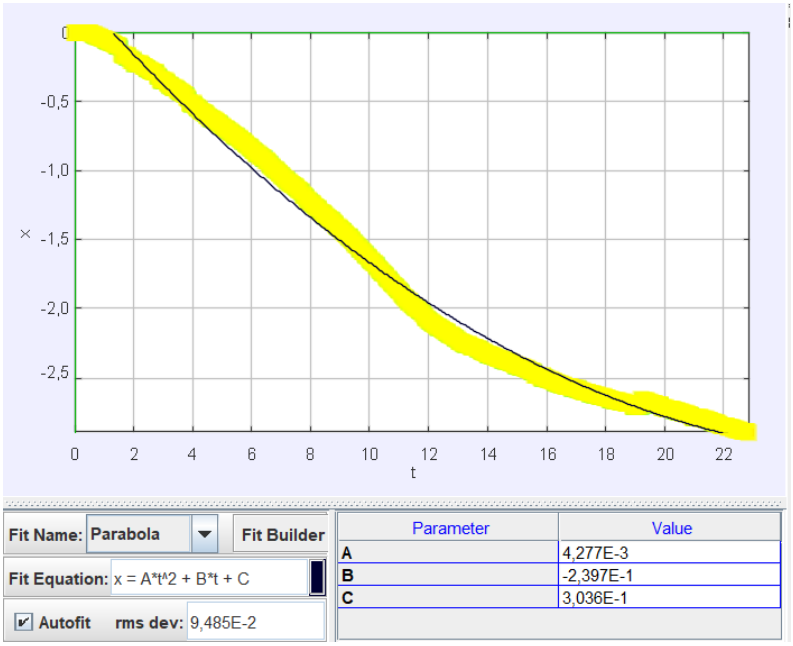


Figure 7.22: position- time- graph from the fourth forward run of the Vessel

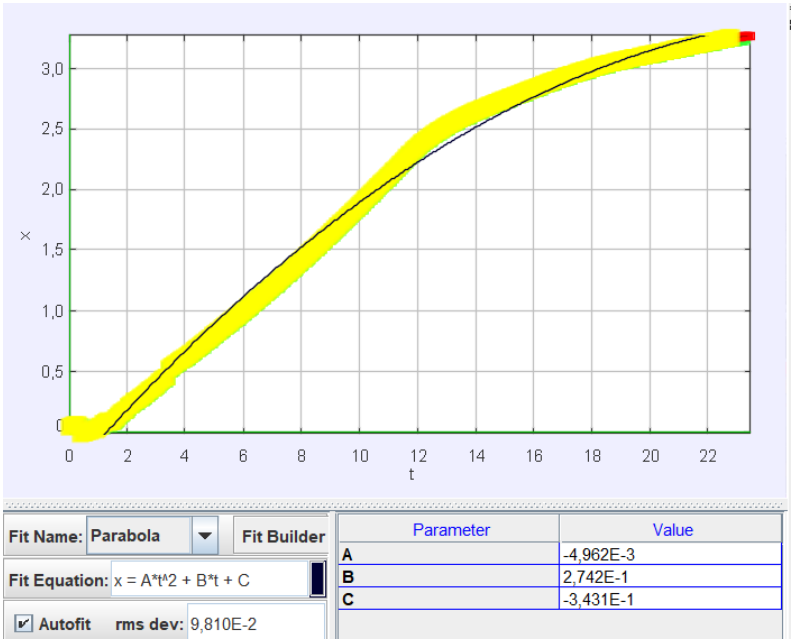


Figure 7.23: Position- time- graph from the fifth forward run of the vessel

With all five graphs computed and the A, B and C variables constructed from the fitted quadratic equation from the figures 7.19, 7.20, 7.21,7.22 and 7.23, the acceleration and the velocity from the graphs was calculated with the following equations.

Computing the acceleration for the vessel from run 1:

$$A = 4.359 \cdot 10^{-3} \tag{7.21}$$

$$a = 2 \cdot A \tag{7.22}$$

$$a = 8.7 \cdot 10^{-3} \quad m/s^2 \tag{7.23}$$

Computing the velocity for the vessel from run 1:

$$B = 2.1 \cdot 10^{-1} \tag{7.24}$$

$$v = B \tag{7.25}$$

$$v = 2.1 \cdot 10^{-1} \quad m/s \tag{7.26}$$

This calculation process was then repeated for all five graphs. See figure 7.24 for more detailed results of the calculated acceleration, and figure 7.25 for more detailed velocity calculation results.

By inspecting the graphs, in the figures: 7.19, 7.20, 7.21,7.22 and 7.23, it was clear to see that the quadratic equation did not fit the graph perfectly. Because the fitted equation was not a perfect match, it was safe to assume that the result might vary from the true values for acceleration and velocity. Thus the constructed values of A, B and C wouldn't be a perfect representation of acceleration, velocity and distance for the vessel.

Because there was some variation and uncertainty in the five measurements for the acceleration and velocity values, statistics where used to further reduce the amount of uncertainty. This was done by representing the result for both acceleration and velocity as: $result = average \pm SE$

(2.1).

For acceleration:

Acceleration average using formula 2.2

$$\bar{a} = \frac{\sum_{i=1}^{5} a_i}{N} = 8.8 \cdot 10^{-3} m/s^2 \tag{7.27}$$

Acceleration standard deviation using formula 2.4

$$\delta a = \sqrt{\frac{\sum_{i=1}^{5} (a_i - 8.8 \cdot 10^{-3})^2}{5}} = 8.9 \cdot 10^{-4} \tag{7.28}$$

Acceleration standard error using formula 2.3

$$\delta \bar{a} = \frac{8.9 \cdot 10^{-3}}{\sqrt{5}} = 4.0 \cdot 10^{-4} m/s^2 \tag{7.29}$$

Finally the result:

$$a = 8.8 \cdot 10^{-3} \pm 4.0 \cdot 10^{-4} m/s^2 \tag{7.30}$$

For velocity:

Velocity average using formula 2.2

$$\bar{v} = \frac{\sum_{i=1}^{5} v_i}{N} = 2.5 \cdot 10^{-1} m/s \tag{7.31}$$

Velocity standard deviation using formula 2.4

$$\delta v = \sqrt{\frac{\sum_{i=1}^{5} (v_i - 2.5 \cdot 10^{-1})^2}{5}} = 3.1 \cdot 10^{-2} \tag{7.32}$$

Velocity standard error using formula 2.3

$$\delta \bar{v} = \frac{3.1 \cdot 10^{-2}}{\sqrt{5}} = 1.4 \cdot 10^{-2} m/s \tag{7.33}$$

Finally the result:

$$v = 2.5 \cdot 10^{-1} \pm 1.4 \cdot 10^{-2} m/s \tag{7.34}$$

Underneath is the collected result for acceleration displayed in figure 7.24, and in figure 7.25 the collected result for the velocity of the vessel is displayed.

Type of Motion	Acceleration [m/s²]	Average Acceleration	Standard error	Result
Forward	Raw values: Run 1: 8.7 ·10-3 Run 2: 9.2 · 10-3 Run 3: 7.5 · 10-3 Run 4: 8.6 · 10-3 Run 5: 9.9 · 10-3	$\bar{a} = (\sum a)/N$ $\bar{a} = 8.8 \cdot 10^{-3}$ m/s²	$\delta\bar{a} = \delta a / \sqrt{N}$ $\delta\bar{a} = 4.0 \cdot 10^{-4}$	Result = $\bar{a} \pm \delta\bar{a}$ Result = $8.8 \cdot 10^{-3} \pm 4.0 \cdot 10^{-4}$ m/s²

Figure 7.24: The acceleration result represented by a table with SE

Type of Motion	Velocity [m/s]	Average Velocity	Standard error	Result
Forward	Raw values: Run 1: 2.1 · 10-1 Run 2: 2.2 · 10-1 Run 3: 2.4 · 10-1 Run 4: 2.4 · 10-1 Run 5: 2.7 · 10-1	$\bar{v} = (\sum v)/N$ $\bar{v} = 2.5 \cdot 10^{-1}$ m/s	$\delta\bar{v} = \delta v / \sqrt{N}$ $\delta\bar{v} = 1.4 \cdot 10^{-2}$	Result = $\bar{v} \pm \delta\bar{v}$ Result = $2.5 \cdot 10^{-1} \pm 1.4 \cdot 10$^-2

Figure 7.25: The velocity result represented by a table with SE

Test via sensor information

During the test runs with the logger enabled, the data was processed to give a general view on how fast the prototype was moving. This was used mainly to set the limits in the navigation stack so the velocity commands were actually realistic.

The data from the test as seen in figure 7.26 show that the data is a bit unstable, but that a steady state approximation is done. This is because its not important exactly how fast it is, just a general ballpark.

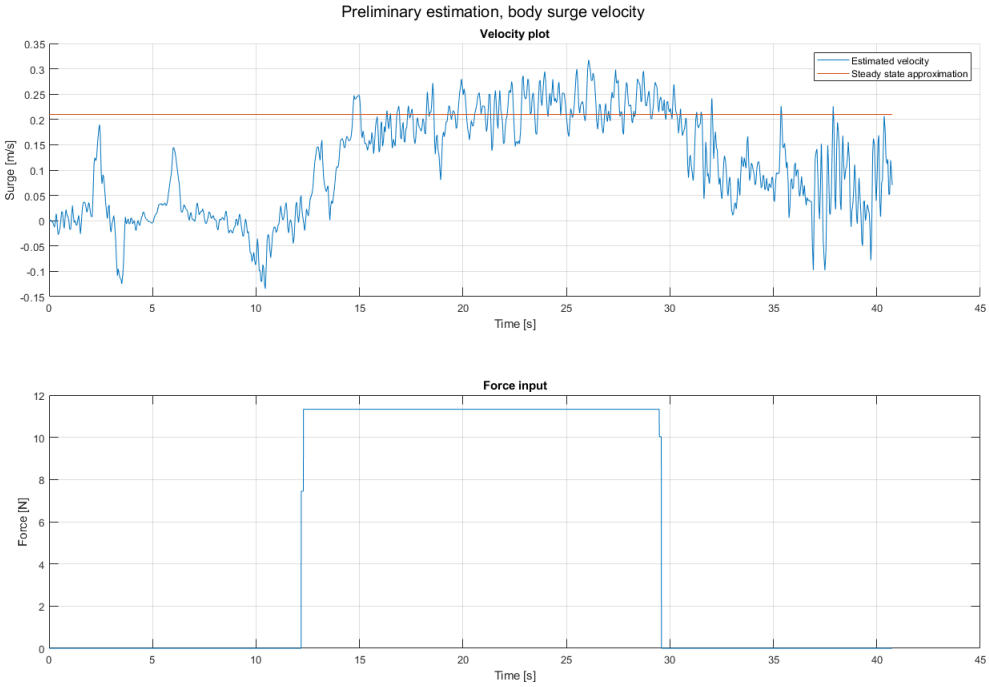


Figure 7.26: Result from forward speed test

It was also run for the lateral shifts, as this was also data that the navigation stack needed. Much of the same is true here as it is for the forward speed test, the data is unstable but a steady state approximation can be made as seen in figure 7.27.

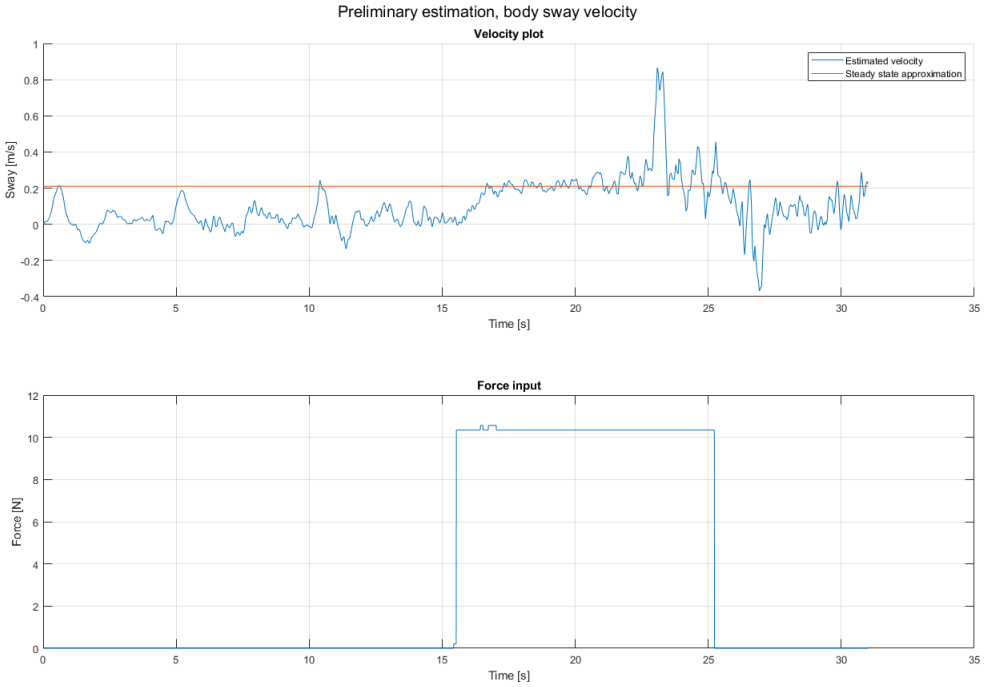


Figure 7.27: Results from lateral speed test

Combined Test Result

From the result of the three different method that tested the forward speed of the vessel, it was possible to find the true speed of the vessel in the forward direction by looking at the combined test result.

From the manual timing test, the equation 7.20 shows that the vessel had a forward speed of $v = 0.251 m/s$. From the Tracker result for velocity displayed in figure 7.25, shows the velocity $v = 0.25 \pm 1.4 \cdot 10^{-2} m/s$. Finally from the graphs displayed in figure 7.26, shows fluctuating results around $v = 0.22 m/s$, this however was affected by the condition of the sensor.

Combining these results showed that all three tests points towards velocity being around $v = 0.22 m/s$ with some variation. This variation was so small that they were negligible and thus indicating that the sensor data gave trustworthy velocity results.

It was made an assumption that the value for sway, displayed in figure 7.27 also would represent an acceptable estimation for velocity, since the sensor data from the forward velocity tests were sufficiently in agreement with the other test methods.

7.8 Navigation Stack Testing

The testing of the navigation stack showed that to really be able to take advantage of the omni-directional possibilities of the prototype, teb_local_planner (from now on referred to as TEB) had to be used. It was the only local planner that actively planned with this in mind, and made full use of it.

When testing dwa_local_planner (from now on referred to as DWA), it worked well in path planning for normal driving where it would make use of rotation as well as normal forward driving. But whenever it overshot the target by a bit, it had to rotate, drive, rotate, drive before getting to the target because it did not make full use of the omni-directional possiblities.

The obstacle avoidance of TEB however was problematic. Because of the low limits in acceleration and speed of the prototype, the amount of time into the future that has to be planned to make a complete path around an obstacle is huge. This combined with TEB being one of the most process-heavy local planners makes it have a lot of trouble avoiding obstacles.

If the prototype approaches quite centered on an obstacle it has trouble deciding which way to go around, and therefore gets stuck just in front of the obstacle. If however the prototype approaches the obstacle more to one side than the other it manages to get around the obstacle without too much trouble.

DWA's obstacle avoidance however worked better than TEB. It avoided obstacles quite fine without any big troubles without needing the prototype to approach more to one side than the other.

When comparing them, TEB was the better fit overall for this project. The fact that it actually made use of the omni-directional possibilities when it was the best choice made it the logical choice, even when one considers the problems with obstacle avoidance.

7.9 Performance Test Results

7.9.1 Pathfollowing Test

From the three pathfollowing tests one can see that when no turning is required, it manages to follow the global path quite closely. Whenever it has to turn before / after doing the path it does deviate a bit to be at the correct angle when finishing.

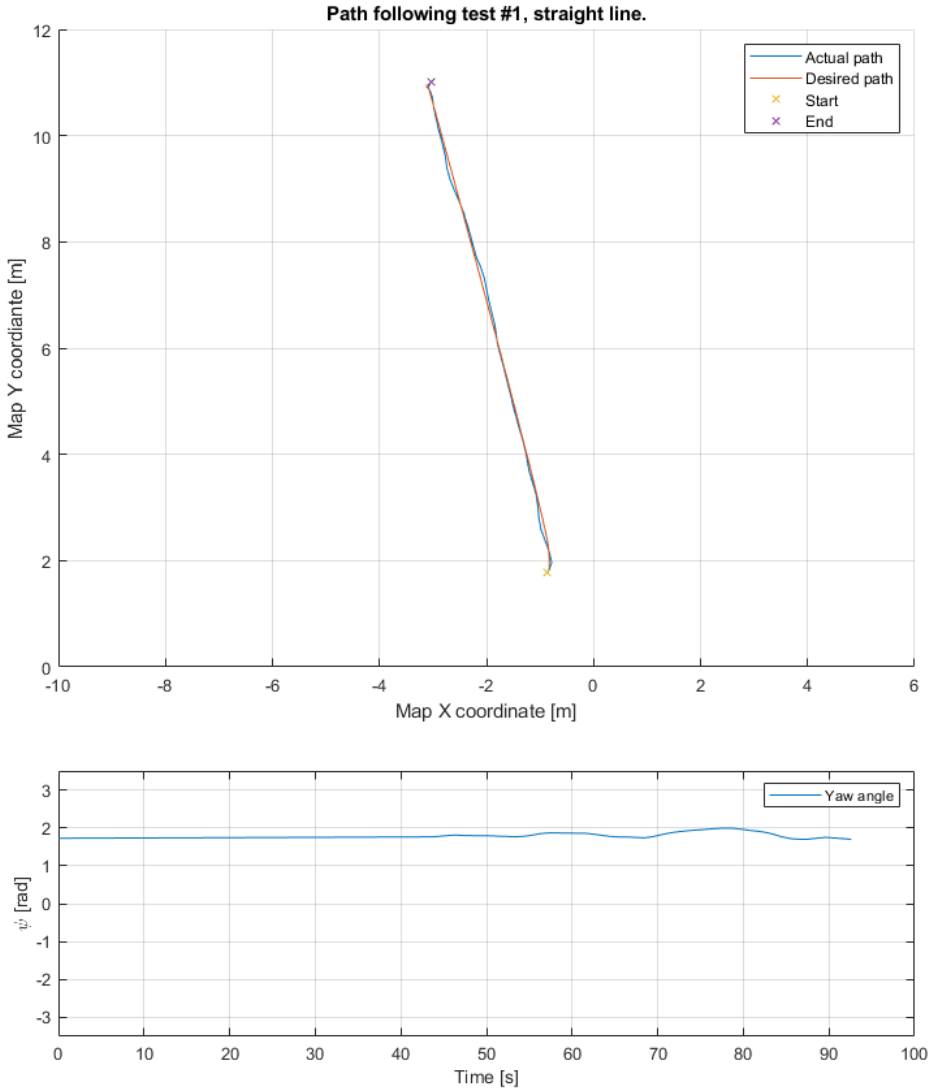


Figure 7.28: Pathfollowing test, straight line, no turning

From the graph showing the yaw-angle in figure 7.28 one can see that here is a slight oscillation, and this shows in the path-plot as well. Over all in this test the prototype drove a total of 9.582m while the global path planned was 9.391m, which is a difference of 19.1cm.

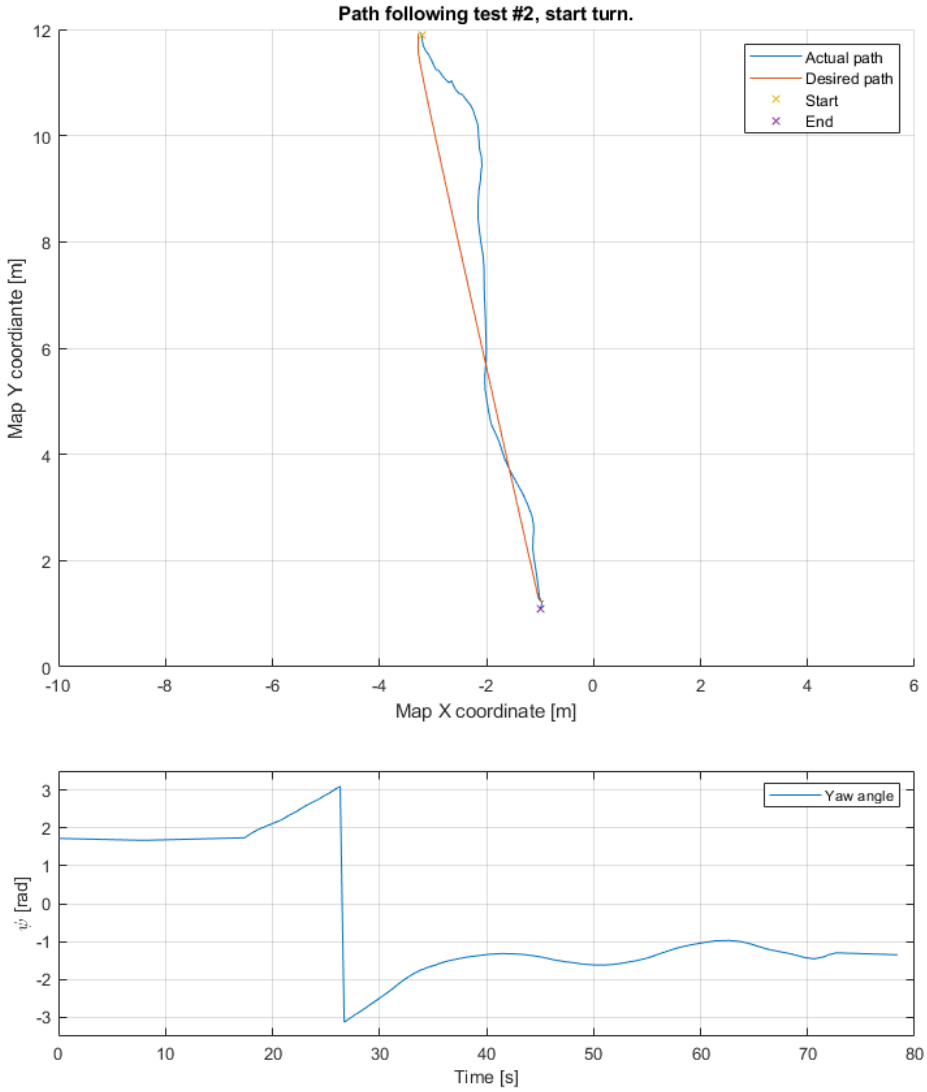


Figure 7.29: Pathfollowing test, 180°turn before driving straight

On the second test the prototype was starting with the front facing the wrong way, and had to turn before following the path. The turn it decided to do was reversing while rotating until the front was facing the right way, then driving forwards and honing in on the path.

This did make it deviate quite a bit from the path that was planned as seen in figure 7.29, and resulted in the prototype traveling 11.615m instead of the global paths 11.016m which makes a difference of 59.9cm.

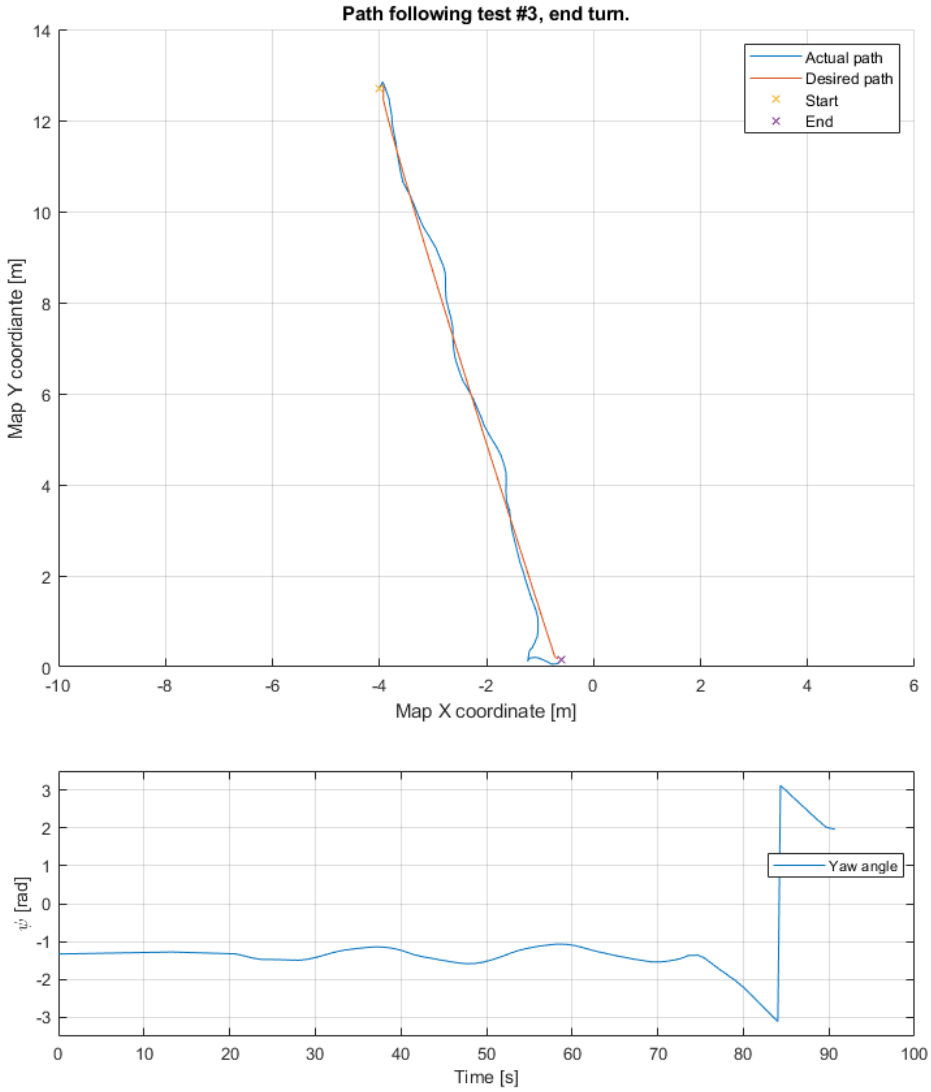


Figure 7.30: Pathfollowing test, 180°after driving straight

In the third test the goal was to turn when arriving at the goal. As one can see from figure 7.30 it followed the path closely with some oscillations, before driving out to the side and reversing into the goal to get the correct orientation.

This does make it deviate a bit from the global path, making it travel a total of 14.173m when the global paths length was 13.078 which makes a difference of 1.095m.

What one can deduce from these tests is that the prototype does follow the given paths quite well, but when it has to turn before / after it has to do some extra maneuvers which automatically makes it deviate a bit from the path.

7.9.2 Drift Test

These drift-tests were taken within minutes of each other to ensure that the weather and situation was as close to each other as possible.

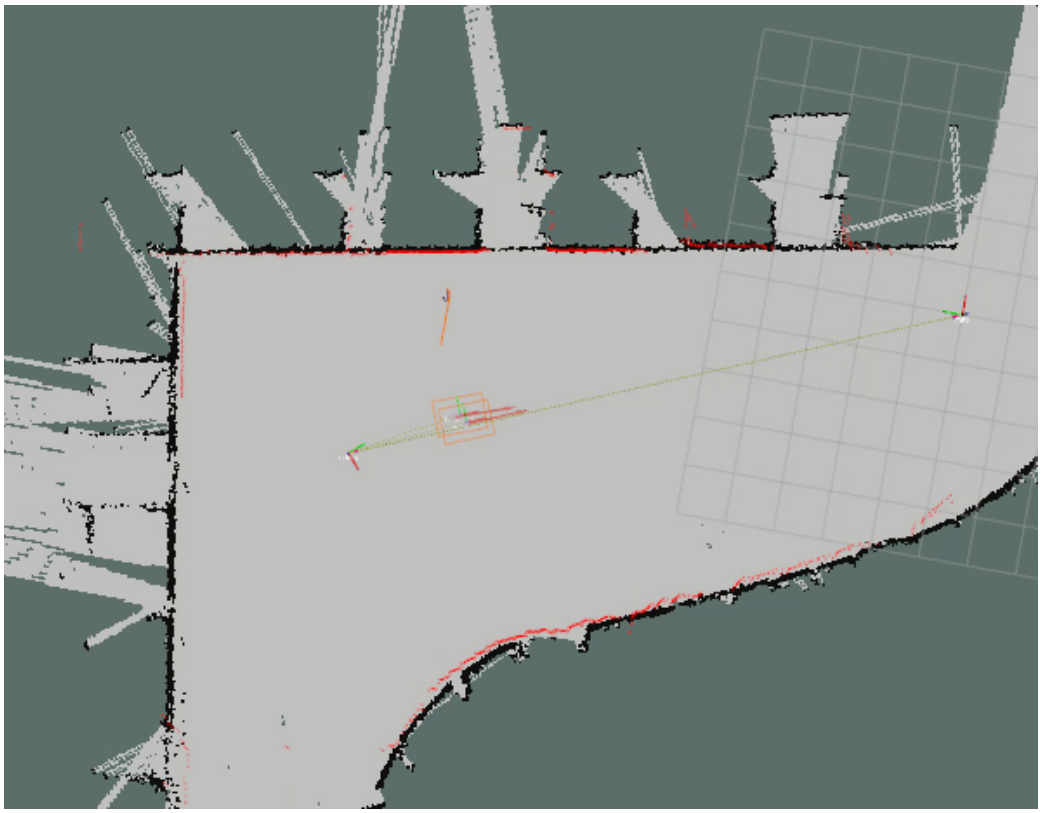


Figure 7.31: Drifting from wind / water with sensor feedback

This first test shown in figure 7.31 was with sensor feedback for the kalman filter. It had a total runtime of 47 seconds where it was drifting freely.

After this time the total distance drifted was 0.288m and an angular drift of -0.0545rad.

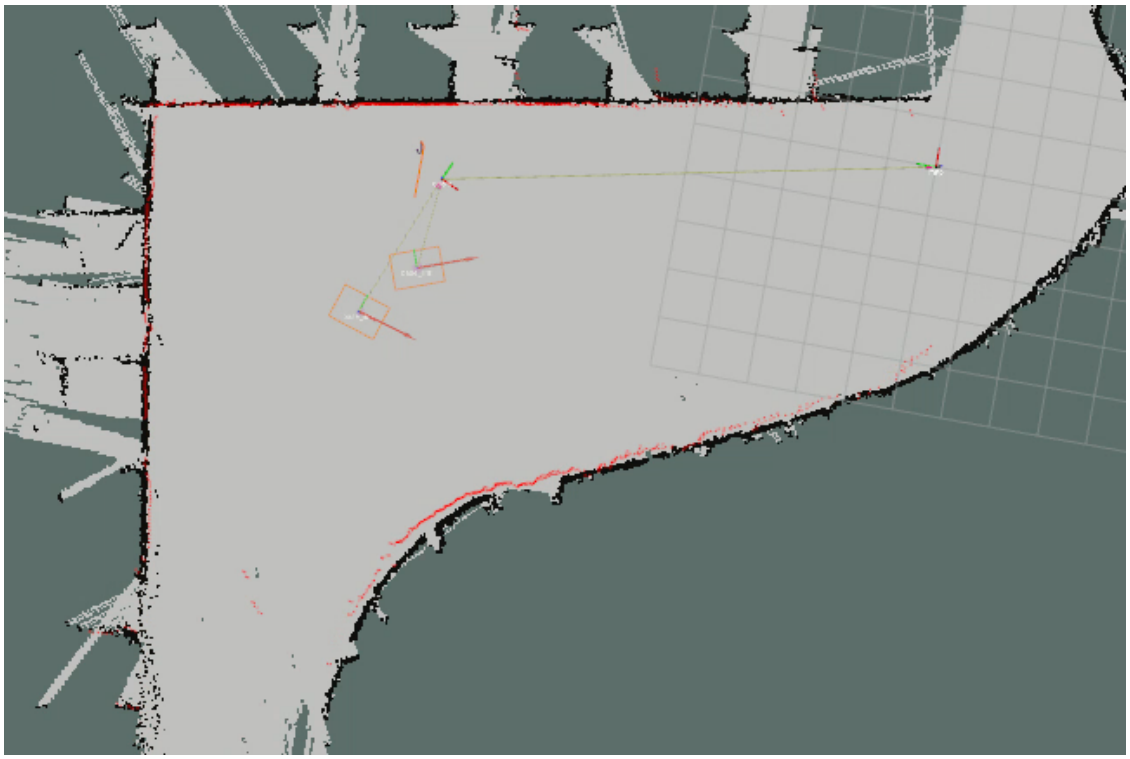


Figure 7.32: Drifting from wind / water with open loop observer

During the second test shown in figure 7.32 the system ran with an open loop observer. It had a total runtime of 56 seconds where it was drifting freely.

After this time the total distance drifted was 1.657m and an angular drift of -0.696rad.

When comparing the open loop drift to the closed loop drift there is no doubt that even though the system regulates velocity and not position, it is quite good at counteracting the influence wind and water has.

7.9.3 Obstacle Avoidance

The testing of obstacle avoidance shows what is touched upon in section 7.8. When approaching dead on the local planner has a lot of trouble managing to create a path around the object without changing its mind all the time. And if approaching more to one side than the other it manages quite fine.

During the testing of this two main tests were done.

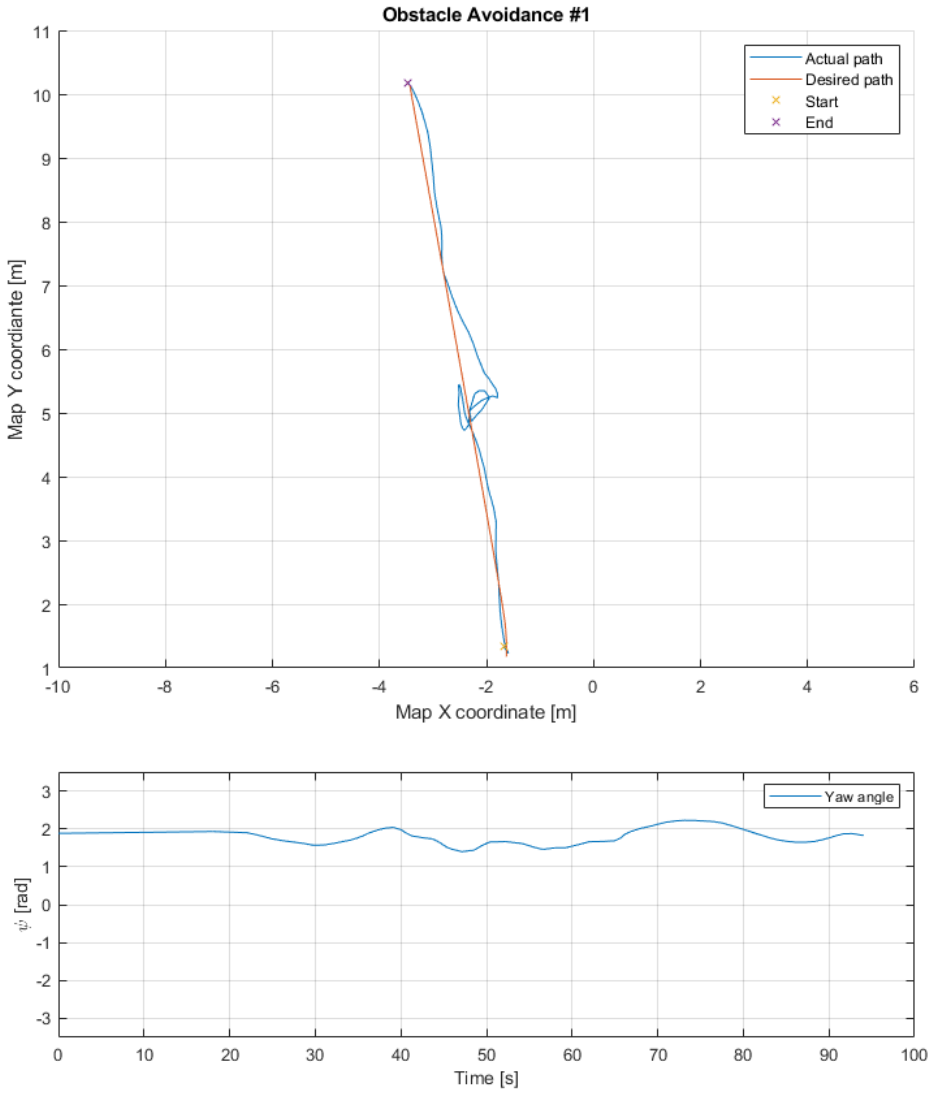
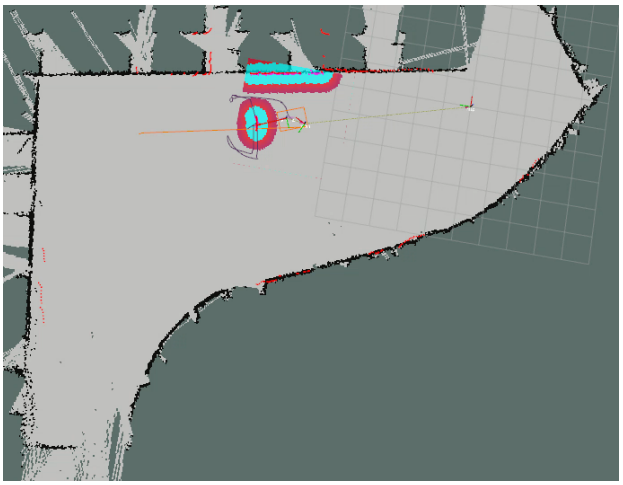
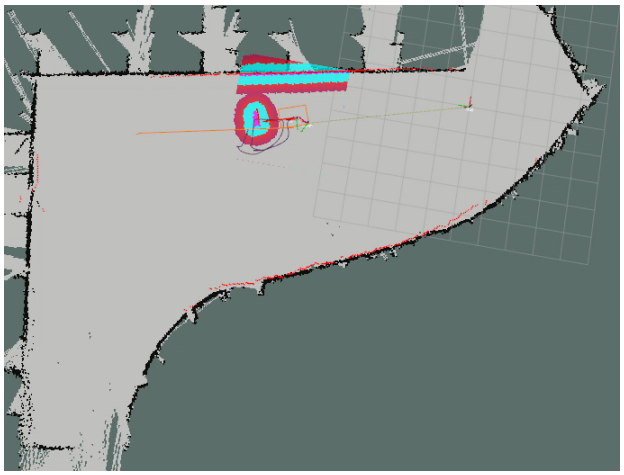


Figure 7.33: Path during obstacle avoidance run #1

(a) Obstacle Avoidance t = 0s



(b) Obstacle Avoidance t = 10s



(c) Obstacle Avoidance t = 20s

Figure 7.34: Obstacle avoidance, prototype approaching dead on, run #1

During this first run the prototype approached the obstacle dead on. As can be seen both from the path in figure 7.33 and the time stamps in figure 7.34 the local planner changes its mind all the time.

As a result of this the prototype just strafes in front of the obstacle for quite some time before managing to get around it slowly.

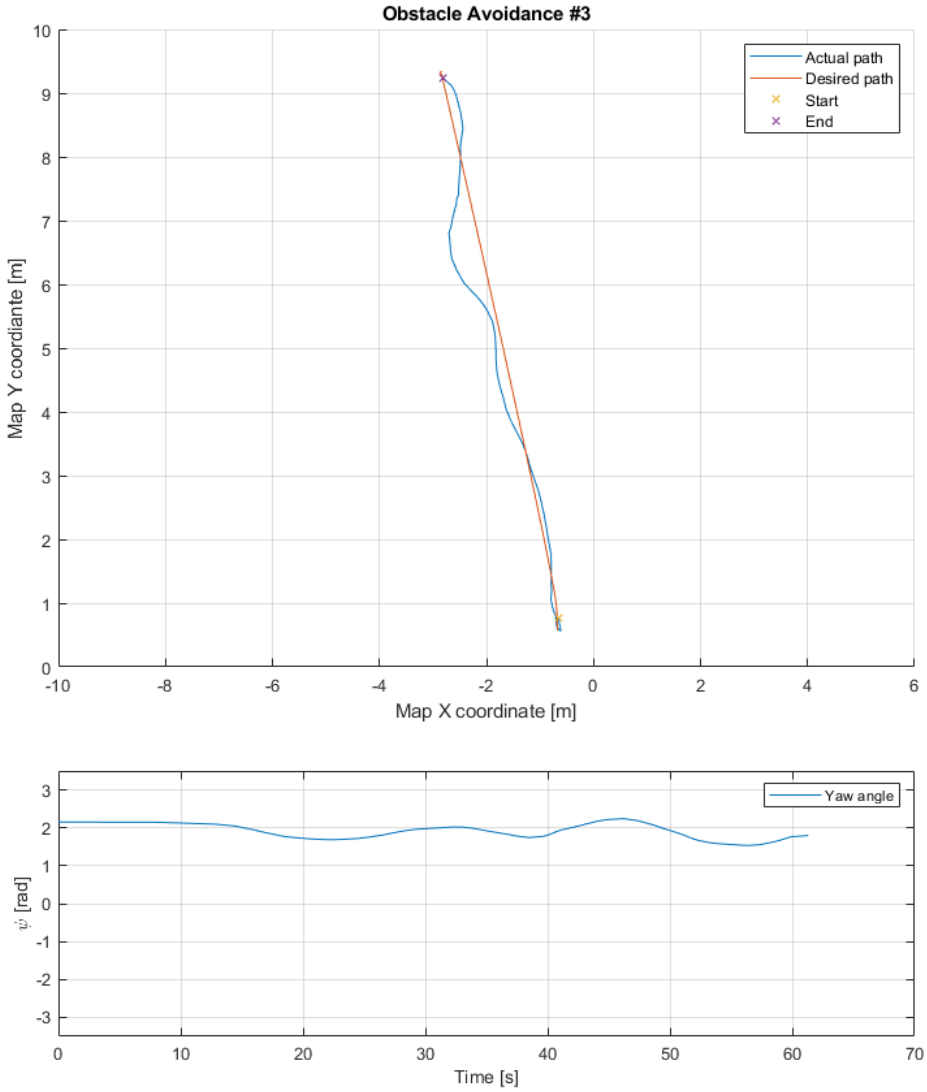
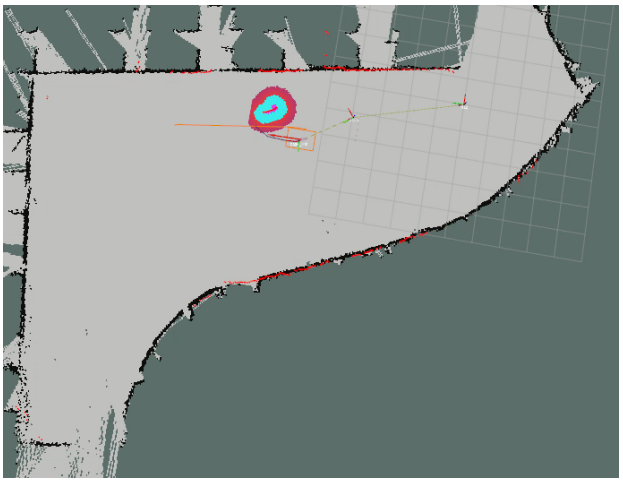


Figure 7.35: Path during obstacle avoidance run #2

(a) Obstacle Avoidance t = 0s



(b) Obstacle Avoidance t = 10s

(c) Obstacle Avoidance t = 20s

Figure 7.36: Obstacle avoidance, prototype approaching more to the left

During this second run the prototype approached the obstacle more from one side than the other, and this made it relatively easy for it to make a path around the obstacle as seen in figure 7.35. This is also reflected in the RVIZ-footage in figure 7.36.

7.9.4 Docking Test

These docking tests were done around the same time to ensure the testing environment was as close to each other as possible. All of them were done with closed looped feedback.

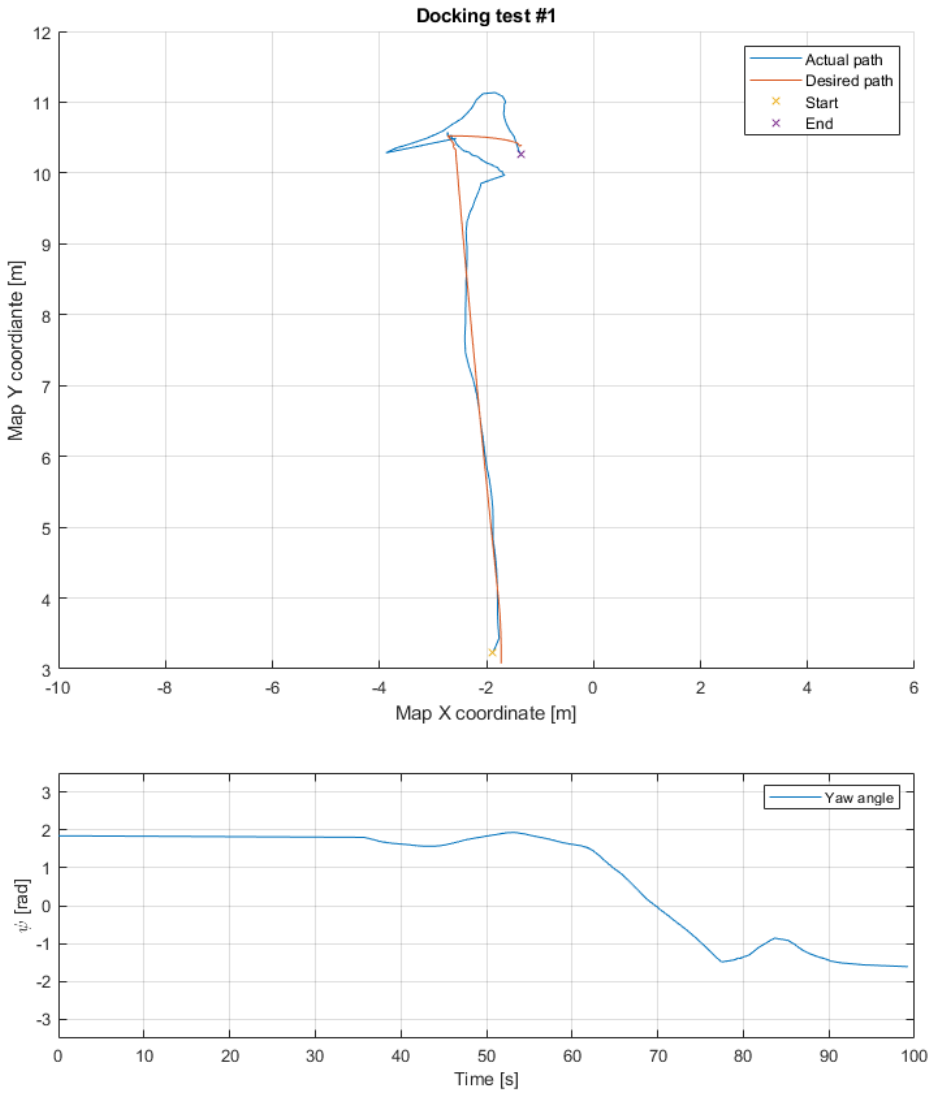


Figure 7.37: Path during docking test #1

During the first docking test, the AMCL lost track of its position and became uncertain, as can be seen in figure 7.38 represented by the purple oval. This caused the navigation to become unstable and it started going a bit haywire as it thought it was closer to the dock than it really

was.

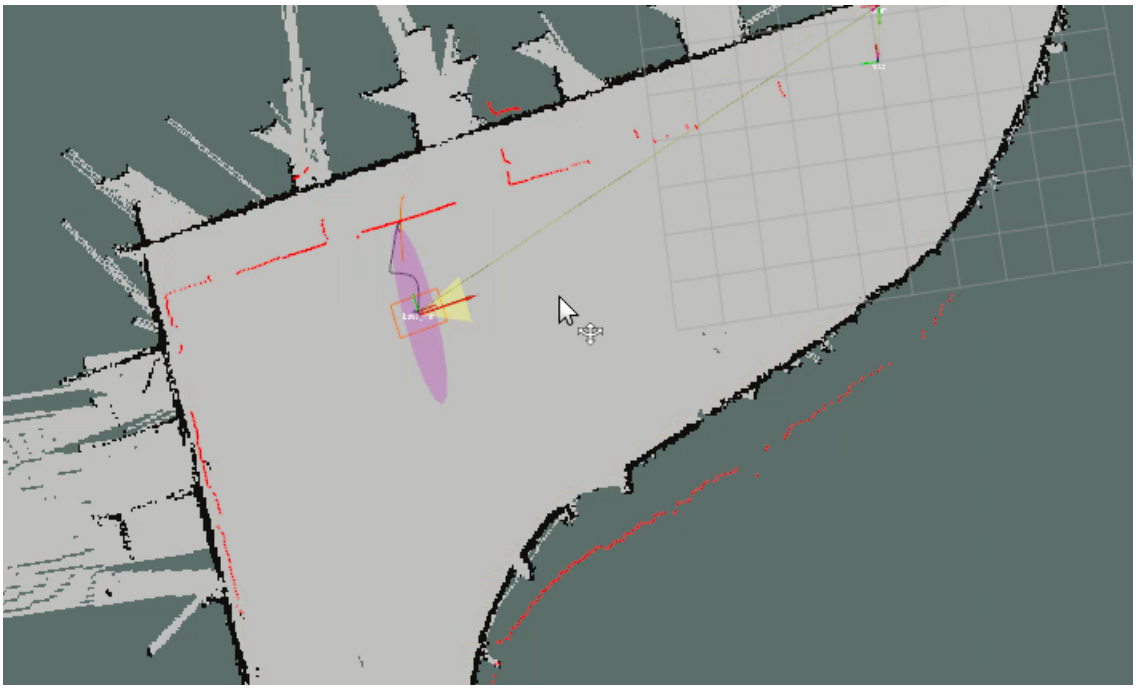


Figure 7.38: Docking test 1, RVIZ-footage

But after a while it stabilized and found its position again which enabled it to get to the final position as desired.

The first spike one can see in figure 7.37 when the prototype reaches the position at [-2.5, 10.5] is because the orientation of the first goal is 180° turned from what it approached the goal in. So it goes in for a turn while approaching the goal. This is to line itself up with the dock to shift sideways in.

In total during this first test the prototype traveled a total of 13.560m while the global path had a planned length of 8.999m which makes a difference of 4.561m. This is quite a lot, but when considering that this was with a loss of position, it is not terrible as it managed to recover.

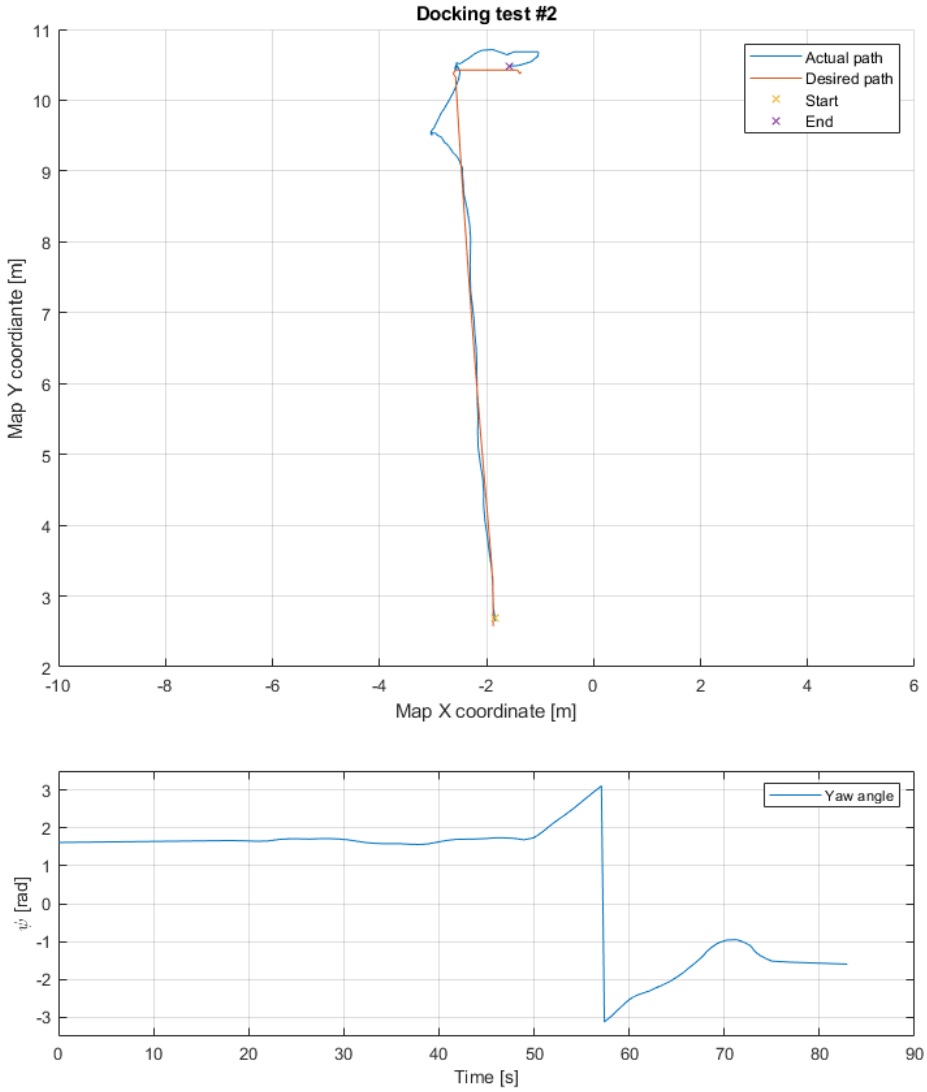


Figure 7.39: Path during docking test #2

During this second docking test one can see in figure 7.39 that the prototype had no problem following the path, and before reaching the first goal deviated a bit to turn itself 180° to line up with the dock. After this a bit of trouble with approaching the dock but managing to recover.

In this test the prototype traveled a total of 11.113m when the global paths planned length was 9.181m. This makes a difference of 1.932m, which isn't super but still not that bad when considering that some of this length is due to the maneuver to turn itself around.

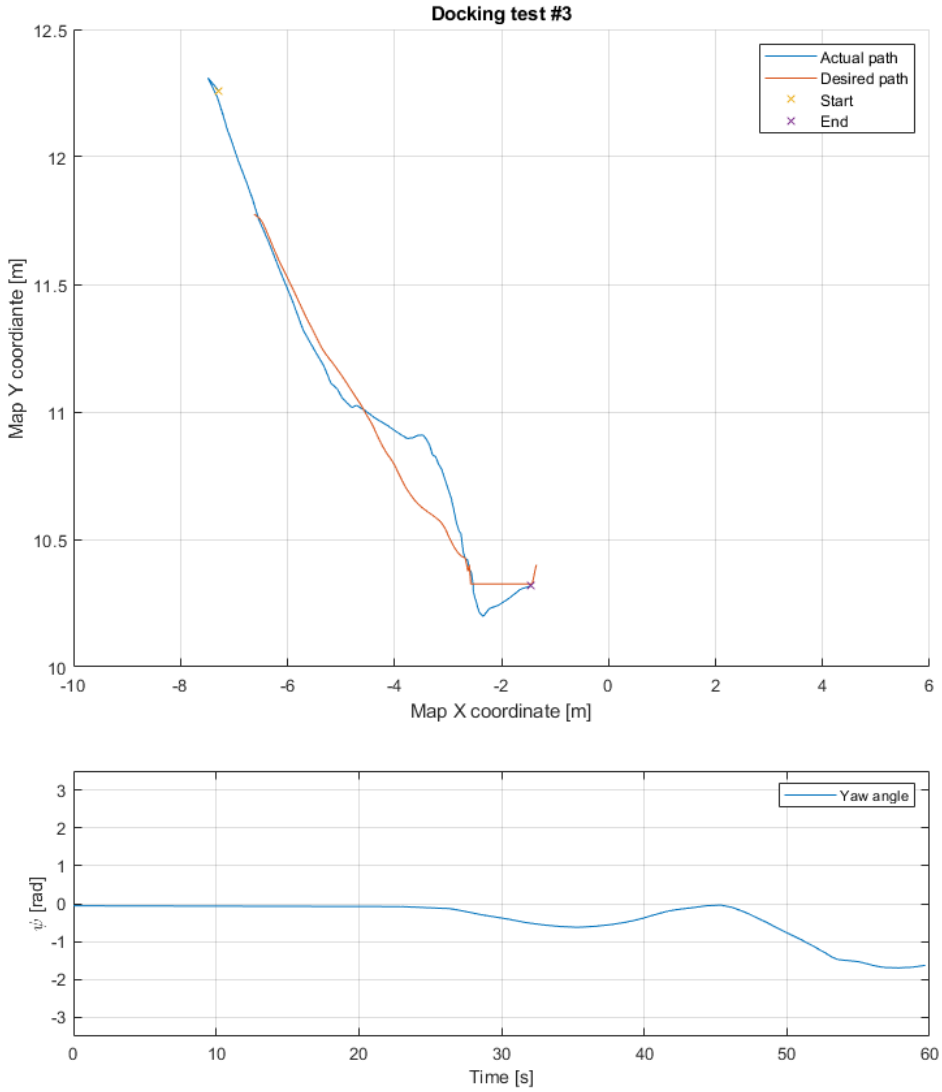


Figure 7.40: Path during docking test #3

This third docking test had a different starting position than the other two tests, as can be seen from the starting coordinate in figure 7.40. In this test since it approached the first goal from the other direction, it did not have to turn itself around to line itself up with the dock.

As one can see it deviated quite a bit from the path struggling to keep on the path, both before and after the first goal. This made the prototype travel a total length of 6.850m, with the global path length being 5.634m. This makes a difference of 1.216m which is quite a bit when

considering that it did not have to do a maneuver to turn itself around.

7.9.5 Open Loop Observer Test Results

When testing the strength of the model, and running the system with an open loop observer for the feedback to the system, one can see that the behavior of the prototype has bigger yaw oscillations than when it is running with sensor feedback.

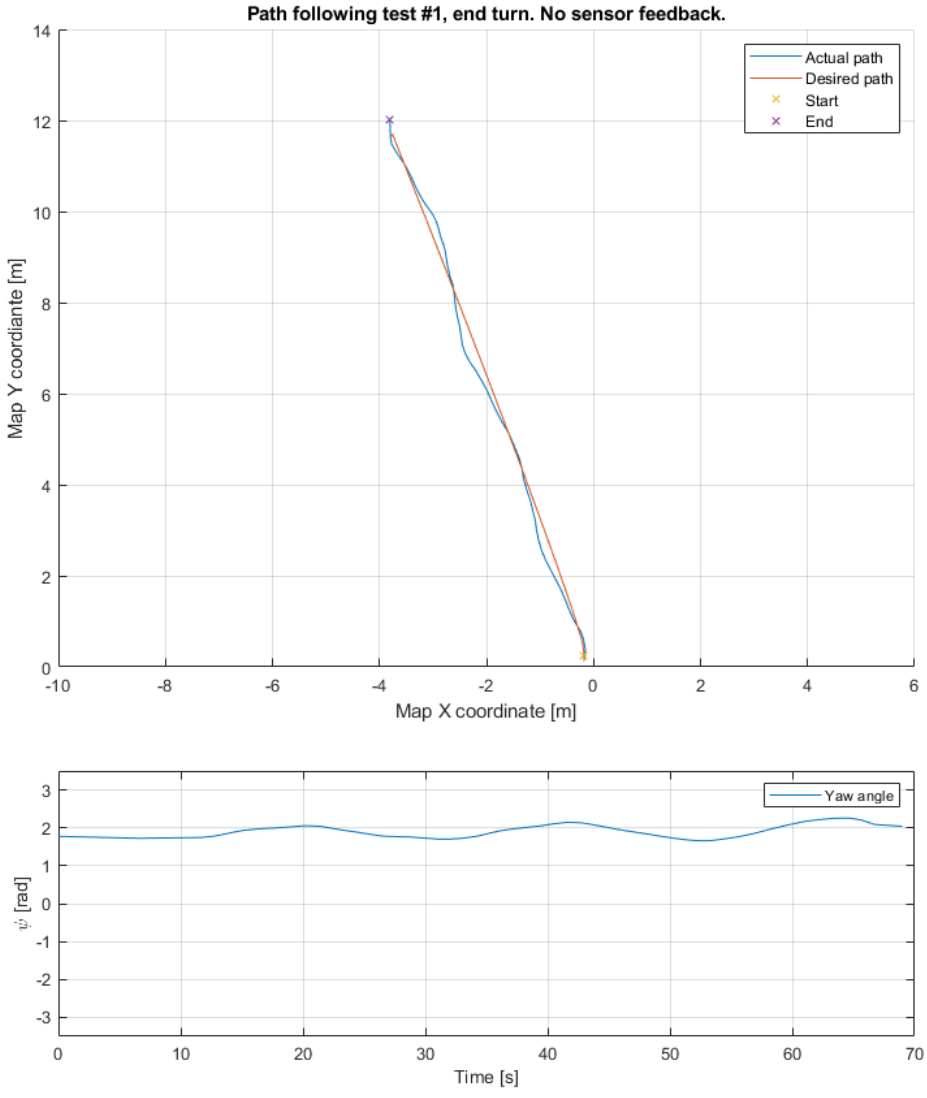


Figure 7.41: Path following test, straight line with open loop observer

In figure 7.41 it can be seen that the yaw oscillations are bigger than the test with the sensor feedback. But it also shows that it still follows the path quite well, and shows that the model for the system is quite spot on. The travelled length of the prototype was 12.581m when the global paths length was 12.165m.

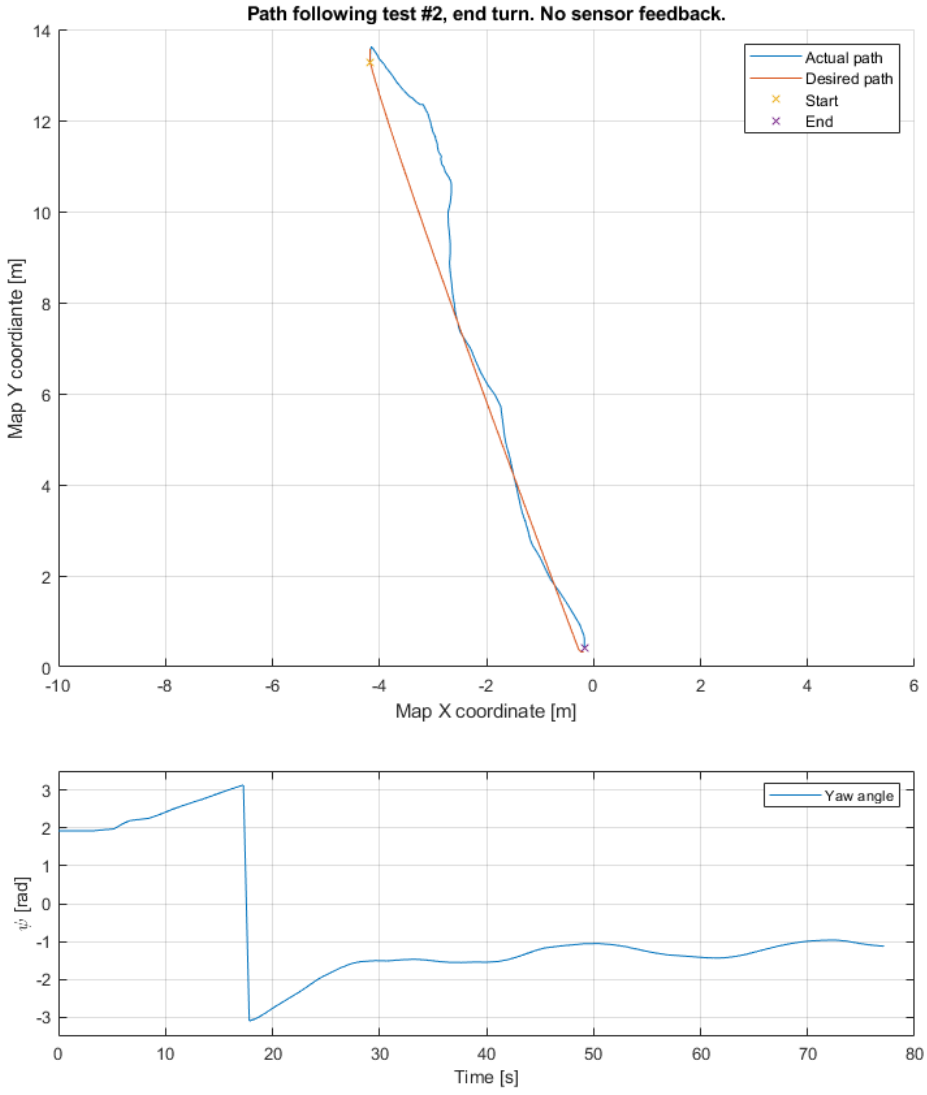


Figure 7.42: Pathfollowing test, turn at start with open loop observer

In figure 7.42 one can see much of the same behavior as in the same test with sensor feedback. A big deviation in the start before it starts honing in on the path again. It oscillates quite a

bit here as well, but all in all it shows that the system works well even with an open loop observer. The travelled length of the prototype was 14.501m when the global paths length was 13.893m.

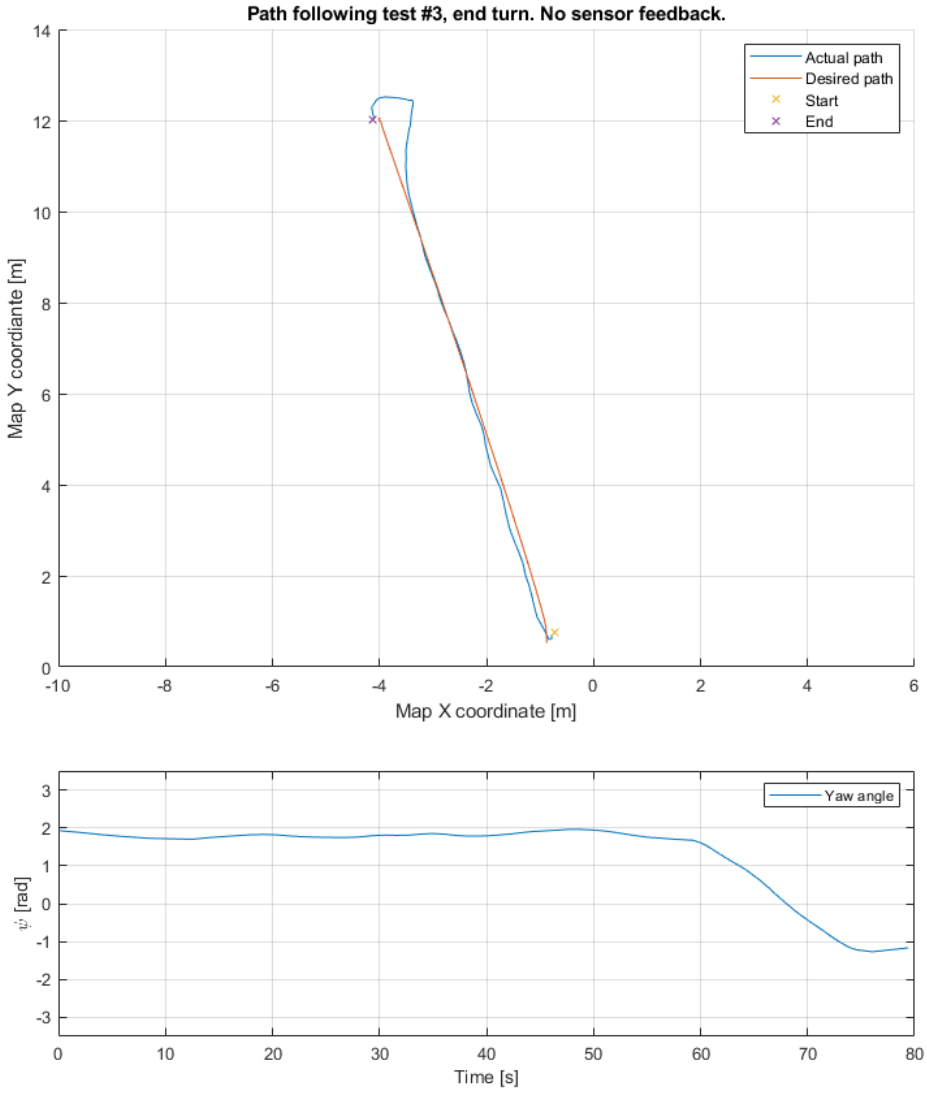


Figure 7.43: Pathfollowing test, turn at end with open loop observer

This last pathfollowing test displayed in figure 7.43 is the one that really shows the strength of the model. Here the prototype traveled a total of 13.6428m where the global path had planned 11.9935m. The difference comes down to the fact that the prototype had to turn to reach the right orientation for the goal. With this close of a margin, it really shows that the model for the

system is quite accurate.

There were also docking tests run with an open loop observer, to see how the system would cope with several goals and a more complex path.

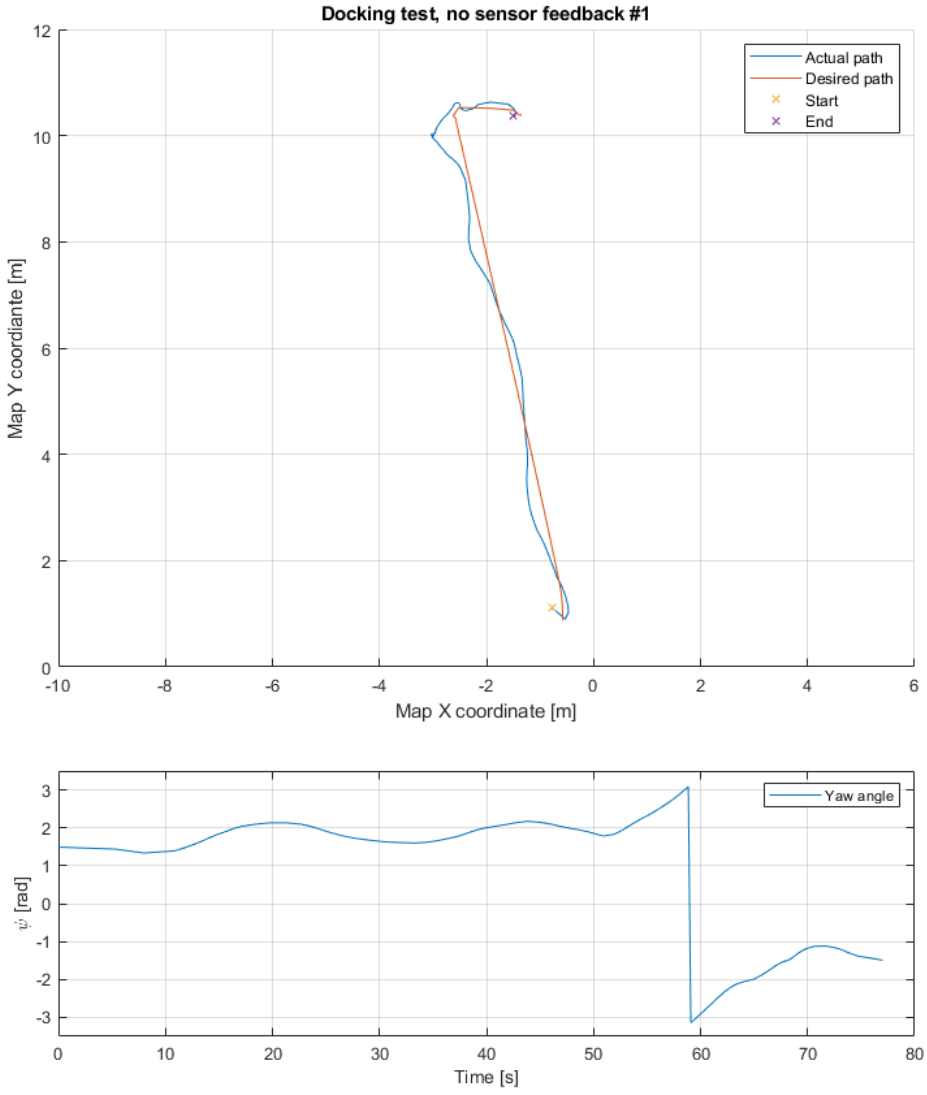


Figure 7.44: Docking run 1, open loop observer

During this first docking test one can clearly see more oscillations in figure 7.44 than in the original docking tests done and described in section 7.9.4. It has the same deviation when clos-

ing in on the first goal where it does the maneuver to turn itself around.

It manages to lateral shift in to the final goal quite well without any big deviations. In this test the prototype traveled a total of 12.431m when the global plans length was 11.147m which makes a difference of 1.284m. This is not bad for an open loop observer.

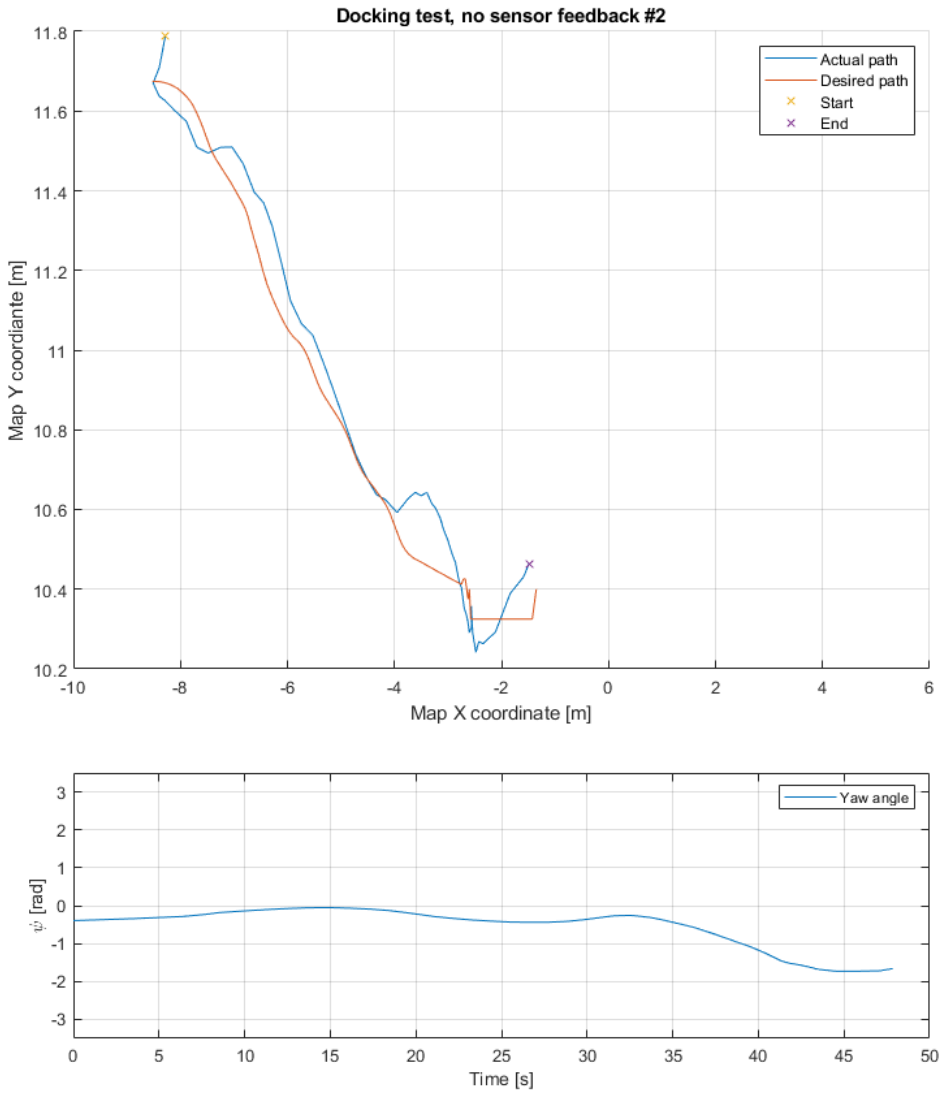


Figure 7.45: Docking run 2, open loop observer

This second test is from the second starting position, where it does not have to do a ma-

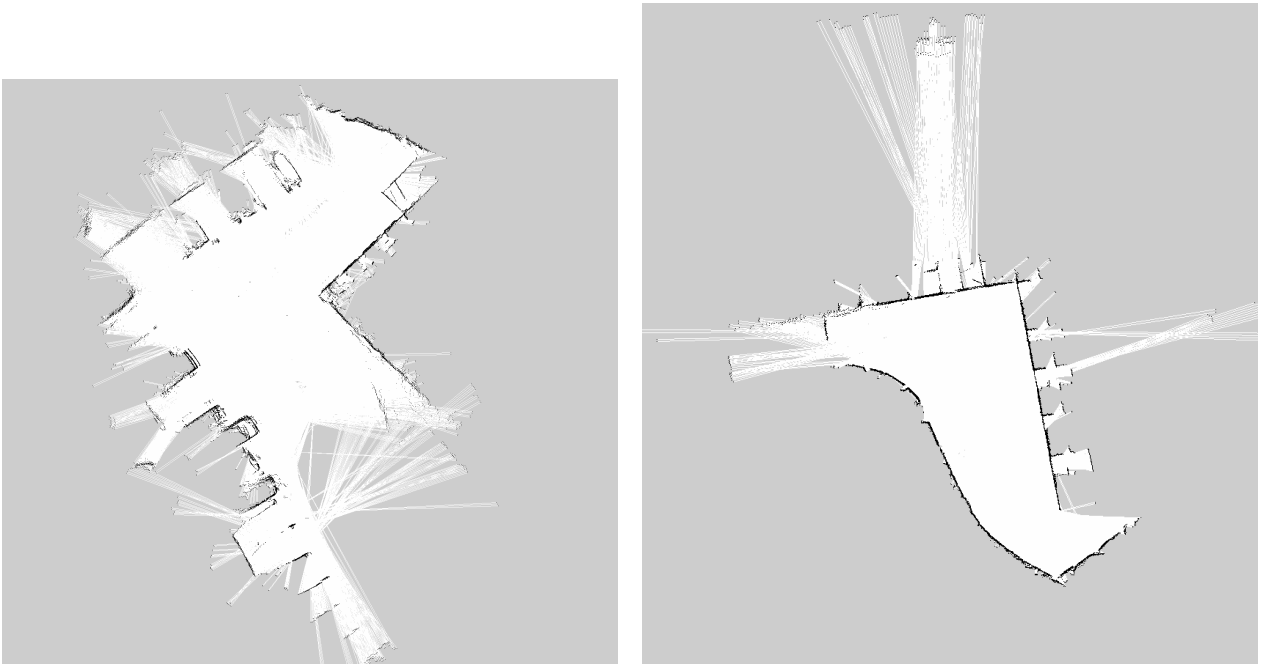
neuver to turn itself around. When observing the path in figure 7.45 one can clearly see that it is struggling in this test. It keeps following the path parallel instead of on it, which might be a result of the localization being unsure.

There are a lot of oscillations and deviations from the path. This makes the prototype travel a total of 7.715m when the global plans length was 7.469m which makes a difference of 0.246m. This is not that bad, but when one takes into account that the path is noticeably shorter than the other docking attempts, it's still not great.

But these tests do show that the model is quite spot on and works well even as an open loop observer.

7.9.6 Mapping and Localization in Bigger Environment

When trying to map and localize in a bigger environment one thing that is quite clear when comparing it to a map from a smaller area in figure 7.46 is that there is a bigger uncertainty of what is a static obstacle and its positioning.

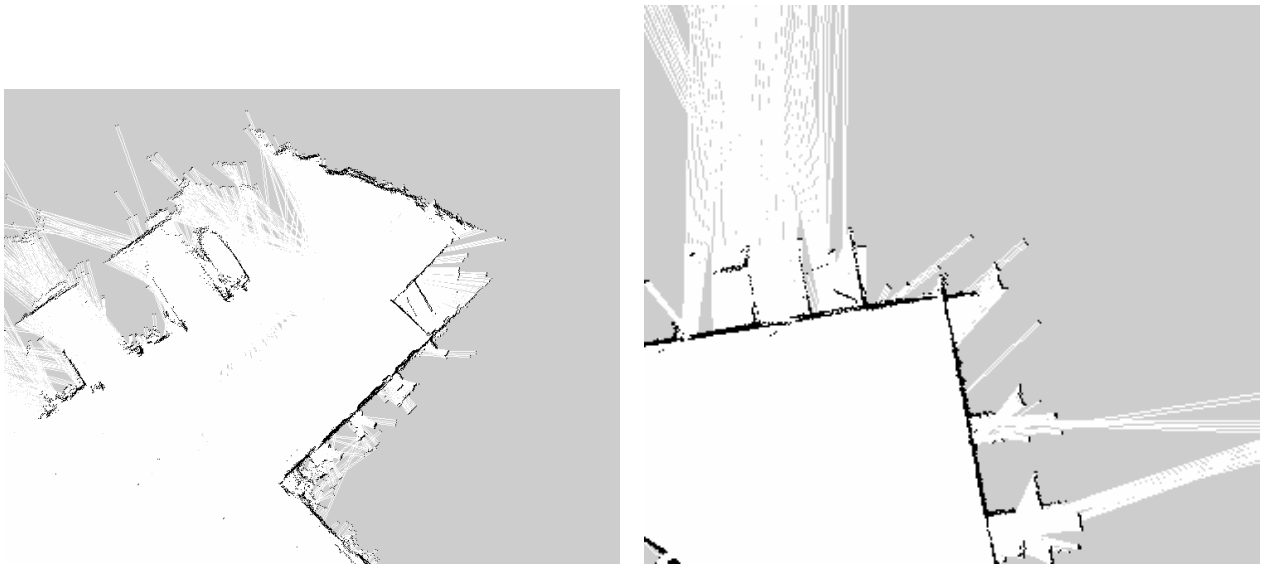


(a) Map from bigger environment (b) Main map used during performance testing

Figure 7.46: Comparison of map from smaller area and bigger area

This is because of the area being quite a lot bigger and therefore there are fewer surfaces for the lidar to find with its limited range. It's especially hard when the only reference the lidar has is a straight surface because it only has a reference for one direction and not both.

When zooming in and comparing the walls one can see the big difference between a map created with a lot of reference surfaces and a map created with fewer references.



(a) Zoomed in map from bigger enviroment (b) Zoomed in map from performance testing

Figure 7.47: Comparison of map from smaller area and bigger area 2

From figure 7.47 the walls on the bigger map in figure 7.47a are less well-defined and there are a lot more small black dots all over the map around the walls, where in the map from the smaller area in figure 7.47b the walls are well-defined and there is almost no uncertainty in where the walls are placed.

Chapter 8

Discussion

In this chapter we will discuss the process of this project as well as all choices we have made. We will discuss what we think of the choices now and if it might have been done differently and in so what could have been changed. We will also discuss how the process went and what we learned from it.

8.1 Design Reviews

One thing we've learned a lot through this project is that even though it's good to just get started on creating a prototype, it's a time saver to use some extra time to actually get some more knowledge in fields where we're uneducated.

In our case the only thing we knew about building a boat was the basic formula of how buoyancy makes things float. We did not have any knowledge about the stability of a boat, and should have sought out someone who knew more about this earlier.

But after each failed attempt, new knowledge was acquired quickly, so in that aspect we did learn quickly and got the prototype on the water within a good time frame.

8.2 Test results

8.2.1 Preliminary Results

The preliminary tests were tests that were done quickly to determine if the iteration of the prototype was good enough to be worked further on. We did not log any data from these tests because we decided it was no point in doing it since it's just to ensure that the prototype was able to be used.

Thus the results of these tests were just testimony from us, and the experiences we got from these tests. We adjusted from them, and got a lot of experience from these tests.

8.2.2 Navigation Stack Testing

The TEB Local planner was by far the best planner for this prototype. It was good at path following and choosing the right kind of movement at the right time. But TEB's characteristic 3-point turn was probably not the best maneuver for a seaborne vessel. A pure rotation would have been more logical for our prototype.

By far the biggest problem with TEB was the heavy processing power required to operate. This was shown to be a problem when trying to deviate from the original path to avoid obstacles. Its one of the major limitations with using TEB, even with a powerful laptop we were not able to give it the processing resources it needed to run optimally.

But even still TEB was all-in-all the best choice for this project and could probably have been even better if there was more time to learn all the parameters it has to offer in terms of tweaking performance for specific needs. TEB has the potential to do a better job if given a dedicated processor and finding a more suitable setup for the different parameters.

8.2.3 Mapping and Localization in Bigger Environment

Trying to map and localize in a bigger environment than we originally had done was not a success. We quickly realized that without a secondary odometry source it took very little before the SLAM-algorithm lost track of itself and the whole map started drifting.

For us this was a result of the Lidar not having the range for this. With a better Lidar it might have been very possible to do this without any big challenges. So it should be noted that even though we weren't able to map in such an area, that it is very well possible with a stronger Lidar, and with a 3D Lidar it might have been even easier without having a much bigger range.

8.3 Software

8.3.1 Choices Made

We chose to implement ROS for this project because we'd heard it was easy to use and made it easy to create new packages and communicate between nodes. This is mostly true, but it should be noted that some of the packages like the navigation stack are easy to implement but significantly harder to tweak properly.

There is a lot of documentation for ROS as well, but when it comes to specifics like what information is sent and how it is sent it's more hit-and-miss. Some packages have extremely well-documented features while others are more black-box where it seems like even the creators aren't sure what impact the different parameters have.

We chose to make this system without a GPS so it could be a system independent from that. We see now that if we had chosen to implement the system with a GPS, we could have used it to supplement information for the system when it worked. It would also have been possible to make the system work fine without the GPS if it ever fell out.

If we had done this we probably would've been able to map and localize in a bigger environ-

ment like we tried in section 7.9.6.

For localization it was chosen to use AMCL instead of SLAM, because of the fact that AMCL uses the same map every time and a given coordinate is always at the same place. We chose this specifically so we could save the coordinates for the dock, and send the same coordinates via a script every time.

The reasoning for this is that we wanted to automate the process of docking instead of having to manually plot in a coordinate every time.

8.3.2 Final Solution

The final solution for the software was fine. There is a lot of room for improvement in how optimized it is, both in the way one starts up the software and in the way it runs on a code-level.

As discussed in several chapters of the report, parts of the program were very process-heavy and required a lot of power. This created problems especially for the local planner. A solution to this might have been to split up parts of the program over either more units or over more processors.

When considering that this is a system that basically navigates and localizes using only a single IMU and a Lidar, it is a fine solution to the problem presented. While this is also one of the systems drawbacks in that it can't navigate bigger areas, it works well in areas with a lot of close range surfaces for the lidar to find.

The map we used during testing was made during high tide. We saw that when we used it throughout the project and the water was at low tide it created problems with the references not being the same anymore, therefore making the AMCL lose itself a bit more than normal. This can be seen in both docking test 1 as well as docking test 3 in section 7.9.4.

We see that the lack of a secondary odometry source makes the system less robust in that if

the system loses itself in regards to what information the lidar has, there is no other part of the system that can pick up the slack. If we had a more precise and reliable IMU, this could have been used as a secondary odometry source.

8.4 Sensors

The sensors chosen for this robot worked well for keeping track of the robots position in the environment, but there was a few shortcomings. First and foremost the lidar was not powerful enough to be used outdoor efficiently. This resulted in bad performance when operating in a bigger area.

For measuring orientation the filter combining IMU-data did a good job of registering change in orientation, but the drift in yaw made it so it was not useable by itself and needed information from the lidar to compensate for this. A better IMU would probably have improved on this even though it was good enough when combined with the lidar-data.

8.5 System Identification

We used a genetic algorithm in order to determine the parameters of a state space model for our prototype, based on a simplification of T. I. Fossen's model for vessel dynamics. I all honesty, this approach worked out much better than expected. The system model performed pretty well despite heavy simplifications, like ignoring certain parameters considering ocean currents and hull shape. It remains an unanswered question whether these simplifications would cause the model to fail during harsher conditions than the prototype was tested in.

If we had more extensive knowledge about how to determine some of the model parameters, specifically the M_A matrix coefficients, and access to reliable state measurements instead of estimations from other data, we think that the GA could have performed even better. As such we think that using a GA for determining unknown variables for state space model can be a viable solution to the problem.

8.6 Prototype

8.6.1 Limitations

There was one main physical limitation of the prototype, which was how fast it could go before the tilting became to severe. We made the prototype like it is to reduce the impact waves would have, but in hindsight we see that in the area we worked waves were basically non-existent anyway.

We could therefore probably have made something that was more affected by waves without any huge problems surfacing from that. If we had done that, we could have made something that could go faster, which might have solved some of the other problems we had during the testing of obstacle avoidance and such.

8.6.2 Improvements

The boat design could probably have been much smaller. This could have drastically reduced the time we had to spend analysing and creating a prototype. In hindsight we see that the prototype was too big and of a sub-optimal design for this use. It would be more optimal with a small one, having said that the prototype as it is did its job well enough even when seeing its shortcomings.

The prototype became this big and heavy because of the design-choice which then set big limitations on how fast the prototype could travel. The design would have been more appropriate if we were going to use it more at open sea where there was a chance of meeting other boats and hitting waves. But for our use this design gave more of a drawback rather than a positive side considering we never met waves and the big limitation on speed gave trouble in other areas as well.

8.6.3 Prototype vs. Product

The goal of the project was to test a system, not create a product. Therefore for us the real solution is the software more than the actual physical prototype. The goal was to see how a Lidar would function in such a system.

We feel that this has been the focus all the way through, but to make the software work we had to put a lot of hours into actually creating a prototype that was well enough constructed to be able to put all the electrical components onto it and safely deploy it to sea.

Because of this a lot of the focus on this thesis fell on how to actually create a stable prototype and not just a focus on the system itself and how to improve upon it. In hindsight it might have been smarter to try to get hold of a finished construction that we could mount our system to instead of creating a boat ourselves.

8.7 Personal Experiences

8.7.1 Extraordinary Situation (Corona)

The extraordinary situation COVID-19 put us in did not help the project, but to be fair we were aware of the chance that the school could be shutting down a week before it actually did. So we had a plan when we got the message that we had to leave the campus.

Because of this, the place where we worked changed but the way we worked did not change too much. We had a lot more safety precautions in place (washing of hands, no sharing of snacks etc.) to ensure that the chance of infection between members was as low as possible.

All in all we all agree that even though COVID-19 changed a lot, we feel the project was done almost as good as it could've been done even if the campus was open. We had most of the equipment we needed privately, and we were able to get all parts we needed as well.

8.7.2 Project Organization

Even though we elected people into specific positions, we feel that we managed to keep decision-making a democracy through the whole process. There were quite a lot of discussions, but they always ended with the right decision and ensured the best product.

The responsibilites the elected persons got were held up with no problem, the group leader took responsibility for communication with the supervisors when needed and the secretary held up the responsiblities of documenting the meetings held without any problem.

8.7.3 Work Method

The actual work method worked quite fine. During the project before the corona-situation, meeting at school at the same time and working together there worked very well. The effectiveness was quite high and a lot got done quickly.

After the corona-situation came and everything changed the work method also changed, and this new method also worked quite fine. But the effectiveness did go down a bit, but not too much. It did make it harder to cooperate properly, but it worked fine considering everything.

It did introduce a bit more time to travel when deciding to go out to test the prototype, which was a bit of a hassle. Some of the members needed up to 30-45minutes to reach the boat house for testing, so it wasn't always too easy to get there quickly and get started if the weather suddenly got better.

Most of the time we worked in groups of 2 which was quite fine, and allowed us to get a reality-check whenever we needed it. And all of us know from earlier project work that working in pairs often increase the effectiveness, especially when programming. This is because one often gets stuck on a problem, and without some sort of feedback from others it's not easy to find the solution.

All in all we're quite happy with our work method, but we see that we could've been more effective in our work, but most of this is because of the situation we came into when COVID-19 came to the country.

8.7.4 Knowledge Acquired During Project

We've learned a lot during this project. One of the big things we've learned is that we need to seek out help from people with more knowledge than ourselves more often, and quicker when we're approaching subjects we have no knowledge on, like building a stable boat.

If we'd sought out Håvard Lien earlier we might have saved both money and time not building prototypes that did not work at all. But in the end it all worked out, but we've certainly learned that we should seek out knowledge earlier in the process if we're in doubt.

We've also learned a lot about the division of work between group members. It's hard to say right at the start how long some parts of the project will take, and therefore some of the work has to be given out at later stages.

A thing that we all learned a lot from was the report we wrote at the start of the semester to get ready to do this project, the pre-project report. We've all seen that doing this helped a lot during the thesis when we were in doubt of how to do things.

Chapter 9

Conclusions

The goal of this project has been to create a prototype with a system that localizes itself and navigates using a Lidar, without using a GPS. This goal has been reached and tested properly and it navigates and docks autonomously. It does have some drawbacks as discussed in the earlier chapters like the fact that the Lidar does need a secondary odometry source to help it when it does not have enough close-range surfaces to find.

Our experiences with the project is that this is a system that has great potential, but a lot of work needs to be done with it before it can actually be implemented on a bigger scale and a Lidar with a bigger range and a better mode for outdoor-use is needed.

As we also have seen from the test of the obstacle avoidance, there is a need for more processing power for such a complex system like this. The laptop used for the processing couldn't handle all the processing alone, and the same is true for the Raspberry Pi 4.

As discussed the design of the prototype was not the best choice for the use in this project, but if it were to be worked on further, the current design would make it possible to use on bigger areas where waves and rougher seas can occur without having to make any changes to it.

Some rework of the design could be beneficial however, as the current design suffers from pitch motion when moving faster than 0.2m/s which is undesired. This could possibly be coun-

teracted by an active ballast system.

9.1 Further work

There are a few things to be worked on and improve,

- Implement a secondary odometry source to help the lidar
- Expand the GUI and finalize TCP-communication between GUI and system
- Improve optimization of the program
- Improve and fix the obstacle avoidance
- Implement active ballast regulation system

Bibliography

[1] 3dsam79. Oil rig 1. URL https://www.turbosquid.com/3d-models/3d-model-oil-rig-platform-1/802663. visited 18.05.2020.

[2] RasPi.TV Alex. How much power does the pi4b use? power measurements, June 2019. URL https://raspi.tv/2019/how-much-power-does-the-pi4b-use-power-measurements. Visited 06.05.2020.

[3] Marco Altosole. Definitions of reference frames and ship motions (surge, sway, heave, roll, pitch and yaw), February 2009. URL https://www.researchgate.net/figure/Definitions-of-reference-frames-and-ship-motions-surge-sway-heave-roll-pitch-and-ya fig1_245386997. Visited 10.05.2020.

[4] Antratek. Rplidar a3m1 360° laser range scanner - 25m range. URL https://www.antratek.com/rplidar-a3m1-360-laser-range-scanner. Visited 14.05.2020.

[5] Arduino. Arduino nano - tech specs. URL https://store.arduino.cc/arduino-nano. Visited 31.01.2020.

[6] Basicmicro. roboclaw datatshet. URL http://downloads.basicmicro.com/docs/roboclaw_datasheet_2x30A.pdf. Visited 12.05.2020.

[7] Basicmicro. Roboclaw 2x30a, 34vdc dual channel brushed dc motor controller, 2016. URL http://downloads.basicmicro.com/docs/roboclaw_datasheet_2x30A.pdf. Visited 06.05.2020.

[8] Alex Becker. About kalman filter. URL https://www.kalmanfilter.net/default.aspx.

[9] OBO BETTERMANN. Koblingsboks obo t250 ip66, . URL https://www.elektroimportoren.no/koblingsboks-obo-t250-ip66/1278235/Product.html?Event=livesearch. Visited 06.05.2020.

[10] OBO BETTERMANN. Koblingsboks obo t350 ip66, . URL https://www.elektroimportoren.no/koblingsboks-obo-t350-ip66/1278236/Product.html. Visited 06.05.2020.

[11] Biltema. Mc-batteri gel 12v 20ah, . URL https://www.biltema.no/bil---mc/mc/mc-deler/mc-batterier/mc-batteri-gel-12v-20ah-2000029892.

[12] Biltema. Koplingsboks ip67 140x230x92, . URL https://www.biltema.no/bygg/elinstallasjoner/elbokser/koplingsboks-2000034324. Visited 10.05.2020.

[13] Greg Welch; Gary Bishop. An introduction to the kalman filter, 2006.

[14] Robin T. Bye. Slides about ga in the course ie303412 cybernetics, 2019. autumn semester 2019, NTNU.

[15] BYJUS. Archimedes principle. URL https://byjus.com/physics/archimedes-principle/. Visited 15.02.2020.

[16] Ion Motion Control. Roboclaw 2x30a motor controller, 2 channel, 30amps per channel, 6-34vdc. URL https://www.amazon.com/RoboClaw-Controller-Channel-30Amps-6-34VDC/dp/B00R1LFTZ2. Visited 14.05.2020.

[17] D-Link. Wireless ac750 dual band router, November 2019. URL https://eu.dlink.com/uk/en/-/media/consumer_products/dir/dir-809/datasheet/dir_809_b1_datasheet_en_eu.pdf. Visited 06.05.2020.

[18] Oxford English Dictionary. Robot - definition. URL https://www.oed.com/view/Entry/166641. Visited 24.01.2020.

[19] Elfa Distrelec. Pi4 model b/4gb & power supply & sd card - pakke med raspberry pi 4 1.5ghz quad-core og rnd-strømforsyning, 4gb ram. URL https://www.elfadistrelec.no/no/

pakke-med-raspberry-pi-5ghz-quad-core-og-rnd-stromforsyning-4gb-ram-raspberry-pi-pi
p/30158761. Visited 14.05.2020.

[20] drev.se. Kajakmotor - haswing w-20. URL https://www.drev.se/produkter/50735?ref=5500. Visited 31.01.2020.

[21] Open Source Robotics Foundation. Ros/introduction, . URL http://wiki.ros.org/ROS/Introduction. Visited 11.05.2020.

[22] Open Source Robotics Foundation. msg, . URL http://wiki.ros.org/msg. Visited 11.05.2020.

[23] Open Source Robotics Foundation. Ros/concepts, . URL http://wiki.ros.org/ROS/Concepts. Visited 18.05.2020.

[24] Open Source Robotics Foundation. Nodes, . URL http://wiki.ros.org/Nodes. Visited 11.05.2020.

[25] Open Source Robotics Foundation. Packages, . URL http://wiki.ros.org/Packages. Visited 11.05.2020.

[26] Raspberry Pi Foundation. Raspberry pi 4 tech specs, . URL https://www.raspberrypi.org/products/raspberry-pi-4-model-b/specifications/. Visited 31.01.2020.

[27] gotk. Problems with costmap and the inflation layer [closed], June 2015. URL https://answers.ros.org/question/211322/problems-with-costmap-and-the-inflation-layer/. forum post.

[28] John Griffiths. Built for the tough jobs, April 2020. URL https://discuss.bluerobotics.com/t/built-for-the-tough-jobs/7177. Visited 08.05.2020.

[29] ABB Group. Azipod xo, June 2012. URL https://library.e.abb.com/public/590ce0d16e7d72f5c1257a330027e777/Azipod_XO_Presentation.pdf. Visited 08.05.2020.

[30] Terje Moe Gustavsen. Ferjesektoren – i dag og framover, Mars 2019. URL https://www.vegvesen.no/fag/trafikk/ferje/ferjekonferansen.

[31] Avram Piltch; Gareth Halfacree. Raspberry pi 4 review: The gold standard for single-board computing, November 2019. URL https://www.tomshardware.com/reviews/raspberry-pi-4-b,6193.html. Visited 17.02.2020.

[32] Randy L. Haupt; Sue Ellen Haupt. Practical genetic algorithms, May 2003. Second Edition.

[33] Eitan Marder-Eppstein; David V. Lu; Dave Hershberger. costmap_2d, January 2018. URL http://wiki.ros.org/costmap_2d. Visited 08.05.2020.

[34] IEEE Hugh Durrant-Whyte, Fellow and Tim Bailey. Simultaneous localisation and mapping (slam): Part i the essential algorithms. URL https://people.eecs.berkeley.edu/~pabbeel/cs287-fa09/readings/Durrant-Whyte_Bailey_SLAM-tutorial-I.pdf.

[35] Kumar. Uart communication protocol – how it works?, July 2018. URL https://www.codrey.com/embedded-systems/uart-serial-communication-rs232/. Visited 04.05.2020.

[36] College Physics Lab. Measurements and error analysis. URL https://www.webassign.net/question_assets/unccolphysmechl1/measurements/manual.html. Visited 15.04.2020.

[37] CHRobotics LLC. Understanding quaternions. URL http://www.chrobotics.com/library/understanding-quaternions. Visited 15.05.2020.

[38] Scott Martin. What is simultaneous localization and mapping?, July 2019. URL https://blogs.nvidia.com/blog/2019/07/25/what-is-simultaneous-localization-and-mapping-nvidia-jetson-isaac-sdk/.

[39] Bradley Mitchell. Tcp vs. udp: The difference between these protocols, November 2019. URL https://www.lifewire.com/tcp-headers-and-udp-headers-explained-817970. Visited 17.02.2020.

[40] Norman S. Nise. Control systems engineering. Seventh Edition.

[41] NOAA. What are microplastics?, January 2020. URL https://oceanservice.noaa.gov/facts/microplastics.html. Visited 19.02.2020.

[42] National Oceanic and Atmospheric Administration (NOAA). What is lidar?, April 2020. URL https://oceanservice.noaa.gov/facts/lidar.html. Visited 04.05.2020.

[43] The Editors of Encyclopaedia Britannica. Reference frame, July 1998. URL https://www.britannica.com/science/reference-frame. Visited 15.05.2020.

[44] ToF-Insights Paul. Lidar and tof cameras – technologies explained, November 2018. URL https://tof-insights.com/time-of-flight-lidar-and-scanners-technologies-explained/. Visited 04.05.2020.

[45] Thor I. Fossen; Tristan Perez. Kalman filtering for positioning and heading control of ships and offshore rigs, December 2009. URL https://ieeexplore.ieee.org/document/5338560. Downloaded 03.2020.

[46] Eitan Marder-Eppstein; Eric Perko. base_local_planner, April 2019. URL http://wiki.ros.org/base_local_planner?distro=melodic. Visited 08.05.2020.

[47] Quanser. State space modeling. URL https://www.quanser.com/experience-controls/. visited 15.05.2020, it requires the use of the Quanser app to access.

[48] Rigzone. How do semisubmersibles work? URL https://www.rigzone.com/training/insight.asp?insight_id=338&c_id=24. Visited 12.05.2020.

[49] Chris Riley. Trim tabs – an explanation, July 2019. URL https://www.boatsafe.com/trim-tabs-explanation/. Visited 15.05.2020.

[50] Robothome. 12a dc-dc 5-40v to 1.2-36v step-down buck power supply module adjustable. URL https://www.ebay.com/itm/12A-DC-DC-5-40V-1-2-36V-Step-Down-Buck-Power-Supply-Module-Adjustable-/223566065975. Visited 31.01.2020.

[51] Sargun Sethi. What materials are used for building ships?, November 2019. URL https://www.marineinsight.com/guidelines/what-materials-are-used-for-building-ships/. Visited 19.02.2020.

[52] SFUptownMaker. I2c, May 2020. URL https://learn.sparkfun.com/tutorials/i2c/all. Visited 08.05.2020.

[53] Martin Nikolai Longva og Thor-Inge Nygård Simon Bårslett. Auto-docking of vessel. *Bacheloroppgave*, 130:65–66, June 2018.

[54] Slamtec. Rplidar a3, November 2019. URL http://bucket.download.slamtec.com/e6f98338018505f6c510e83f78f03b6abb35454b/LD310_SLAMTEC_rplidar_datasheet_A3M1_v1.7_en.pdf.

[55] STMicroelectronics. Lsm303dlhc - ultra compact high performance e-compass, 3d accelerometer and 3d magnetometer module, April 2011. URL https://cdn-shop.adafruit.com/datasheets/LSM303DLHC.PDF.

[56] STMicroelectronics. L3gd20 - mems motion sensor: Three-axis digital output gyroscope, August 2011. URL https://cdn-shop.adafruit.com/datasheets/L3GD20.pdf.

[57] Seeed Studio. Arduino nano v3. URL https://www.seeedstudio.com/Arduino-Nano-v3-p-1928.html. Visited 14.05.2020.

[58] Arizona State University. Fluid mechanics. URL https://www.asu.edu/courses/kin335/documents/Fluid%20mechanics.pdf. Visited 24.02.2020.

[59] Arturo Urquizo. Controlador pid en lazo cerrado, October 2008. URL https://commons.wikimedia.org/wiki/File:PID.svg#metadata. Visited 14.05.2020.

[60] Statens Vegvesen. Ferjedatabanken. URL http://fdb.triona.no/analyzeTrafficData.xhtml. Visited 23.01.2020.

[61] Kemp & Young. Ship stability notes & examples., 2001. URL https://epdf.pub/ship-stability-notes-and-examples-third-edition-kemp-amp-young-series-kemp-amp-y.html. Third Edition.

Appendices

Appendix A

Project Planning

A.1 Pre-Project Report

FORPROSJEKT - RAPPORT

FOR BACHELOROPPGAVE

NTNU
Kunnskap for en bedre verden

TITTEL:
Autodocking v.h.a. Lidar

KANDIDATNUMMER(E):
Håkon Bjerkgaard Waldum, Ruben Ole Berg Natvik, Ruben Svedal Jørundland, Vebjørn Rimstad Wille

DATO:	EMNEKODE:	EMNE:	DOKUMENT TILGANG:
30/01-2020	**IE303612**	**Bacheloroppgave**	- Åpen

STUDIUM:	ANT SIDER/VEDLEGG:	BIBL. NR:
BACHELOR I INGENIØRFAG - AUTOMATISERINGSTEKNIKK	14 / 2	- Ikke i bruk -

OPPDRAGSGIVER(E)/VEILEDER(E):
Ottar L. Osen – NTNU Robin T. Bye – NTNU Øivind Kåre Kjerstad – Kongsberg Maritime

OPPGAVE/SAMMENDRAG:

Oppgaven er utdelt av Kongsberg Maritime ved kontaktperson Øivind Kåre Kjerstad. Oppgaven går ut på å lage et system for autonom docking av skip uten bruk av GPS. Det er tatt et valg om å bruke lidar til å kunne kartlegge og posisjonere skipet lokalt.

Det er gjennom rapporten satt opp prosedyrer for godkjenning av arbeid internt, avviksbehandling samt en struktur i prosjektgruppen for å kunne sikre god flyt og kontinuerlig oppdatering av status så det enkelt kan omprioriteres ressurser ved nødvendighet. Det er også foretatt en risikovurdering av prosjektet og arbeidet som inngår i dette, som dermed gjør det mulig å jobbe tryggere gjennom perioden.

Det er satt opp liste for hva slags dokumentasjon som skal produseres gjennom arbeidet, noe av det som inngår er rapport, eltegninger, utstyrsliste og bruksanvisning. Det er også satt opp en prosjektplan for hva som skal gjøres og til hvilken tid dette skal utføres.

Denne oppgaven er en eksamensbesvarelse utført av student(er) ved NTNU i Ålesund.

Postadresse
Høgskolen i Ålesund
N-6025 Ålesund
Norway

Besøksadresse
Larsgårdsvegen 2
Internett
www.hials.no

Telefon
70 16 12 00
Epostadresse
postmottak@hials.no

Telefax
70 16 13 00

Bankkonto
7694 05 00636
Foretaksregisteret
NO 971 572 140

INNHOLD

1 INNLEDNING

Oppgaven som vil bli utført gjennom dette prosjektet er å lage et system for autonom docking ved hjelp av en lidar. Oppdragsgiver i dette prosjektet er Kongsberg Maritime, med Øivind Kåre Kjerstad som kontaktperson.

Den grunnleggende problemstillingen er å finne et godt system for hvordan båten som skal docke autonomt skal kunne orientere seg for å klare å utføre operasjonen uten bruk av en GPS. Tanken er at dette skal gjøres med en lidar og hjelp av andre sensorer (gyro og akselerometer). Systemet vil gjøres uavhengig av resten av systemet på båten.

2 BEGREPER

BEGREP	BETYDNING
ROS	Robot Operating System
SLAM	Simultaneous Localization And Mapping
GUI	Graphical User Interface

3 PROSJEKTORGANISASJON

3.1 Prosjektgruppe

Studentnummer(e)
488575 – Håkon Bjerkgaard Waldum
488583 – Ruben Natvik
488584 – Ruben Svedal Jørundland
460617 – Vebjørn Rimstad Wille

Tabell: Studentnummer(e) for alle i gruppen som leverer oppgaven for bedømmelse i faget ID 302906

3.1.1 Oppgaver for prosjektgruppen - organisering

Prosjektleder: Håkon Bjerkgaard Waldum
Sekretær: Ruben Natvik
Øvrige: Ruben Svedal Jørundland
Vebjørn Rimstad Wille

3.1.2 Oppgaver for prosjektleder

Som leder av prosjektet vil ansvarsområder være å holde kontakt med veileder hos NTNU samt kontaktperson hos Kongsberg Maritime. Dette vil da inkludere korrespondanse på mail samt avtaling av møter og lignende.

Det vil også inkludere å tilse at utdelte arbeidsoppgaver blir utført som avtalt, samt oppfølge status av prosjekt under arbeidet. Leder har også overordnet ansvar for å ta endelig beslutning i tilfeller der gruppen er uenig om retning som skal tas for en eventuell problemstilling.

3.1.3 Oppgaver for sekretær

Hovedansvaret for sekretær er å loggføre fremdrift i prosjektet og føre referat for alle møter som blir holdt gjennom prosjektperioden. Dette inkluderer dermed å holde system på fremdrift i form av rapporter samt sende disse rapportene videre til prosjektleder for videre korrespondanse med oppdragsgiver og veileder.

3.1.4 Oppgaver for øvrige medlem(mer)

Oppgaver for øvrige medlemmer vil være å holde prosjektleder samt sekretær ansvarlig for de oppgavene de har blitt utdelt og se til at de gjør jobben som de har tatt. Arbeidsoppgaver øvrige medlemmer har utover dette er å loggføre sitt eget arbeid i form av dagbøker og generell dokumentasjon.

3.2 Styringsgruppe (veileder og kontaktperson oppdragsgiver)

- Ottar L. Osen
- Robin T. Bye
- Øivind Kjerstad

4 AVTALER

4.1 Avtale med oppdragsgiver

Det er lagd noen avtaler med oppdragsgiver under oppstartsmøtene av prosjektet. Det ble avtalt å holde møte hver 14. dag for å ha en liten oppdatering om status i prosjektet og tilbakemeldinger. Det vil også sendes en ukentlig rapport om status på uken som er gått, hva som skal gjøres i uken som kommer og hvordan det ligger an i forhold til planen som er lagt opp.

4.2 Arbeidssted og ressurser

Under prosjektarbeidet blir det flere arbeidsområder for å kunne utføre prosjektet som planlagt. Tunglaben vil bli brukt for å utføre arbeidet med å bygge det fysiske systemet med båten og det som trengs i forhold til å teste systemet. Det vil også bli utført noen tester i bassenget nede på verkstedene for å kunne teste båtens flyteevne.

Når det kommer til arbeid med programmering og rapportskriving vil det bli utført enten på rom L167 på Lab-bygget, eller på et av grupperommene som er tilgjengelige. Når det kommer til endelig testing av prosjektet vil dette bli utført ved sjøen, mest sannsynlig ved siden av Sunnmøre Museum.

Tilgang til ressurser gjennom prosjektet blir komponenter og lignende som NTNU har tilgjengelig, samt supplering gjennom bestilling av varer. Disse bestillingene må forholde seg til at det er satt et budsjett for prosjektet på mellom 3000kr og 5000kr.

Tilgang til personer for hjelp gjennom prosjektet blir jo primært veiledere og kontaktperson, men det vil også bli benyttet hjelp fra labansvarlig der det er relevant.

I dette prosjektarbeidet er det ingen informasjon som vil være unndratt offentligheten, og derfor vil det ikke bli lagt noe i ekstra datasikkerhet i forhold til det å holde deler av arbeidet hemmeligholdt.

Avtalt rapportering er hver uke i form av rapporter, samt møte med veiledere og kontaktperson hver andre uke.

4.3 Gruppenormer – samarbeidsregler – holdninger

For arbeidet innad i gruppen er det visse normer man vil sette fokus på for å holde et godt samhold gjennom hele prosjektarbeidet, samt klare å holde den kvaliteten man ønsker på sluttproduktet. Den viktigste normen som settes fokus på innad i gruppen er ansvarlighet. Innad i denne normen anses punktlighet, ansvarsbevissthet og god samarbeidsvillighet som viktige punkter å opprettholde. For å sikre prosjektets kvalitet er det viktig at egen ære ikke settes over å søke hjelp eller andre løsninger fra kolleger.

Når det kommer til å være utøver av profesjonen automasjonsingeniør anses å opprettholde en god standard for kvalitet og funksjonalitet i produktet som viktigst. Dette for å sikre at når et produkt er levert vil det ikke oppstå situasjoner der produktet svikter etter kort tid som resultat av dårlig fokus gjennom arbeidet. En annen viktig holdning man vil sette fokus på i utviklingen av nye produkt og løsninger er bærekraft. En enkel måte å bygge et bærekraftig produkt på er å sikre at det vil vare lengst mulig, og ikke måtte byttes ut etter kort tid med en nyere versjon, men heller kan vedlikeholdes og oppdateres. For å klare dette må dermed produktet være lagd på en slik måte at det enkelt kan utvides via å være modulær på en form eller måte.

5 PROSJEKTBESKRIVELSE

5.1 Problemstilling - målsetting - hensikt

Den grunnleggende problemstillingen er å kunne lage et system for autonom docking av båter uten hjelp av GPS. Dette systemet vil derfor bli bygd med egne sensorer, og ikke basere seg på sensorer som allerede er på båten. Dermed vil dette systemet ikke bli påvirket dersom noen av hovedsystemene til båten slutter å fungere.

Det skal derfor i dette prosjektet kommes fram til en god løsning for dette, ved hjelp av en lidar som vil brukes for lokal posisjonering. Målet vil være å ta i bruk teknologiene og prinsippene RoS (Robot Operating System) samt SLAM (Simultaneous Localization and Mapping) for å utføre arbeidet.

Hovedmålene i prosjektarbeidet er delt opp i:

- Dokumentasjon
- Prototypebygging
- Programmering
- Testing

Under disse hovedmålene vil det være mange delmål som må oppnås for å kunne regne hovedmålet som utført.

For dokumentasjon vil det være delmålene:

- Rapport
- El-tegninger
- Kode
- Bruksanvisning

Med disse delmålene vil man oppnå en komplett dokumentasjon for hele prosjektet som et produkt.

For prototypebyggingen er det delt opp i:

- Research og modellering
- Bygging

- Testing

Programmeringen er den største delen av prosjektet og har mange delmål:

- Datainnhenting fra sensor
- Databehandling fra sensor
- Kommunikasjonsprotokoll
- GUI
- Pathfinding
- Object Detection
- Reguleringssystem
- Sammensying av moduler

Dette er i all hovedsak de delmålene programmeringen vil bli delt inn i sånn som det blir ansett nå.

Det siste hovedmålet av prosjektet blir da testing, som vil bestå av:

- Testing av program
- Testing av prototype
- Testing av komplett løsning

En del av testingen vil ta plass underveis i prosjektet, mens testingen av den komplette løsningen vil ta plass som siste del av arbeidet med prosjektet før det blir gått over til å kun skrive rapport og dokumentasjon.

5.2 Krav til løsning eller prosjektresultat – spesifikasjon

Spesifikasjoner og krav som er satt opp for prosjektet er at det skal holdes innenfor budsjettrammene som er gitt, dvs. 3000-5000kr. Samt at dette skal være et såkalt «self-contained» system som ikke er avhengig av alle andre systemer som er på en båt. Det er også et krav om at dette er et autonomt system som lokaliserer seg ved hjelp av noe annet enn en GPS.

Leveranse av prosjektet skal inneholde en del ting;

- Generell dokumentasjon (Rapport, bruksanvisning, el-tegninger, kode, budsjett, eventuell statistikk/resultatdata innhentet gjennom arbeidet)
- Ferdig og fungerende prototype

Ved å levere dette vil prosjektet være regnet som ferdig og levert innenfor de spesifikasjoner som er satt for prosjektet.

5.3 Planlagt framgangsmåte(r) for utviklingsarbeidet – metode(r)

Planlagt fremgangsmåte for å utvikle dette produktet er å bruke en iterativ utviklingsprosess. Dette gjør at man kjapt kan komme i gang og begynne å teste mulige måter å utføre prosjektet på.

En kjent problematikk med denne typen framgangsmåte er at man kan støte på problemer lengre inn i prosjektarbeidet med at løsningen man har funnet på en problemstilling kan vise seg å ikke fungere godt nok i sammenheng med andre moduler man også skal ha i produktet. Dette kan skape problemer med å måtte gå tilbake til egentlige ferdigstilte moduler for å gjøre om på måten de kommuniserer på eller fungerer på en generell basis.

Det som vil gjøres for å unngå en slik problemstilling er å sette opp en krav-spec for å spesifisere hvordan den skal fungere i sammenheng med andre moduler og på hvilken måte disse modulene skal kommunisere. På denne måten kan det fastsettes hvordan alle moduler skal jobbe i sammenheng med de andre på et teoretisk nivå.

Det vil bli tatt i bruk noe som er kjent som «Fast Prototyping» for å bygge båten som skal brukes i arbeidet. Konseptet med Fast Prototyping er å bruke 3D-modellering for å fort kunne se hva som kan fungere, før man så benytter 3D-printing og andre ressurser som er tilgjengelige for å få ut en prototype fort for å kunne teste om dette vil fungere.

Dette er noe som ønskes å ta i bruk for å få ferdig prototypen kjapt for å kunne teste at det fungerer og dermed kunne hoppe videre til å bygge styringssystemet rundt denne båten som skal produseres.

5.4 Informasjonsinnsamling – utført og planlagt

Under arbeidet så langt med forprosjektet er det funnet og blitt tipset om forskjellige måter å implementere et lignende system som er ønsket i dette prosjektet. Det vil dermed prøves å tas i bruk systemene ROS og SLAM for å utføre arbeidet.

Det er allerede gjort en del research på hvordan disse systemene er implementert på andre robotsystemer. Dette er gjort via å lese vitenskapelige artikler fra utviklere innen ROS og SLAM, samt tidligere bacheloroppgaver og masteroppgaver som er skrevet for NTNU.

Det vil også innhentes mer informasjon om hvilke teknikker som er brukt og kan brukes ved hjelp av å søke rundt i vitenskapelige tidsskrifter for å lete etter tidligere forskning på temaet med autonom docking.

Det er allerede også påbegynt research angående robotikk, men det er fortsatt en del som gjenstår på dette punktet. Dette anses som noe som er viktig å informere seg godt på, da ROS baserer seg tungt på robotikk.

5.5 Vurdering – analyse av risiko

Det er utført en generell risikovurdering for prosjektet som helhet, og denne risikovurderingen er vedlagt som vedlegg i denne rapporten. Se vedlegg nr. 2

5.6 Hovedaktiviteter i videre arbeid

Nr	Hovedaktivitet	Ansvar	Kostnad	Tid/omfang
A1	Dokumentasjon	**Alle**	0	Semester
A2	Prototype-bygging	RSJ	3000kr	2 uker
A21	R&D	RSJ	0	3 dager
A22	Bygging	RSJ	3000kr	5 dager
A23	Testing	RSJ	0	2 dager
A3	Research	RN/VRW	0	1 uke
A31	RoS	RN/VRW	0	3 dager
A32	Slam	RN/VRW	0	3 dager
A33	Eksisterende teknikker	RN/VRW/HBW	0	2 dager
A331	i.f.t Docking	RN/VRW	0	2 dager
A332	i.f.t autonomt	HBW	0	2 dager
A4	Programmering	Alle	0	10 uker

A41	Datainnhenting	RN	0	2 uker
A42	Databehandling	VRW	0	2 uker
A43	Kommunikasjon	RSJ/HBW	0	1 uke
A44	GUI	VRW	0	3 uker
A45	Pathfinding (?)	HBW	0	3 uker
A46	Object detection	HBW/RSJ	0	2 uker
A47	Reguleringssystem	RN/VRW	0	3 uker
A48	Sammensying	HBW	0	1 uke
A5	Testing	Alle	0	3 uker
A51	Program	Alle	0	1 uke
A52	Testing av prototype	Alle	0	1 uke
A53	Komplett testing	Alle	0	1 uke

Dette var en innledende versjon av prosjektplanleggeren, den endelige versjonen er lagt ved som et vedlegg.

5.7 Framdriftsplan – styring av prosjektet

5.7.1 Hovedplan

Hovedtrekk i gjennomføringen vil være at dokumentasjon skal produseres jevnlig mens det blir jobbet med prosjektet for å slippe at det kommer en ekstrem mengde papirarbeid som må gjøres når alt annet er ferdig mot slutten av semesteret. Det vil så fokuseres på å bygge en prototype mens en del av prosjektgruppen bruker tid på å gjøre research på forskjellige utføringsmetoder samt prinsipper som vil være nødvendig å ha en kunnskap om gjennom prosjektet.

Når disse tingene er utført vil det gå over til å bli et hovedfokus på programmeringen og de forskjellige delmålene som må nås i dette hovedmålet. Når alt innenfor programmering er utført og testet vil det gå over i en testings-fase der produktet som en helhet vil bli testkjørt og justert inn slik at det fungerer som ønsket.

Når det kommer til prototypen er dette et hovedmål som er delt opp i 3 deler; modellering og research, bygging og testing. Forventet start på dette er uke 3, og den forventes ferdig i løpet av uke 6. Dette er den første milepælen i prosjektet.

Samtidig som det blir jobbet med prototypen vil en del av gruppen ta del i research. Researchen inneholder informasjonsinnhenting om ROS, SLAM, samt informasjon om hvordan docking gjøres manuelt samt hvordan det utføres autonomt. Hele dette hovedmålet er tiltenkt å brukes 3 uker på. Det er forventet at dette skal starte i uke 3 og blir ferdig i løpet av uke 5. Her må det tas forbehold om at mye av uke 3 vil gå til Industri 4.0.

Når det kommer til programmeringen så må det gjøres mange ting av forskjellige personer som jobber synkront. Forventet start er uke 5, og dermed forventes hele programmeringsfasen å være ferdig i løpet av uke 15. Det er lagt på ekstra uker her da man av erfaring vet at det kan oppstå problematikk med de forskjellige modulene og de kan ta lengre tid enn man har planlagt. Det må også legges til grunn at i et par måneder framover vil det være Industri 4.0 hver andre uke, som vil ta en del tid vekk fra der man ellers ville jobbet med prosjektet.

Datainnhenting/databehandling henger veldig sammen og vil være avhengige av hverandre. Disse vil derfor jobbes med litt om hverandre i prosessen. Disse forventes å starte på i uke 5 og være ferdig i løpet av uke 7.

Kommunikasjon henger veldig sammen med datainnhentingen og vil derfor jobbes med rett etter datainnhenting er ferdig. Denne modulen forventes å startes på i uke 7 og ferdigstilles i uke 8.

En veldig basic GUI forventes å lages tidlig i prosjektet for å kunne enkelt framstille den informasjonen som innhentes fra sensorer, og vil dermed påbegynnes tidlig. En komplett og god løsning på GUI'en vil derfor ferdigstilles senere ut i prosjektløpet. Denne første iterasjonen av GUI'en forventes dermed å startes på i slutten av uke 5 og ferdigstilles i løpet av uke 7. Resterende del av GUI blir dermed tatt når en større del av programmet er ferdig.

Pathfinding og Object Detection vil komme naturlig etter punktsky fra lidar er innhentet, men research og en grunnstruktur kan bygges opp ved å undersøke hva som fungerer godt med resultatet som innhentes fra lidar. Pathfinding forventes å starte i uke 8 og forventes ferdig i løpet av uke 10. Object detection kan påbegynnes i slutten av uke 9 etter pathfinding er kommet i gang og forventes å ferdigstilles i uke 10. Dersom pathfinding viser seg å kunne ta seg av obstacle detection, vil det ikke lages et separat system for Object Detection.

Reguleringssystemet kan påbegynnes med en gang datainnhenting er ferdigstilt, men er avhengig av resultatet fra Pathfinding. Dette forventes å påbegynnes i uke 8 og forventes å ferdigstilles i løpet av uke 9.

Sammensying vil komme naturlig gjennom hele programmeringsfasen, og skje når hver enkelt modul er klar til å merges inn i hovedprogrammet.

Per nå er det vanskelig å si hvilken av milepælene som blir den store utfordringen og vil ta lengre tid enn forventet, så dermed er dette en foreløpig plan med forbehold om ingen problemer.

Beslutninger underveis i forhold til de modulene og milepælene som er satt opp blir i hovedsak tatt av de som er satt som ansvarlige for disse oppgavene. Disse kan dermed delegere og bestemme innenfor disse. Ved eventuell problematikk eller avvik vil beslutninger tas i henhold til avviksprotokollen som er nevnt i kapittel 8.

5.7.2 Styringshjelpemidler

Hjelpemidler som vil bli tatt i bruk for å styre og veilede prosjektet vil være:

- Timeliste
- Prosjektplanlegger
- Individuelle dagbøker

Timelistene vil bli brukt for å kunne holde hver enkelt person i prosjektet ansvarlig for å dra sin del av lasset når det kommer til prosjektarbeidet. Her kan det også enkelt ses i korte trekk hva hver enkelt har gjort på de forskjellige dagene og hvor mange timer som er lagt inn i arbeidet. Timelisten vil inneholde:

- Dato
- Antall Timer
- Kort beskrivelse av hva som er gjort

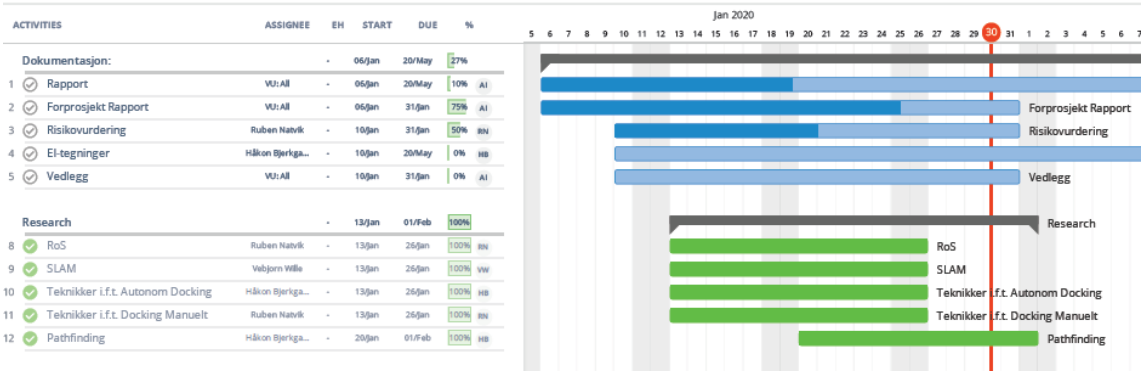
	A	B	C
1	Navn:	Håkon B. Waldum	
2	Timer Totalt:	12	
3			
4	Timeliste		
5			
6	***Dato***	***Antall Timer***	***Beskrivelse***
7	06.01.2020	4	Forarbeid. Forrapport / Dokumentlaging / System
8	09.01.2020	2	Oppdatering av Vebjørn på hva som er gjort, generell status
9	10.01.2020	4	Møte med Kongsberg, forrapport
10	12.01.2020	2	Reinskriving av forrapport

Utklipp av timeliste

Prosjektplanlegger er et dokument der mål og delmål blir oppført med en forventet arbeidstid, samt en ansvarlig person som skal se dette gjort. Dette brukes for å kunne enkelt se hva hver enkelt skal gjøre fra uke til uke samt kunne se framdriften på hvert enkelt delmål. Prosjektplanlegger vil derfor inneholde:

- Mål/delmål
- Ansvarlig person for gitt mål/delmål
- Progresjon i delmål
- Forventet start og slutt på arbeidet med mål/delmål

Dette gjør det enkelt å se hva man skal gjøre og til hvilken tid det er forventet at arbeidet skal være ferdig.



Utkast av prosjektplanleggeren

Individuelle dagbøker blir brukt som en ressurs for videre dokumentasjon når det kommer til rapporten for prosjektet. Ved å skrive individuelle dagbøker kan hver enkelt person føre inn hva som er gjort på de forskjellige dagene på en mer detaljert måte enn i timelistene, samt lime inn referanser som er brukt under arbeidet for å få fullført forskjellige oppgaver. En individuell dagbok skal dermed inneholde:

- Dato
- Beskrivelse av problemstilling som er jobbet med
- Eventuelle løsninger som er funnet for problemstillingen
- Eventuelle referanser som er brukt for å løse gitt problemstilling

Fredag 10.01.2010

Oppstartsmøte bachelor, arbeid med forprosjekt-rapport

Sett inn referat her

Referanser

Ingen

Oppsett av hvordan et dagbokinnlegg vil se ut

5.7.3 Utviklingshjelpemidler

- PyCharm (Python)
- Visual Studio Code (JavaScript, Html, CSS)
- Sublime Text
- RoS
- Slam
- Fusion 360 (3D-Modellering)
- Matlab (regulering og modellering av systemet)

5.7.4 Intern kontroll – evaluering

Intern kontroll og evaluering av prosjektet på løpende basis vil bli gjennomført ved å holde ukentlige fredagsmøter med en statusrapport. På dette fredagsmøtet skal visse punkter gås gjennom:

- Status på uke som er gått sammenlignet med plan fra uken før
- Status på hver enkelt persons arbeidsoppgaver
- Eventuelle problemstillinger som er oppstått
- Videre plan for uken som kommer
- Eventuelle ressursmangler
- Eventuell omprioritering av ressursbruk

Deretter skal det skrives en oppsummering i form av en statusrapport samt et referat av møtet.

I forhold til kriterier/kjennetegn på at mål/delmål er nådd er at ønsket funksjon av dette delmålet er tilstede, og at dens funksjonalitet er stabil og uten videre problematikk. Det må også legges til grunn at dette målet/delmålet fungerer tilstrekkelig med andre moduler som eventuelt skal jobbe i plenum med dette målet/delmålet.

5.8 Beslutninger – beslutningsprosess

Generelle beslutninger om avgrensning og presisering av oppgaven under arbeidet med forprosjektet er tatt i plenum med et grunnlag i kunnskap og tid nødvendig for å tilegne seg den nødvendige informasjonen for å kunne utføre prosjektarbeidet til et tilfredsstillende nivå.

Under viktige beslutninger vil en risikoanalyse av hver av mulighetene utføres, gruppen informeres om disse risikoanalysene og deretter vil en avstemning utføres. Ved en eventuell uavgjort vil dermed prosjektleder få en dobbeltstemme for å avgjøre endelig valgt løsning, dette ses som en nødvendighet da gruppen består av 4 medlemmer og en eventuell uavgjort ses som en reell situasjon som kan oppstå.

6 DOKUMENTASJON

6.1 Rapporter og tekniske dokumenter

Når det kommer til dokumentasjon som skal utarbeides i løpet av prosjektet er det en del som må gjøres. Dokumentasjon som skal produseres er:

- Hovedrapport
- Timeliste
- Ukentlige statusrapporter
- El-Tegninger
- Bruksanvisning

Dette er det som anses som dokumentasjon som vil være viktig å få med i løpet av prosjektet og dermed når prosjektet skal overleveres.

Det er en del rutiner som vil inngå i løpet av dette prosjektet, både individuelt samt generelt på gruppe-basis. Disse rutinene er:

- Fylle ut timeliste hver dag
- Skrive i individuell dagbok
- Oppdatere status i prosjektplanlegger
- Ukentlige fredagsmøter med statusrapport
- Oppmøte 0815 på hverdager om ikke annet er avtalt

Med disse rutinene på plass legges en god grunnmur for et godt prosjektarbeid med et sluttprodukt som vil leve opp til forventningene satt under forprosjektet.

Det vil være nødvendig å ha et system på plass for godkjenning av merge-requests når det kommer til kode, samt godkjenning av moduler som skal inn i rapporten. Dermed er det kommet fram til et system for å gjøre dette enkelt:

- For merge-requests er det nødvendig at minst én annen person av prosjektgruppen ser over og godkjenner denne merge-requesten før koden merges inn i hovedkoden. Dette loggføres automatisk i GitLab.
- For rapport-relaterte moduler skal modulen godkjennes av minst 2 andre personer av prosjektgruppen før denne modulen legges inn og godkjennes i rapporten. Dette vil loggføres via bruk av kommentar-funksjonen i Word.

7 PLANLAGTE MØTER OG RAPPORTER

7.1 Møter

7.1.1 Møter med styringsgruppen

Det er per nå avtalt å møtes med styringsgruppen hver 14. dag for å ha en liten statusoppdatering på prosjekt og tilbakemeldinger om arbeidet og prosjektet generelt.

7.1.2 Prosjektmøter

Det vil føres ukentlige prosjektmøter hver fredag klokken 0815 til 0915 i ukene. Hensikten med disse møtene er å sjekke status på hver enkelt person samt planlegging av videre framdrift og refleksjon på endt ukes resultat. Her vil det skrives en statusrapport som vil leveres til prosjektleder for eventuell videresending til veileder hos NTNU eller kontaktperson i Kongsberg Maritime.

7.2 Periodiske rapporter

7.2.1 Framdriftsrapporter (inkl. milepæl)

Planlagte rapportformer for dette prosjektet er ukesrapporter fra statusmøter hver mandag. Dermed vil det være ukentlige rapporter tilgjengelige for veileder samt kontaktpersoner.

8 PLANLAGT AVVIKSBEHANDLING

Ved eventuelle avvik i prosjektet må det inkalles til et hastemøte for å avgjøre hva som skal skje videre. En avgjørelse angående problemstillingen vil avgjøres i plenum avhengig av dens alvorlighetsgrad. Her kan det eventuelt omdelegeres ressurser for å hjelpe til med å løse problemstillingen.

Ved en eventuell endring i prosjektet vil det opprettes en ny revisjon av prosjektplanleggeren (dvs. ny side i Excel) og markere med dato for når denne nye revisjonen ble tatt i bruk.

Ruben S. Jørundland vil stå som hovedansvarlig for behandling av avvik som kan oppstå under prosjektet.

9 UTSTYRSBEHOV/FORUTSETNINGER FOR GJENNOMFØRING

- Thrustere med motorkontrollere
- Lidar
- Bassenget nede ved tunglaben
- Raspberry Pi 4
- Batteri
- Gyroskop / Akselerometer
- Rørdeler til konstruksjonen
- Spenningsregulatorer
- Arduino Nano

10 REFERANSER

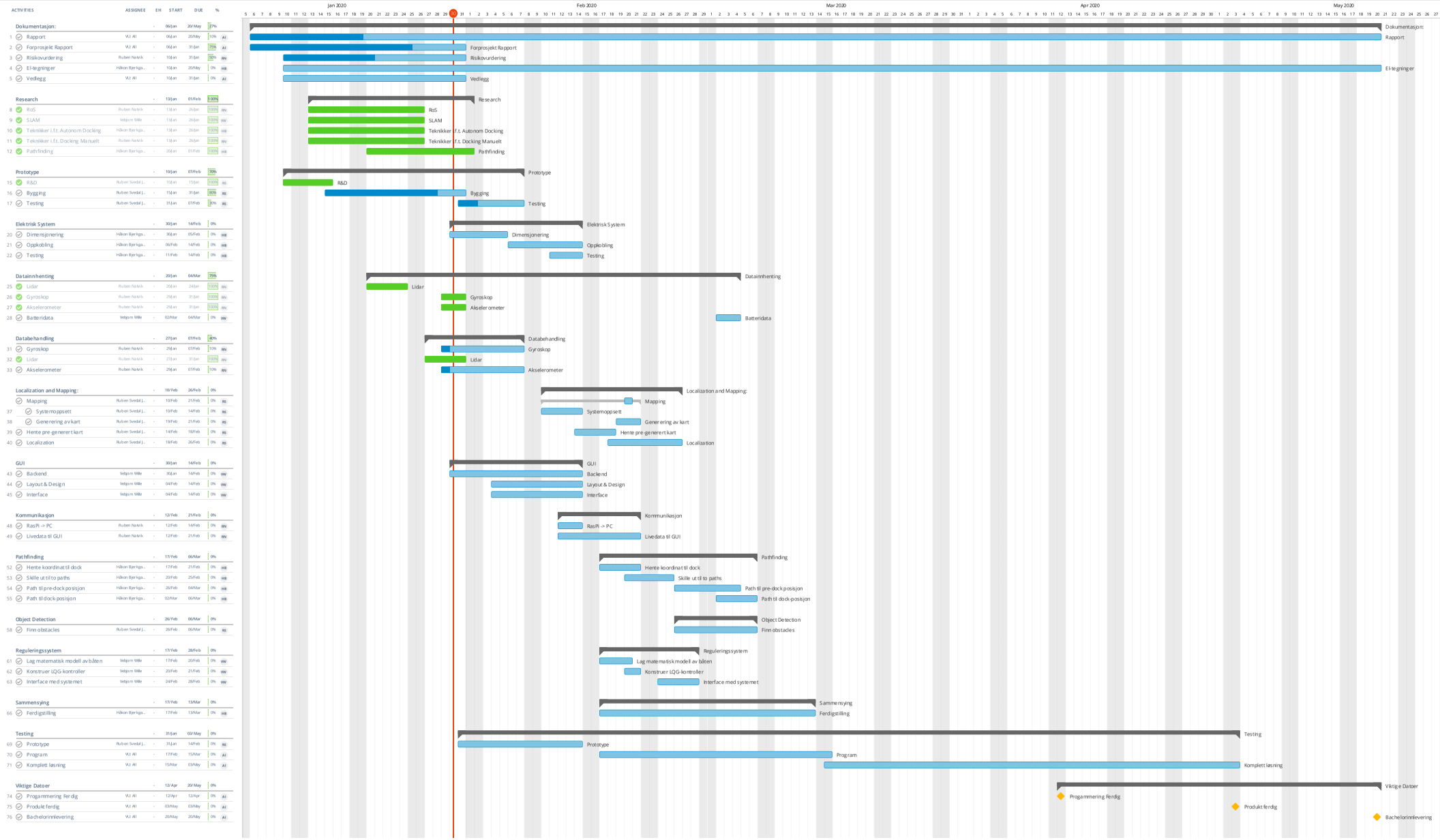
VEDLEGG

Vedlegg 1	Prosjektplanlegger
Vedlegg 2	Risikovurdering

A.2 Original Project Plan

Bachelor

Read-only view, generated on 30 jan 2020

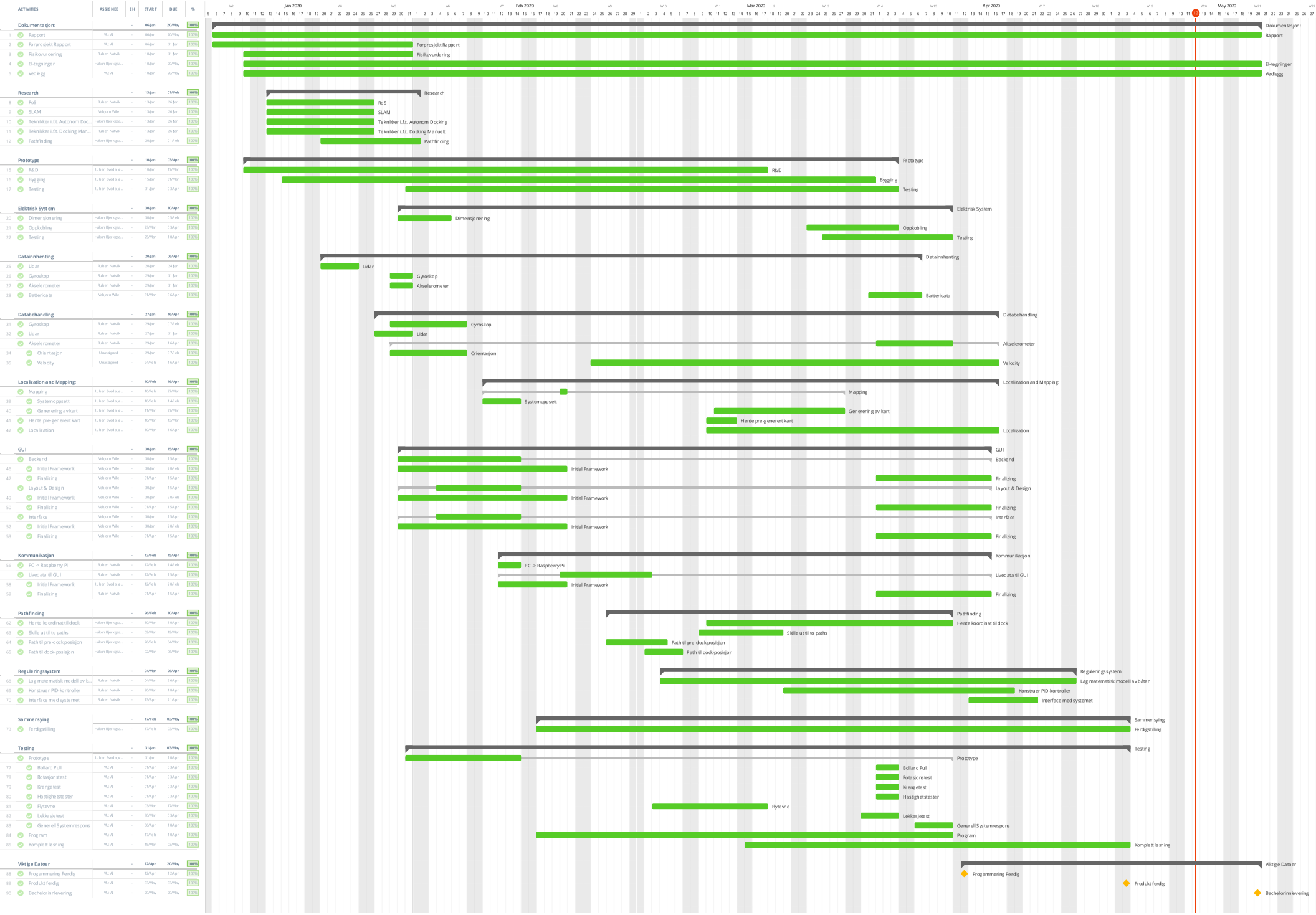


	ACTIVITIES	ASSIGNEE	EH	START	DUE	%	
	Dokumentasjon:		-	06/Jan	20/May	27%	
1	Rapport	VU, AI	-	06/Jan	20/May	10%	AI
2	Forprosjekt Rapport	VU, AI	-	06/Jan	31/Jan	75%	AI
3	Risikovurdering	Ruben Nævik	-	10/Jan	31/Jan	50%	RN
4	El-tegninger	Håkon Bjerkga...	-	10/Jan	20/May	0%	HB
5	Vedlegg	VU, AI	-	10/Jan	31/Jan	0%	AI
	Research		-	13/Jan	01/Feb	100%	
8	RoS	Ruben Nævik	-	13/Jan	26/Jan	100%	RN
9	SLAM	Vebjørn Wille	-	13/Jan	26/Jan	100%	VW
10	Teknikker i.f.t. Autonom Docking	Håkon Bjerkga...	-	13/Jan	26/Jan	100%	HB
11	Teknikker i.f.t. Docking Manuelt	Ruben Nævik	-	13/Jan	26/Jan	100%	RN
12	Pathfinding	Håkon Bjerkga...	-	20/Jan	01/Feb	100%	HB
	Prototype		-	10/Jan	07/Feb	70%	
15	R&D	Ruben Svedal J...	-	10/Jan	15/Jan	100%	RS
16	Bygging	Ruben Svedal J...	-	15/Jan	31/Jan	80%	RS
17	Testing	Ruben Svedal J...	-	31/Jan	07/Feb	30%	RS
	Elektrisk System		-	30/Jan	14/Feb	0%	
20	Dimensjonering	Håkon Bjerkga...	-	30/Jan	05/Feb	0%	HB
21	Oppkobling	Håkon Bjerkga...	-	06/Feb	14/Feb	0%	HB
22	Testing	Håkon Bjerkga...	-	11/Feb	14/Feb	0%	HB
	Datainnhenting		-	20/Jan	04/Mar	75%	
25	Lidar	Ruben Nævik	-	20/Jan	24/Jan	100%	RN
26	Gyroskop	Ruben Nævik	-	29/Jan	31/Jan	100%	RN
27	Akselerometer	Ruben Nævik	-	29/Jan	31/Jan	100%	RN
28	Batteridata	Vebjørn Wille	-	02/Mar	04/Mar	0%	VW
	Databehandling		-	27/Jan	07/Feb	40%	
31	Gyroskop	Ruben Nævik	-	29/Jan	07/Feb	10%	RN
32	Lidar	Ruben Nævik	-	27/Jan	31/Jan	100%	RN
33	Akselerometer	Ruben Nævik	-	29/Jan	07/Feb	10%	RN
	Localization and Mapping:		-	10/Feb	26/Feb	0%	
	Mapping	Ruben Svedal J...	-	10/Feb	21/Feb	0%	RS
37	Systemoppsett	Ruben Svedal J...	-	10/Feb	14/Feb	0%	RS
38	Generering av kart	Ruben Svedal J...	-	19/Feb	21/Feb	0%	RS
39	Hente pre-generert kart	Ruben Svedal J...	-	14/Feb	18/Feb	0%	RS
40	Localization	Ruben Svedal J...	-	18/Feb	26/Feb	0%	RS
	GUI		-	30/Jan	14/Feb	0%	
43	Backend	Vebjørn Wille	-	30/Jan	14/Feb	0%	VW
44	Layout & Design	Vebjørn Wille	-	04/Feb	14/Feb	0%	VW
45	Interface	Vebjørn Wille	-	04/Feb	14/Feb	0%	VW
	Kommunikasjon		-	12/Feb	21/Feb	0%	
48	RasPi -> PC	Ruben Nævik	-	12/Feb	14/Feb	0%	RN
49	Livedata til GUI	Ruben Nævik	-	12/Feb	21/Feb	0%	RN
	Pathfinding		-	17/Feb	06/Mar	0%	
52	Hente koordinat til dock	Håkon Bjerkga...	-	17/Feb	21/Feb	0%	HB
53	Skille ut til to paths	Håkon Bjerkga...	-	20/Feb	25/Feb	0%	HB
54	Path til pre-dock posisjon	Håkon Bjerkga...	-	26/Feb	04/Mar	0%	HB
55	Path til dock-posisjon	Håkon Bjerkga...	-	02/Mar	06/Mar	0%	HB
	Object Detection		-	26/Feb	06/Mar	0%	
58	Finn obstacles	Ruben Svedal J...	-	26/Feb	06/Mar	0%	RS
	Reguleringssystem		-	17/Feb	28/Feb	0%	
61	Lag matematisk modell av båten	Vebjørn Wille	-	17/Feb	20/Feb	0%	VW
62	Konstruer LQG-kontroller	Vebjørn Wille	-	20/Feb	21/Feb	0%	VW
63	Interface med systemet	Vebjørn Wille	-	24/Feb	28/Feb	0%	VW
	Sammensying		-	17/Feb	13/Mar	0%	
66	Ferdigstilling	Håkon Bjerkga...	-	17/Feb	13/Mar	0%	HB
	Testing		-	31/Jan	03/May	0%	
69	Prototype	Ruben Svedal J...	-	31/Jan	14/Feb	0%	RS
70	Program	VU, AI	-	17/Feb	15/Mar	0%	AI
71	Komplett løsning	VU, AI	-	15/Mar	03/May	0%	AI
	Viktige Datoer		-	12/Apr	20/May	0%	
74	Progammering Ferdig	VU, AI	-	12/Apr	12/Apr	0%	AI
75	Produkt ferdig	VU, AI	-	03/May	03/May	0%	AI
76	Bachelorinnlevering	VU, AI	-	20/May	20/May	0%	AI

A.3 Final Project Plan

Bachelor

Read-only view, generated on 12 May 2020



A.4 Risk Assessment

Konsekvens: Sannsynlighet:	1. Ubetydelig	2. Mindre alvorlig/ En viss fare	3. Betydelig/ Kritisk	4. Alvorlig/farlig	5. Svært alvorlig/ katastrofalt
5. Svært sannsynlig	5	10	15	20	25
4. Meget Sannsynlig	4	8	12	16	20
3. Sannsynlig	3	6	9	12	15
2. Mindre sannsynlig	2	4	6	5	10
1. Lite sannsynlig	1	2	3	4	5

Matrise hentet fra: https://www.i (12.01.2020)

Hendelse / Situasjon	Aktuelt?	Sannsynlig	Konsekvens	Risiko	Kommentar / tiltak
Vann nært / på elektronikk i fartøyet	Ja	5. Svært sannsynlig	3. Kritisk Ødelagte komponenter / Miste kontroll på styring	15	Det må tas særlige forhåndsregler for å hindre kritiske feil ved å sørge for at elektroniske komponenter er skjermet mot vann.
Feil ved fremdrift / manøvrering	Ja	3. Sannsynlig	3. Kritisk Tap av prototype, skade på nærliggende gjenstander	9	For å hindre tap av prototypen og skade på nærliggende gjenstander ved eventuell feil på manøvrering og motor, må der være mulighet for å stoppe motorene eksternt samt tjore prototypen fast til land slik at den kan hales inn igjen ved motorstans
Personell og utstyr i nærhet av vann / sjø	Ja	3. Sannsynlig	2. Mindre alvorlig Personell / utstyr faller ut i vannet	6	For å begrense sannsyligheten for at personell / utstyr faller i vann må det tas forhåndsregler som f.eks minimum avstand fra landbasert utstyr til vannkant. Dersom personell skulle falle i vann, må stedet personen jobber fra ligge i nærheten av steder personen kan komme seg tilbake på land raskt. Pledd / tepper må medbringes ved arbeid ute ved havet.
Arbeid med elektrisk utstyr (person)	Ja	2. Mindre Sannsynlig	1. Ubetydelig For lav spenning til å ha noen effekt	2	Sannsynligheten for å få støt under arbeid med elektrisk utstyr er absolutt tilstede, men siden det jobbes med utstyr som har spenningsområde under 20V, anses det ikke som noen form for risiko for personskade som resultat av kontakt med ledende deler.
Arbeid med elektrisk utstyr (utstyret)	Ja	3. Sannsynlig	3. Betydelig Komponenter kan slutte å fungere	9	Under arbeidet med det elektriske systemet anses det som sannsynlig at en kortslutning kan forekomme. Dette kan få kritiske følger da komponentene kan bli ødelagt av dette. For å unngå dette vil det legges opp nøye planer for hva som skal gjøres, samt koble opp uten spenningsledende deler. Delene vil også adskilles så godt som det kan gjøres.
Arbeid med epoxy	Ja	4. Meget Sannsynlig	3. Betydelig Kan få epoxy på hud, allergisk reaksjon / irritasjon	12	Det må tas forhåndsregler for å hindre at epoxy kommer i kontakt med hudflater. Det vil derfor pålegges at arbeid med epoxy skal utføres med hansker og generell tildekking av hud.

A.5 Extraordinary Risk Assessment

Konsekvens: Sannsynlighet:	1. Ubetydelig	2. Mindre alvorlig/ En viss fare	3. Betydelig/ Kritisk	4. Alvorlig/farlig	5. Svært alvorlig/ katastrofalt
5. Svært sannsynlig	5	10	15	20	25
4. Meget Sannsynlig	4	8	12	16	20
3. Sannsynlig	3	6	9	12	15
2. Mindre sannsynlig	2	4	6	5	10
1. Lite sannsynlig	1	2	3	4	5

Matrise hentet fra: https://www. (12.01.2020)

Hendelse / Situasjon	Aktuelt?	Sannsynlig	Konsekvens	Risiko	Kommentar / tiltak
Gruppemedlem blir smittet	Ja	4. Meget Sannsynlig	4. Alvorlig	16	Risikoen for at et gruppemedlem blir smittet anses som meget sannsynlig, og kan få store konsekvenser om personen smitter resten av gruppen. For å hindre dette prøver gruppemedlemmer å holde seg mest mulig unna store samlinger
Hele gruppen blir smittet	Ja	2. Mindre Sannsynlig	5. Katastrofalt	10	Sjansen for at hele gruppen skal bli smittet anses som lav, men om dette skulle skje vil det bli katastrofalt for prosjektarbeidet. For å unngå dette vil gruppen prøve å samles minst mulig.
Arbeid med deler av / hele gruppen samlet	Ja	4. Meget Sannsynlig	2. Mindre Alvorlig	8	Gruppen må på deler av prosjektet jobbe samlet for å få gjort unna arbeidet som må gjøres. Ved arbeid samlet vil forhåndsregler som ingen deling av mat/drikke opprettholdes og gode håndvaskrutiner.

Appendix B

Bill of Materials (BOM)

The prototype is also constructed using 3D-printed parts. These are shown in appendix E.

B.1 BOM

Table B.1: Bill of Materials

Component	Description	Amount	Price each	Total Price
RPLidar A3	Lidar for the prototype	1	6110,-	6110,-
Raspberry Pi 4	Main microcontroller for the prototype	1	729,-	729,-
Arduino Nano	Microcontroller used for communicating with IMU	1	223,-	223,-
IMU 10DOF	10DOF IMU used for orientation	1	-	-
Roboclaw 2x30A	Motor controller	2	1274,33kr	2548,66kr
Haswing 20	Motor	4	749,-	2996,-
USB 2.0 A - Micro,~0.5m	Used to connect roboclaw to power	1	46,90kr	46,90kr
Ethernet Cable, 1m	Used for connecting raspberry pi to router	1	39,90kr	39,90kr

Ethernet Cable, 5m	Used as a cable for signals	1	84,90kr	84,90kr
Internet Router	Used for communication between prototype and PC	1	-	-
Main Fuse Holder	Holder for main fuse for whole electrical system	1	46,90kr	46,90kr
Main Fuse	Main fuse for whole electrical system	1	84,90kr	84,90kr
Circuit Fuse Holder	Holder for circuit fuses	1	64,90kr	64,90kr
Circuit Fuses Set (3A, 5A)	Fuses for the circuits to components	1	79,90kr	79,90kr
Waste Tube 110x3000mm	Tubing for prototype	2	249,-	498,-
Waste Tube Plug 110mm	Plugs used on bottom of prototype	6	69,13kr	414,78kr
Plexi Glass Plate 1x0.7m (Top)	Plexi glass used for top plate of prototype	1	-	-
Plexi Glass Plate 1x0.7m (Bot)	Plexi glass used for bottom plate of prototype	1	-	-
20x20x2000mm Alum. Profile	Alu profiling used to support plates	4	100,62kr	402,48kr
T250 IP66 Box	Electrical box used for battery and components	2	209-,	418,-
T350 IP66 Box	Electrical box used for components	1	549,-	549,-
Electric Box IP67~140x230x92	Electrical box used for roboclaws	1	64,90kr	64,90kr
12V 20Ah Battery	Battery for prototype	1	779,-	779,-
Buck Converter (5V)	Used to power RaspPi and Arduino	1	-	-
Buck/Boost Converter (12V)	Used to power Router	1	-	-

Main Power Switch	Main power switch for electrical system	1	199,-	199,-
Neutral Rail	Used to distribute power for electrical system	1	54,90kr	54,90kr
Total Price				16435,02 kr

B.2 URL for BOM

Table B.2: Bill of Materials (URL)

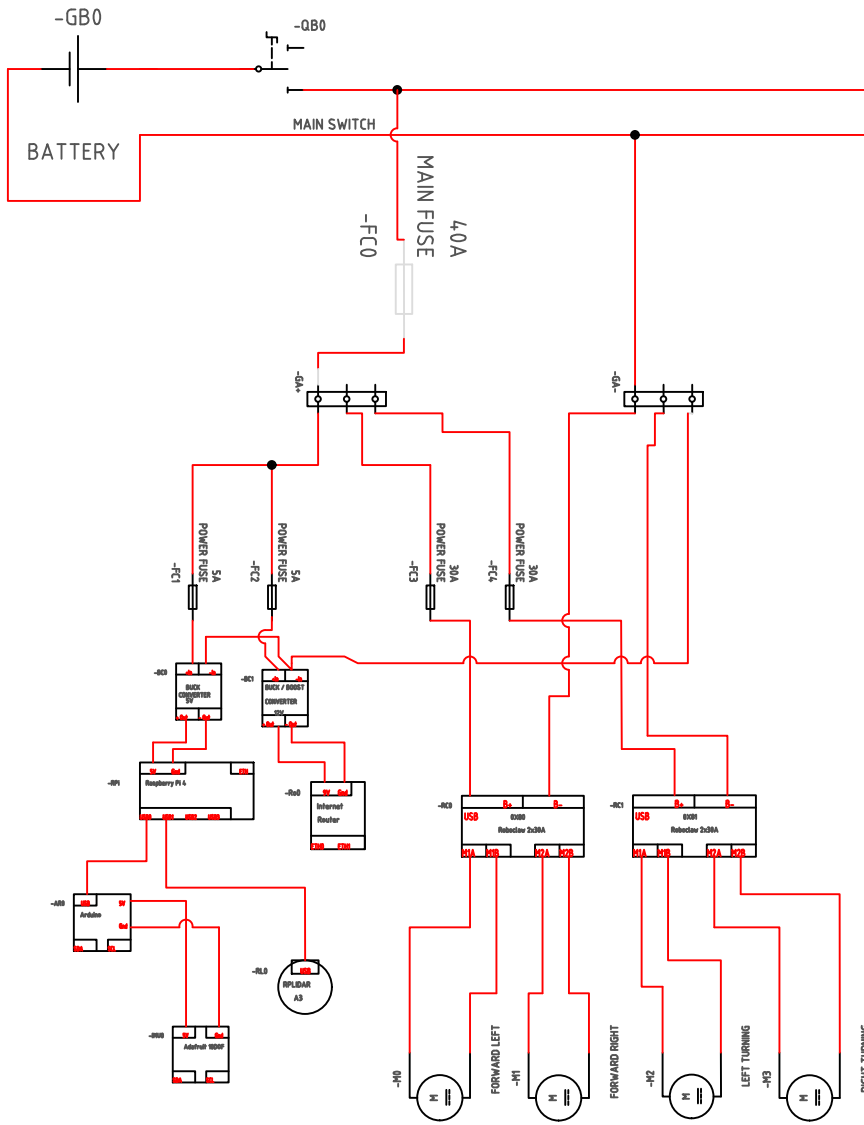
Component~	URL
RPLidar A3	https://www.slamtec.com/en/Lidar/A3
Raspberry Pi 4~	https://www.raspberrypi.org/products/raspberry-pi-4-model-b/
Arduino Nano	https://store.arduino.cc/arduino-nano
IMU 10DOF	https://www.adafruit.com/product/1604
Roboclaw 2x30A	https://www.basicmicro.com/RoboClaw-2x30A-Motor-Controller__.html
Haswing 20	https://www.westsystem.no/p/17762/elektrisk-baatmotor-haswing-20
USB 2.0, A - Micro, 0.5m	https://www.biltema.no/kontor-teknikk/datatilbehor/datakabler/usb-micro/usb-20-a—micro-2000033376
Ethernet Cable, 1m	https://www.biltema.no/kontor-teknikk/datatilbehor/nettverkskabler/cat-5-kabler/nettverkskabel-cat-5e-utp-2000033447
Ethernet Cable, 5m	https://www.biltema.no/kontor-teknikk/datatilbehor/nettverkskabler/cat-5-kabler/nettverkskabel-cat-5e-utp-2000033447
Internet Router	https://eu.dlink.com/uk/en/products/dir-809-wireless-ac750-dual-band-router
Main Fuse Holder	https://www.biltema.no/bil—mc/elektrisk-anlegg/sikringer/sikringsholder-2000032927
Main Fuse	https://www.biltema.no/bil—mc/elektrisk-anlegg/sikringer/sikringer-agu-6-stk-2000032928
Circuit Fuse Holder	https://www.biltema.no/bil—mc/elektrisk-anlegg/sikringer/sikringsholder-2000031352
Circuit Fuses (3A, 5A, 7,5A)	https://www.biltema.no/bil—mc/elektrisk-anlegg/flatstiftsikringer/flatstiftsikringer-120-stk-2000016459

Waste Tube 110x1000mm	https://coop.no/sortiment/obs-bygg/bad-og-kjokken-fbbffc80/ror-og-avlopsdeler/uponor-ror-110x1000mm?variantCode=450077
Waste Tube Plug 110mm	https://coop.no/sortiment/obs-bygg/bad-og-kjokken-fbbffc80/ror-og-avlopsdeler/uponor-rorpropp-110mm?variantCode=450108
Plexi Glass Plate 1x0.7m (Top)	
Plexi Glass Plate 1x0.7m (Bottom)	
20x20mm Aluminum Profile	https://coop.no/sortiment/obs-bygg/bad-og-kjokken-fbbffc80/ror-og-avlopsdeler/rkc-firkantror-alu-20x20?variantCode=207802
T250 IP66 Box	https://www.elektroimportoren.no/koblingsboks-obo-t250-ip66/1278235/Product.html?Event=searchlist
T350 IP66 Box	https://www.elektroimportoren.no/koblingsboks-obo-t350-ip66/1278236/Product.html
Electric Box, IP67~140x230x92	https://www.biltema.no/bygg/elinstallasjoner/elbokser/koplingsboks-2000034324
12V 20Ah Gel Battery	https://www.biltema.no/bil—mc/mc/mc-deler/mc-batterier/mc-batteri-gel-12v-20ah-2000029889
Buck Converter (5V)	
Buck Converter (12V)	
Main Power Switch	https://www.biltema.no/batutstyr/elektrisk-utstyr/strombrytere/batterivelger-1-2-12-off-2000041888
Neutral Rail	https://www.biltema.no/bygg/elinstallasjoner/installasjonsror/jordnoytral-skinne-2-stk-2000023132

Appendix C

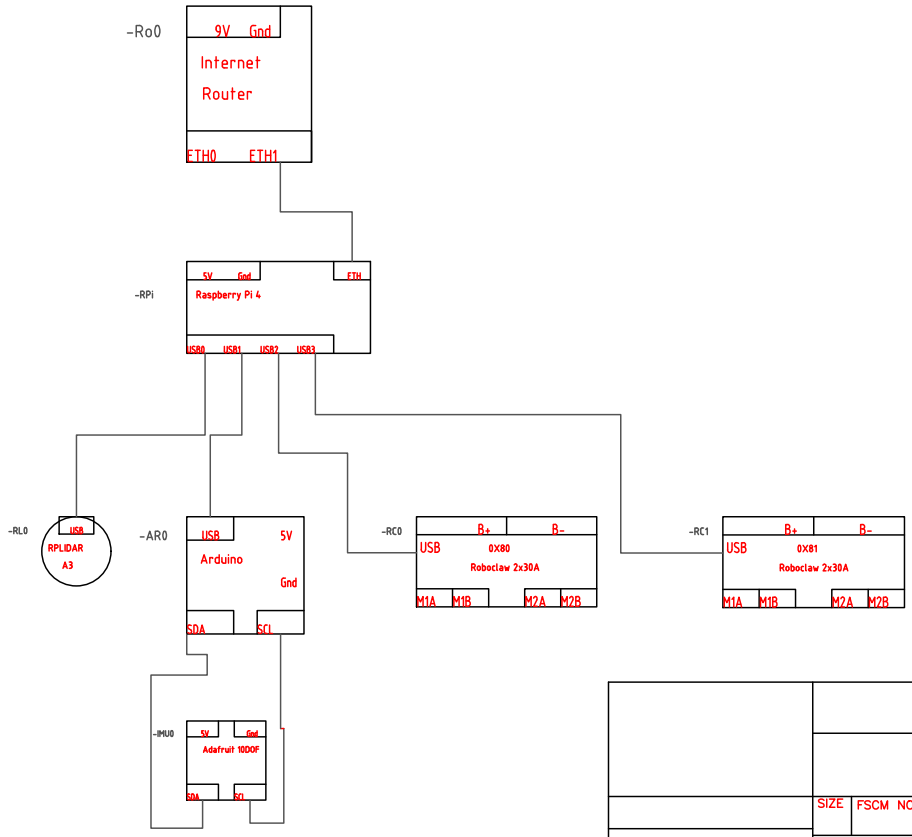
Electrical Drawings

REVISIONS				
XXX13	XXX14	XXX15	XXX16	XXX17



	BACHELOR – AUTODOCKING			
	Power Circuit			
	SIZE A4	FSCM NO	DRAWING NUMBER 1	REV 1
	SCALE 1		SHEET 1	

REVISIONS				
XXX13	XXX14	XXX15	XXX16	XXX17



	BACHELOR – AUTODOCKING			
	Communication Circuit			
	SIZE	FSCM NO	DRAWING NUMBER 1	REV 1
	SCALE 1		SHEET 2	

Appendix D

Progress Reports

Rapport Uke 04

Bachelor

Håkon Bjerkgaard Waldum

Ruben Ole Berg Natvik

Ruben Svedal Jørundland

Vebjørn Rimstad Wille

Prosjekt idé
Lage et system for autonom docking av skip vha primært Lidar.

Forrige uke status
Første uke

Nåverende status

Software
Har fått satt opp workspace for arbeid med ROS som kan brukes med IDEen PyCharm. Derunder også systemer for køring av flere noder samtidig, oprettelse av egendefinerte meldinger for ROS systemet.

Hardware
Deler til prototypen er kjøpt inn. 3D- printede deler er ikke klare enda, men forventes ferdig ila dagen idag.

Har sendt mail til veileder med forespørsel om elektrisk utsyr til prototype og avventer status.

Research
Har funnet ut hvordan skip normalt sett legger til kai.

Har begynt research for pathfinding.

Har funnet ulike verktøy som kan benyttes I forbindelse med docking vha lidar. Til dette er det funnet systemer kalt Hector-SLAM og AMCL.

Har researchet hvordan man kan benytte seg av ROS og de utgitte ros nodene I sammenheng med egne pakker.

Oppdagede problemer
Ingen problemer så langt.

Neste steg / uke

Teste flyteevene til prototype eventuelle utbedringer.

Undersøke hvilke data som kommer ut fra Hector-SLAM

Strukturere layout for GUI

Få tak I thrustere pronto.

Report Week 05

Bachelor

Håkon Bjerkgård Waldum

Ruben Natvik

Ruben Svedal Jørundland

Vebjørn Rimstad Wille

Prosjekt idé

Lage et system for autonom docking av skip vha primært Lidar.

Forrige Uke Status

Software

Har fått satt opp workspace for arbeid med ROS som kan brukes med IDEen PyCharm. Derunder også systemer for køring av flere noder samtidig, oprettelse av egendefinerte meldinger for ROS systemet.

Hardware

Deler til prototypen er kjøpt inn. 3D- printede deler er ikke klare enda, men forventes ferdig ila dagen idag.

Har sendt mail til veileder med forespørsel om elektrisk utsyr til prototype og avventer status.

Research

Har funnet ut hvordan skip normalt sett legger til kai. Har begynt research for pathfinding. Har funnet ulike verktøy som kan benyttes I forbindelse med docking vha lidar. Til dette er det funnet systemer kalt Hector-SLAM og AMCL. Har researchet hvordan man kan benytte seg av ROS og de utgitte ros nodene i sammenheng med egne pakker.

Oppdagede problemer

Ingen problemer så langt.

Nåværende status

Software

Har tatt et steg tilbake i forhold til software. Pakkene vi såg for oss å bruke trenger mer støtte rundt for å kunne benyttes.

Hardware

Fikk testet flytevnen til platformen. Denne var underdimensjonert. Revisjon 2, med økt flytevne under produksjon.

Oppdagede problemer.

Hector-SLAM sliter med lokalisering og mapping dersom yaw angle øker raskt. Mulig å fikse med gyroskop + extended kalman filter (sensor fusion) for bruk med odometry frame?

Neste uke

Ferdigstille forprosjektrapport.

Designe, printe og laminere deler til prototyp revisjon 2.

Starte utvikling av GUI.

Lage konkret materialliste for bestilling.

Undersøke om hector_localization kan brukes for sensor fusion.

Report Week 05

31/01/2020

Real Time Programming

Håkon Bjerkgaard Waldum

Ruben Natvik

Ruben Svedal Jørundland

Vebjørn Rimstad Wille

Prosjekt idé

Lage et system for autonom docking av skip vha primært Lidar. Det skal lages en fysisk prototype basert på design av en semi-submersible platform. Det vil bli benyttet en rimelig IMU for estimering av pitch, roll og yaw til å hjelpe med reguleringen av systemet. Prototypen vil styres med 4 motorer plassert i hjørnene på platformen, montert 45 grader på endene.

Prosjektet vil basere seg på at et system på kaia sender et kart over layouten på kaia, med informasjon om hvor fartøyet skal legge til kai.

Det vil også opprettes en GUI for overvåking av systemet som også vil fungere som systemet på kaia.

Forrige uke status

Software

Har tatt et steg tilbake i forhold til software. Pakkene vi såg for oss å bruke trenger mer støtte rundt for å kunne benyttes.

Hardware

Fikk testet flytevnen til platformen. Denne var underdimensjonert.Revisjon 2, med økt flytevneunder produksjon.

Oppdagede problemer.

Hector-SLAM sliter med lokalisering og mapping dersom yaw angle øker raskt. Mulig å fikse med gyroskop + extended kalman filter (sensor fusion)for bruk med odometry frame?

Nåværende status

Software

Har funnet et system for bruk av ROS mellom forskjellige enheter (over nettverk).

Har startet oppsett av Python med Flask for web basert GUI.

Jobber med sensor fusion for IMU til hector slam.

Hardware

Elektrisk dimensjonering er påbegynt og nærmes ferdig. Mangler eventuelle tap i motorkontrollere da disse ikke er bestemt enda.

Har satt opp en bestillingsliste og sendt til Ottar. Venter på endelig svar angående hvilke motorer som vil bli skaffet.

Fysisk konstruksjon er blitt vraket siden denne må dimmensjoneres på nytt grunnet tyngre batteri og motorer enn forventet.

Generelt

Har kommet i gang med rapporten. Her er det fylt inn om det som har vert aktuelt om teoretisk grunnlag hittil, samt elektriske komponenter som vil bli brukt.

Oppdagede problemer

Båten var underdimensjonert (rev 2) da det måtte benyttes tyngre motorer og batteri enn forventet.

Vanskelig å benytte IMU for estimering av roll pitch og yaw.

Next steps

Sette opp layout for GUI, samt undersøke hvordan man kan live streame data til en web basert GUI.

Opprette backend for GUI.

Se videre på sensor fusion i forbindelse med estimering av roll, pitch og yaw fra IMU. Finne ut hvordan dette kan benyttes sammen med Hector Slam.

Kjøpe inn deler og begynne konstruksjon av prototyp rev 3.

Report Week 06

07/02/2020

Real Time Programming

Håkon Bjerkgaard Waldum

Ruben Natvik

Ruben Svedal Jørundland

Vebjørn Rimstad Wille

Prosjekt idé

Lage et system for autonom docking av skip vha primært Lidar. Det skal lages en fysisk prototype basert på design av en semi-submersible platform. Det vil bli benyttet en rimelig IMU for estimering av pitch, roll og yaw til å hjelpe med reguleringen av systemet. Prototypen vil styres med 4 motorer plassert i hjørnene på platformen, montert 45 grader på endene.

Prosjektet vil basere seg på at et system på kaia sender et kart over layouten på kaia, med informasjon om hvor fartøyet skal legge til kai.

Det vil også opprettes en GUI for overvåking av systemet som også vil fungere som systemet på kaia.

Forrige uke status

Software

Har funnet et system for bruk av ROS mellom forskjellige enheter (over nettverk).

Har startet oppsett av Python med Flask for web basert GUI.

Jobber med sensor fusion for IMU til hector slam.

Hardware

Elektrisk dimensjonering er påbegynt og nærmes ferdig. Mangler eventuelle tap i motorkontrollere da disse ikke er bestemt enda.

Har satt opp en bestillingsliste og sendt til Ottar. Venter på endelig svar angående hvilke motorer som vil bli skaffet.

Fysisk konstruksjon er blitt vraket siden denne må dimmensjoneres på nytt grunnet tyngre batteri og motorer enn forventet.

Generelt

Har kommet i gang med rapporten. Her er det fylt inn om det som har vert aktuelt om teoretisk grunnlag hittil, samt elektriske komponenter som vil bli brukt.

Oppdagede problemer

Båten var underdimensjonert (rev 2) da det måtte benyttes tyngre motorer og batteri enn forventet.

Vanskelig å benytte IMU for estimering av roll pitch og yaw.

Nåværende status

Software

Har funnet en løsning for estimering av roll, pitch og yaw basert på IMU. Usikkert hvilken SLAM algoritme som er mest fornuftig å bruke av Hector SLAM og Gmapping, basert på manglende odometri translation sensorer.

Navigation stack er implementert, slik at det er mulig å sende ønsket mål til fartøyet.

GUI er fortsatt under utvikling, men har vert fremgang. Gjenstår dataoverføring mellom ros applikasjon og GUI, styling og layout.

Hardware

Mangler motorer og motorkontrollere. Har den fysiske konstruksjonen på pause til det er klart hvilke motorer vi får tilgang til.

Generelt

Kommet et lite stykke lengre i rapporten, men har vert lite innfylling siden forrige uke.

Oppdagede problemer

Har problemer med å estimere posisjon ved hjelp av IMU. Må finne andre løsninger for å lese av fart eller posisjon på fartøyet.

Mangler konkret plan for fysisk konstruksjon for å utvikle en matematisk modell for systemet i forbindelse med reguleringssystem som er tiltenkt som LQR kontroller.

Neste Uke

Finne ut konkret hvilke motorer og materialer vi får tilgang til og utarbeide en plan for videre arbeid.

Videre utvikling av GUI med hensyn til dynamisk data med Flask og Python

Videre testing av IMU for posisjonsestimering.

Undersøke muligheter for bruk av Hector SLAM i forbindelse med ekstern odometri.

Report Week 07

14/02/2020

Bachelorgruppe

Håkon Bjerkgaard Waldum

Ruben Natvik

Ruben Svedal Jørundland

Vebjørn Rimstad Wille

Prosjekt idé

Lage et system for autonom docking av skip vha primært Lidar. Det skal lages en fysisk prototype basert på design av en semi-submersible platform. Det vil bli benyttet en rimelig IMU for estimering av pitch, roll og yaw til å hjelpe med reguleringen av systemet. Prototypen vil styres med 4 motorer plassert i hjørnene på platformen, montert 45 grader på endene.

Prosjektet vil basere seg på at et system på kaia sender et kart over layouten på kaia, med informasjon om hvor fartøyet skal legge til kai.

Det vil også opprettes en GUI for overvåking av systemet som også vil fungere som systemet på kaia.

Forrige uke status

Software

Har funnet en løsning for estimering av roll, pitch og yaw basert på IMU. Usikkert hvilken SLAM algoritme som er mest fornuftig å bruke av Hector SLAM og Gmapping, basert på manglende odometri translation sensorer.

Navigation stack er implementert, slik at det er mulig å sende ønsket mål til fartøyet.

GUI er fortsatt under utvikling, men har vert fremgang. Gjenstår dataoverføring mellom ros applikasjon og GUI, styling og layout.

Hardware

Mangler motorer og motorkontrollere. Har den fysiske konstruksjonen på pause til det er klart hvilke motorer vi får tilgang til.

Generelt

Kommet et lite stykke lengre i rapporten, men har vert lite innfylling siden forrige uke.

Oppdagede problemer

Har problemer med å estimere posisjon ved hjelp av IMU. Må finne andre løsninger for å lese av fart eller posisjon på fartøyet.

Mangler konkret plan for fysisk konstruksjon for å utvikle en matematisk modell for systemet i forbindelse med reguleringssystem som er tiltenkt som LQR kontroller.

Nåværende status

Software

Fikk oprettet et transformasjonstre som kan benyttes sammen med hector slam for å få benytte odometri.

Lite videre utvikling av gui og resten av softwaren grunnet kort uke i forbindelse med industri 4.0.

Hardware

Har fått klarhet i hvilke motorer som vil være tilgjengelige. Startet arbeid med kalkulasjoner innenfor hydrostatikk for platformen.

Generelt

Har begynt arbeid med å oversette nåværende rapport til engelsk

Oppdagede problemer

Problemer med oppdatering av dynamisk data i GUI.

Neste Uke

Utregninger i forbindelse med prototypens stabilitet.

Konstruere prototype.

Utvikle matematisk modell av konstruksjonen for regulator.

Fikse dynamisk data i GUI, teste med live data fra Lidar.

Se på muligheter for å benytte flere IMU for sensor fusion av accelerometer data.

Report Week 08

21/02/2020

Bachelorgruppe

Håkon Bjerkgaard Waldum

Ruben Natvik

Ruben Svedal Jørundland

Vebjørn Rimstad Wille

Prosjekt idé

Lage et system for autonom docking av skip vha primært Lidar. Det skal lages en fysisk prototype basert på design av en semi-submersible platform. Det vil bli benyttet en rimelig IMU for estimering av pitch, roll og yaw til å hjelpe med reguleringen av systemet. Prototypen vil styres med 4 motorer plassert i hjørnene på platformen, montert 45 grader på endene.

Prosjektet vil basere seg på at et system på kaia sender et kart over layouten på kaia, med informasjon om hvor fartøyet skal legge til kai.

Det vil også opprettes en GUI for overvåking av systemet som også vil fungere som systemet på kaia.

Forrige uke status

Software

Fikk oprettet et transformasjonstre som kan benyttes sammen med hector slam for å få benytte odometri.

Lite videre utvikling av gui og resten av softwaren grunnet kort uke i forbindelse med industri 4.0.

Hardware

Har fått klarhet i hvilke motorer som vil være tilgjengelige. Startet arbeid med kalkulasjoner innenfor hydrostatikk for platformen.

Generelt

Har begynt arbeid med å oversette nåværende rapport til engelsk

Oppdagede problemer

Problemer med oppdatering av dynamisk data i GUI.

Nåværende status

Software

GUI oppdateres nå med live data fra lidar over nettverk. Grunnstruktur og rammeverk for GUI er ferdig, mangler eventuell user interaction og overføring av kart.

Jobber med løsning for å samkjøre flere IMUer for estimering av translation av platformen.

Jobber med system for kalibrering av de forskjellige prosessene og sensorene i systemet.

Hardware

Utregninger for stabilitet og flytevne for konstruksjonen er gjort og kontrollert med sammarbeid med en ships-design student samt Håvard V. Lien.

Har påbegynt konstruksjon av konstruksjonen. Rammeverk og platform er ferdig kontruert, venter på deler som må 3D-printes.

Generelt

Fikk ikke påbegynt matematisk modellering av prototypen, dette planlegges for neste uke.

Har fylt ut mer i rapport, jobber med å fylle inn kapitler om hydrostatikk som omhandler utregningene for stabilitet utført denne uken.

Oppdagede problemer

Ingen problemer vi har satt oss fast på denne uken.

Neste Uke

Fullføre og flyt-teste prototypen.

Utvikle matematisk modell av konstruksjonen for regulator. (Ordne møte med Aleksander for diskusjon rundt løsning av dette)

Utarbeide fullstendig plan for det elektriske systemet.

Ferdigstille og implementere kalibreringsprogram.

Se på muligheter rundt videresending av kart til GUI.

Feridgstille og teste løsning for fusion av IMU sensorer.

Report Week 09

28/02/2020

Bachelorgruppe

Håkon Bjerkgaard Waldum

Ruben Natvik

Ruben Svedal Jørundland

Vebjørn Rimstad Wille

Prosjekt idé

Lage et system for autonom docking av skip vha primært Lidar. Det skal lages en fysisk prototype basert på design av en semi-submersible platform. Det vil bli benyttet en rimelig IMU for estimering av pitch, roll og yaw til å hjelpe med reguleringen av systemet. Prototypen vil styres med 4 motorer plassert i hjørnene på platformen, montert 45 grader på endene.

Prosjektet vil basere seg på at et system på kaia sender et kart over layouten på kaia, med informasjon om hvor fartøyet skal legge til kai.

Det vil også opprettes en GUI for overvåking av systemet som også vil fungere som systemet på kaia.

Forrige uke status

Software

GUI oppdateres nå med live data fra lidar over nettverk. Grunnstruktur og rammeverk for GUI er ferdig, mangler eventuell user interaction og overføring av kart.

Jobber med løsning for å samkjøre flere IMUer for estimering av translation av platformen.

Jobber med system for kalibrering av de forskjellige prosessene og sensorene i systemet.

Hardware

Utregninger for stabilitet og flytevne for konstruksjonen er gjort og kontrollert med sammarbeid med en ships-design student samt Håvard V. Lien.

Har påbegynt konstruksjon av konstruksjonen. Rammeverk og platform er ferdig kontruert, venter på deler som må 3D-printes.

Generelt

Fikk ikke påbegynt matematisk modellering av prototypen, dette planlegges for neste uke.

Har fylt ut mer i rapport, jobber med å fylle inn kapitler om hydrostatikk som omhandler utregningene for stabilitet utført denne uken.

Nåværende status

Software

Initiell løsning av mapping av omgivelser og lokalisering av platform via Lidar er ferdigstilt, men må testes i reelt scenario.

Struktur for obstacle avoidance og pathing er ferdig, med enkelte bugs som må utbedres. Må testes.

Gjenstår arbeid med regulator og sensorikk.

Rammeverk for GUI er ferdig, gjenstår impementering av user interaction, men dette er satt på vent til det trengs å implementeres.

Hardware

Har kjørt flyttest på platformen. Den flyter og vil ikke tippe, men gir store utslag på roll og er veldig sensitiv for vektfordeling i ballastrør og komponenter på toppen av platformen.

Har fått tilgang til en JetsonTX2. Denne virker som den er veldig godt egnet til formålet da vi nå kan kjøre alt av prosessering lokalt på platformen, uten å være avhengig av kommunikajson over trådløst nettverk.

Generelt

Gjenstår mye arbeid, men så langt er arbeidet godt i rute. Problemer med konstruksjonen

Oppdagede problemer

Konstruksjonen er stabil og velter ikke, men gir store utslag på roll. Skal teste med stabilisatorfinner på bunn for å dempe oscillasjonene. Konstruksjonen er også for sensitiv i forhold til vektfordeling på komponentene som skal monteres på topplaten. Ser etter løsninger for dette.

Neste Uke

Stabilisator-finner til reduksjon av roll.

Mandag, montere stabilisatorfinner, endre avstivingsmekanisme for vertikale rør.

Mandag eller tirsdag, ny flyt-test i sjø.

Jobbe videre med modellering av systemet og navigation stack.

Report Week 10

09/03/2020

Bachelorgruppe

Håkon Bjerkgaard Waldum

Ruben Natvik

Ruben Svedal Jørundland

Vebjørn Rimstad Wille

Prosjekt idé

Lage et system for autonom docking av skip vha primært Lidar. Det skal lages en fysisk prototype basert på design av en semi-submersible platform. Det vil bli benyttet en rimelig IMU for estimering av pitch, roll og yaw til å hjelpe med reguleringen av systemet. Prototypen vil styres med 4 motorer plassert i hjørnene på platformen, montert 45 grader på endene.

Prosjektet vil basere seg på at et system på kaia sender et kart over layouten på kaia, med informasjon om hvor fartøyet skal legge til kai.

Det vil også opprettes en GUI for overvåking av systemet som også vil fungere som systemet på kaia.

Forrige uke status

Software

Initiell løsning av mapping av omgivelser og lokalisering av platform via Lidar er ferdigstilt, men må testes i reelt scenario.

Struktur for obstacle avoidance og pathing er ferdig, med enkelte bugs som må utbedres. Må testes.

Gjenstår arbeid med regulator og sensorikk.

Rammeverk for GUI er ferdig, gjenstår impementering av user interaction, men dette er satt på vent til det trengs å implementeres.

Hardware

Har kjørt flyttest på platformen. Den flyter og vil ikke tippe, men gir store utslag på roll og er veldig sensitiv for vektfordeling i ballastrør og komponenter på toppen av platformen.

Har fått tilgang til en JetsonTX2. Denne virker som den er veldig godt egnet til formålet da vi nå kan kjøre alt av prosessering lokalt på platformen, uten å være avhengig av kommunikajson over trådløst nettverk.

Generelt

Gjenstår mye arbeid, men så langt er arbeidet godt i rute. Problemer med konstruksjonen

Oppdagede problemer

Konstruksjonen er stabil og velter ikke, men gir store utslag på roll. Skal teste med stabilisatorfinner på bunn for å dempe oscillasjonene. Konstruksjonen er også for sensitiv i forhold til vektfordeling på komponentene som skal monteres på topplaten. Ser etter løsninger for dette.

Nåværende status

Software

Utvidet kalkulator script til å ta hensyn til alle plan i forhold til posisjonering av massesenter, flytsenter og metasenter.

Hardware

Har bygd tralle for å sjøsette og hente ut plattformen fra vannet.

Har begynt å bygge et lite kjøretøy (bil) for testing av programmoduler uten å måtte benytte plattformen i havet.

Generelt

Har jobbet videre med modellering for systemet.

Oppdagede problemer

Har løst problemer med store utslag for roll motion, men konstruksjonen er fortsatt veldig sensitiv for vektbalanseringen på toppen av plattformen. Vurderer å prøve pontonger på siden.

Under utvikling av testkjøretøyet ble det oppdaget at Jetson brettet ikke ville kommunisere med roboclaw motorcontrollerene. Usikkert hva som er årsaken, men tenker å benytte en raspberry som seriell slave modul.

Neste Uke

Utvikle og teste system for å redusere effekten av skeiv vektfordeling på plattformen.

Teste de tidligere utviklede software systemene på testkjøretøy.

Jobbe videre med modellering av system og metoder for sensor fusion.

Kan gjøres ved senere tidspunkt: Teste og hente inn data angående kraftkurven til motorene (Kraft gitt av spenning / strøm ut til motor).

Report Week 11

16/03/2020

Bachelorgruppe

Håkon Bjerkgaard Waldum

Ruben Natvik

Ruben Svedal Jørundland

Vebjørn Rimstad Wille

Prosjekt idé

Lage et system for autonom docking av skip vha primært Lidar. Det skal lages en fysisk prototype basert på design av en semi-submersible platform. Det vil bli benyttet en rimelig IMU for estimering av pitch, roll og yaw til å hjelpe med reguleringen av systemet. Prototypen vil styres med 4 motorer plassert i hjørnene på platformen, montert 45 grader på endene.

Prosjektet vil basere seg på at et system på kaia sender et kart over layouten på kaia, med informasjon om hvor fartøyet skal legge til kai.

Det vil også opprettes en GUI for overvåking av systemet som også vil fungere som systemet på kaia.

Forrige uke status

Software

Utvidet kalkulator script til å ta hensyn til alle plan i forhold til posisjonering av massesenter, flytsenter og metasenter.

Hardware

Har bygd tralle for å sjøsette og hente ut plattformen fra vannet.

Har begynt å bygge et lite kjøretøy (bil) for testing av programmoduler uten å måtte benytte plattformen i havet.

Generelt

Har jobbet videre med modellering for systemet.

Oppdagede problemer

Har løst problemer med store utslag for roll motion, men konstruksjonen er fortsatt veldig sensitiv for vektbalanseringen på toppen av plattformen. Vurderer å prøve pontonger på siden.

Under utvikling av testkjøretøyet ble det oppdaget at Jetson brettet ikke ville kommunisere med roboclaw motorcontrollerene. Usikkert hva som er årsaken, men tenker å benytte en raspberry som seriell slave modul.

Nåværende status

Software

Har klart å mappe et rom, sende dette tilbake til robot og teste AMCL. Vi ser at roll pitch yaw, fungerer bra, men vi er avhengige av en form for odometri for translation og velocity for at AMCL skal fungere.

Hardware

På grunn av Koronasituasjonen, har vi hentet ut alt av hardware til prototypen og satt dette i garasjen til Håkon. Vi planlegger å få tatt en test av flytevnen til plattformen ved senere anledning.

Generelt

Jobber med metoder for sensor fusion av flere IMU. Ser på metoder for fusing av både rådata og kombinert orientation data.

Har satt opp kalkulasjoner for plattform uten ballastrør og batteri i bunn av plattformen.

Har sett videre på modellering og simulert motion dynamics i Matlab.

Oppdagede problemer

Har funnet en ros package som kanskje kan hjelpe med velocity og translation odometri for AMCL kalt laser scan matcher, men er foreløpig bare utgitt for ROS kinetic distro.

Neste Uke

Løse AMCL problematikken. Skal teste hvordan laser_scan_matcher fungerer i forhold til dette.

Kjøpe batteri og vanntett beholder til batteriet (lage handleliste).

Lage plan for eventuelle justeringer og endringer av oppgaven dersom koronasituasjonen blir vedvarende. Få satt opp et møte med Ottar for å diskutere dette.

Utregninger for benytting av eksisterende prototyp med batteri i bunn.

Fortsette arbeid med sensor fusion og modellering.

Report Week 12

20/03/2020

Bachelorgruppe

Håkon Bjerkgaard Waldum

Ruben Natvik

Ruben Svedal Jørundland

Vebjørn Rimstad Wille

Prosjekt idé

Lage et system for autonom docking av skip vha primært Lidar. Det skal lages en fysisk prototype basert på design av en semi-submersible platform. Det vil bli benyttet en rimelig IMU for estimering av pitch, roll og yaw til å hjelpe med reguleringen av systemet. Prototypen vil styres med 4 motorer plassert i hjørnene på platformen, montert 45 grader på endene.

Prosjektet vil basere seg på at et system på kaia sender et kart over layouten på kaia, med informasjon om hvor fartøyet skal legge til kai.

Det vil også opprettes en GUI for overvåking av systemet som også vil fungere som systemet på kaia.

Forrige uke status

Software

Har klart å mappe et rom, sende dette tilbake til robot og teste AMCL. Vi ser at roll pitch yaw, fungerer bra, men vi er avhengige av en form for odometri for translation og velocity for at AMCL skal fungere.

Hardware

På grunn av Koronasituasjonen, har vi hentet ut alt av hardware til prototypen og satt dette i garasjen til Håkon. Vi planlegger å få tatt en test av flytevnen til plattformen ved senere anledning.

Generelt

Jobber med metoder for sensor fusion av flere IMU. Ser på metoder for fusing av både rådata og kombinert orientation data.

Har satt opp kalkulasjoner for plattform uten ballastrør og batteri i bunn av plattformen.

Har sett videre på modellering og simulert motion dynamics i Matlab.

Oppdagede problemer

Har funnet en ros package som kanskje kan hjelpe med velocity og translation odometri for AMCL kalt laser scan matcher, men er foreløpig bare utgitt for ROS kinetic distro.

Nåværende status

Software

Mye av AMCL-problematikken fra forrige uke er løst. Laser_scan_matcher er implementert og hjelper veldig med posisjonering i kartet. AMCL fungerer nå ganske bra.

Hardware

Alt av nødvendig utstyr for videre arbeid med prototype er innhandlet. Batteri, bokser etc. Alt er nå klart til montering.

Utregninger med batteri i bunnen av prototypen ble gjort, og ble funnet å gi en god flyt-evne samt god stabilitet. Dette ble testet i praksis, og virker å fungere veldig bra. Prototypen er fortsatt stabil, og med lavere vertikalrør har dette gitt mindre utslag på skeiv vektfordeling. Det anses derfor som et bra resultat så langt, og kan påbegynne montering av elektriske komponenter.

Generelt

Koronasituasjonen er diskutert med Ottar. Om det verste skulle skje, vil oppgaven skifte fokus fra der vi har nådd og over til en simulert løsning mot slutten. Sånn som situasjonen er nå vil oppgaven fortsette som normalt.

Arbeid med modellering og sensor fusion er pågående. Har funnet en løsning for selve modellen, men parameter og dempingskoeffisienter må finnes eksperimentelt.

Oppdagede problemer

Ingen særlige problemer har oppstått.

Neste Uke

Fokus på hardware og konstruksjon, samt kjøre ny test av prototype med motor og det meste av elektriske systemer tilkoblet.

I disse testene skal vi hente ut nødvendig data for modellering (drag-koeffisienter etc.) samt hvordan kurven til motor er i forhold til spenning kontra kraft så godt det går.

Lage et enkelt styresystem for testing av motorer påmontert prototype for å kunne manuelt kjøre litt fram og tilbake. Dette for å kunne samle inn data

Report Week 13

27/03/2020

Bachelorgruppe

Håkon Bjerkgaard Waldum

Ruben Natvik

Ruben Svedal Jørundland

Vebjørn Rimstad Wille

Prosjekt idé

Lage et system for autonom docking av skip vha primært Lidar. Det skal lages en fysisk prototype basert på design av en semi-submersible platform. Det vil bli benyttet en rimelig IMU for estimering av pitch, roll og yaw til å hjelpe med reguleringen av systemet. Prototypen vil styres med 4 motorer plassert i hjørnene på platformen, montert 45 grader på endene.

Prosjektet vil basere seg på at et system på kaia sender et kart over layouten på kaia, med informasjon om hvor fartøyet skal legge til kai.

Det vil også opprettes en GUI for overvåking av systemet som også vil fungere som systemet på kaia.

Forrige uke status

Software

Mye av AMCL-problematikken fra forrige uke er løst. Laser_scan_matcher er implementert og hjelper veldig med posisjonering i kartet. AMCL fungerer nå ganske bra.

Hardware

Alt av nødvendig utstyr for videre arbeid med prototype er innhandlet. Batteri, bokser etc. Alt er nå klart til montering.

Utregninger med batteri i bunnen av prototypen ble gjort, og ble funnet å gi en god flyt-evne samt god stabilitet. Dette ble testet i praksis, og virker å fungere veldig bra. Prototypen er fortsatt stabil, og med lavere vertikalrør har dette gitt mindre utslag på skeiv vektfordeling. Det anses derfor som et bra resultat så langt, og kan påbegynne montering av elektriske komponenter.

Generelt

Koronasituasjonen er diskutert med Ottar. Om det verste skulle skje, vil oppgaven skifte fokus fra der vi har nådd og over til en simulert løsning mot slutten. Sånn som situasjonen er nå vil oppgaven fortsette som normalt.

Arbeid med modellering og sensor fusion er pågående. Har funnet en løsning for selve modellen, men parameter og dempingskoeffisienter må finnes eksperimentelt.

Oppdagede problemer

Ingen særlige problemer har oppstått.

Nåværende status

Software

Scriptet for manuell kjøring er satt opp og tilsynelatende oppe og går. Har oppdaget et problem med denne da der er en massiv forsinkelse i prosesseringen av innkommende data for å sende kommandoer ut til motorene.

Hardware

Har koblet opp det meste av elektronikk på plattformen, batteriboksen er tilsynelatende vanntett og fylt med olje. Vanntettingen må testes på ca 1m dyp. Har ikke rukket å teste prototypen med systemene oppe å gå enda.

Generelt

Har satt opp en løsning for å finne modellparametere med GA. Dette krever at vi har tilgang på måledata angående kraft fra motorene ved en gitt spenning, samt velocity responsen til platformen for de gitte inputs. Har testet dette scriptet med en testmodell med tilfeldige parametre, og GA finner parameterene innenfor ca 10% av faktisk verdi.

Oppdagede problemer

En av motorene til konstruksjonen har mest trolig dårlig kontakt i motorhuset. Motoren har altså ingen kontakt mellom + og – med mindre motorhuset vris å vendes litt på.

Som nevnt går scriptet for å sende kommandoer ut til motorene alt for sakte. Prøver pr nå å finne ut hva som er problemet.

Neste Uke

Fullføre oppkobling av elektronikken på prototypen, samt festeanordning for prototypen. Utføre tester for respons ved ulike manøvrer samt bollard pull for å finne kraftkurven til motorene.

For målingene sin del er det ønskelig å se litt på om det kan benyttes en liten gps for måling av posisjonsdata, siden det er meldt dårlig vær neste uke med tanke på at lidaren ikke er vanntett.

Dersom alt går som planlagt og vi får hentet inn målingene mandag eller tirsdag vil vi fortsette arbeid med modellering for kontroll og feedback system, samt ros navigasjon-stack for pathing.

Report Week 14

06/04/2020

Bachelorgruppe

Håkon Bjerkgaard Waldum

Ruben Natvik

Ruben Svedal Jørundland

Vebjørn Rimstad Wille

Prosjekt idé

Lage et system for autonom docking av skip vha primært Lidar. Det skal lages en fysisk prototype basert på design av en semi-submersible platform. Det vil bli benyttet en rimelig IMU for estimering av pitch, roll og yaw til å hjelpe med reguleringen av systemet. Prototypen vil styres med 4 motorer plassert i hjørnene på platformen, montert 45 grader på endene.

Prosjektet vil basere seg på at et system på kaia sender et kart over layouten på kaia, med informasjon om hvor fartøyet skal legge til kai.

Det vil også opprettes en GUI for overvåking av systemet som også vil fungere som systemet på kaia.

Forrige uke status

Software

Scriptet for manuell kjøring er satt opp og tilsynelatende oppe og går. Har oppdaget et problem med denne da der er en massiv forsinkelse i prosesseringen av innkommende data for å sende kommandoer ut til motorene.

Hardware

Har koblet opp det meste av elektronikk på plattformen, batteriboksen er tilsynelatende vanntett og fylt med olje. Vanntettingen må testes på ca 1m dyp. Har ikke rukket å teste prototypen med systemene oppe å gå enda.

Generelt

Har satt opp en løsning for å finne modellparametere med GA. Dette krever at vi har tilgang på måledata angående kraft fra motorene ved en gitt spenning, samt velocity responsen til platformen for de gitte inputs. Har testet dette scriptet med en testmodell med tilfeldige parametre, og GA finner parameterene innenfor ca 10% av faktisk verdi.

Oppdagede problemer

En av motorene til konstruksjonen har mest trolig dårlig kontakt i motorhuset. Motoren har altså ingen kontakt mellom + og – med mindre motorhuset vris å vendes litt på.

Som nevnt går scriptet for å sende kommandoer ut til motorene alt for sakte. Prøver pr nå å finne ut hva som er problemet.

Nåværende status

Software

GUI er nesten ferdig. Her gjenstår det å finne en måte å vise kartet samt posisjon i gui. I tillegg gjenstår det å lage ferdig TCP clienten som er ansvarlig for å sende kommandoer fra GUI til server på båten.

Har fått satt opp systemer for å mappe test området fra båten.

Hardware

Fullført alt av elektriske systemer ombord på båten utenom Jetson.

Fysisk konstruksjon er oppe og går. En grov vektbalansering er utført. Vil måtte finjusteres etter at Jetson monteres på.

Generelt

Har fått utført responstester for ulike manøvre. Har også utført bollard pull test.

GA til bruk for å finne modellparametere for båten er under arbeid.

Oppdagede problemer

I forbindelse med matematisk modellering av systemet for LQR kontroller / Kalman filter kan det se ut som om systemmodellen vi har basert utregningene på ikke representerer systemet tilstrekkelig. Mulig at modellen må endres, eller at vi må se på andre løsninger for system kontroll.

Neste Uke

Jobbe videre med navigation stack og logikk rundt posisjonering og path planning.

Jobbe videre med GUI.

Jobbe videre med GA og systemmodell. Eventuelt se på andre løsninger dersom det viser seg å ikke være hensiktsmessig å lage LQR kontroller for systemet.

Report Week 15

12/04/2020

Bachelorgruppe

Håkon Bjerkgaard Waldum

Ruben Natvik

Ruben Svedal Jørundland

Vebjørn Rimstad Wille

Prosjekt idé

Lage et system for autonom docking av skip vha primært Lidar. Det skal lages en fysisk prototype basert på design av en semi-submersible platform. Det vil bli benyttet en rimelig IMU for estimering av pitch, roll og yaw til å hjelpe med reguleringen av systemet. Prototypen vil styres med 4 motorer plassert i hjørnene på platformen, montert 45 grader på endene.

Prosjektet vil basere seg på at et system på kaia sender et kart over layouten på kaia, med informasjon om hvor fartøyet skal legge til kai.

Det vil også opprettes en GUI for overvåking av systemet som også vil fungere som systemet på kaia.

Forrige uke status

Software

GUI er nesten ferdig. Her gjenstår det å finne en måte å vise kartet samt posisjon i gui. I tillegg gjenstår det å lage ferdig TCP clienten som er ansvarlig for å sende kommandoer fra GUI til server på båten.

Har fått satt opp systemer for å mappe test området fra båten.

Hardware

Fullført alt av elektriske systemer ombord på båten utenom Jetson.

Fysisk konstruksjon er oppe og går. En grov vektbalansering er utført. Vil måtte finjusteres etter at Jetson monteres på.

Generelt

Har fått utført responstester for ulike manøvre. Har også utført bollard pull test.

GA til bruk for å finne modellparametere for båten er under arbeid.

Oppdagede problemer

I forbindelse med matematisk modellering av systemet for LQR kontroller / Kalman filter kan det se ut som om systemmodellen vi har basert utregningene på ikke representerer systemet tilstrekkelig. Mulig at modellen må endres, eller at vi må se på andre løsninger for system kontroll.

Nåværende status

Software

Begrenset mengde arbeid har blitt utført med nav stack og logikk rundt den pga vær.

GUI er utviklet videre og har nå fått implementert TCP client for sending av kommandoer til fartøyet, samt visning av kart og lokasjon.

Hardware

Det ser ut til at vi ikke trenger å benytte Jetson til prototypen da kommunikasjonen virker stabil og vi kan kjøre tung prosessering på ekstern laptop.

Generelt

Rapport har blitt satt litt i fokus denne uken siden været ikke har vert godt nokk til å få testet prototypen denne uken og har blitt arbeidet videre med.

Har funnet en forbedret løsning for datakonvertering for bruk med GA. Har fått gode resultater for modellen med individuelle dataset, men grunnet drift fra vann og havstrømmer blir parameterene fra testene litt forskjellige fra dataset til dataset. Jobber nå med LQR kontrol system til nye tester med kalibrering for drift kan tas.

Oppdagede problemer

PLA var ikke ideelt for brakettene til prototypen da disse ikke tåler særlig godt saltvann og begynner å bli sprø og skjør.

Neste Uke

Kjøre full autonom test

Hente nye dataserier der vi kan ta hensyn til drift fra vannstrøm og vind.

Skrive kodeimplementasjon for kalmanfilter i sensor feedback delen av systemet.

Dårlig vær betyr rapportskriving, fint vær blir til testing av systemet.

Gå igjennom rapport og sette opp en mer detaljert oversikt over hva som skal skrives hvor osv.

Report Week 16

19/04/2020

Bachelorgruppe

Håkon Bjerkgaard Waldum

Ruben Natvik

Ruben Svedal Jørundland

Vebjørn Rimstad Wille

Prosjekt idé

Lage et system for autonom docking av skip vha primært Lidar. Det skal lages en fysisk prototype basert på design av en semi-submersible platform. Det vil bli benyttet en rimelig IMU for estimering av pitch, roll og yaw til å hjelpe med reguleringen av systemet. Prototypen vil styres med 4 motorer plassert i hjørnene på platformen, montert 45 grader på endene.

Prosjektet vil basere seg på at et system på kaia sender et kart over layouten på kaia, med informasjon om hvor fartøyet skal legge til kai.

Det vil også opprettes en GUI for overvåking av systemet som også vil fungere som systemet på kaia.

Forrige uke status

Software

Begrenset mengde arbeid har blitt utført med nav stack og logikk rundt den pga vær.

GUI er utviklet videre og har nå fått implementert TCP client for sending av kommandoer til fartøyet, samt visning av kart og lokasjon.

Hardware

Det ser ut til at vi ikke trenger å benytte Jetson til prototypen da kommunikasjonen virker stabil og vi kan kjøre tung prosessering på ekstern laptop.

Generelt

Rapport har blitt satt litt i fokus denne uken siden været ikke har vert godt nokk til å få testet prototypen denne uken og har blitt arbeidet videre med.

Har funnet en forbedret løsning for datakonvertering for bruk med GA. Har fått gode resultater for modellen med individuelle dataset, men grunnet drift fra vann og havstrømmer blir parameterene fra testene litt forskjellige fra dataset til dataset. Jobber nå med LQR kontrol system til nye tester med kalibrering for drift kan tas.

Oppdagede problemer

PLA var ikke ideelt for brakettene til prototypen da disse ikke tåler særlig godt saltvann og begynner å bli sprø og skjør.

Nåværende status

Software

Har fått implementert kalmanfilter for estimering av body velocities.

Har begynt testing av GUI. Mangler motor og hastighetsinformasjon fra server til client.

Hardware

Ingenting nytt på hardware siden av prosjektet denne uken

Generelt

Har jobbet videre med rapport og satt opp en generell struktur for informasjon som skal fylles inn.

Jobber med bearbeiding av nye målesett for mer presis modell av systemet.

Jobber med simulering og tuning av kontrollsystem.

Oppdagede problemer

Serverside GUI socket script har en feil som gjør at ruteren stopper.

Guidance systemet lar ikke platformen kjøre sidelengs og baklengs.

Neste Uke

Legge opp bedre kabelføringer på prototype.

Lage ny brakett for å holde lidaren.

Finne en løsning for å la platformen kjøre sidelengs og baklengs.

Løse problemet med server side kommunikasjon til gui.

Utvikle og implementere hastighetsregulator for platformen.

Testing

Rapport

Report Week 17

26/04/2020

Bachelorgruppe

Håkon Bjerkgaard Waldum

Ruben Natvik

Ruben Svedal Jørundland

Vebjørn Rimstad Wille

Prosjekt idé

Lage et system for autonom docking av skip vha primært Lidar. Det skal lages en fysisk prototype basert på design av en semi-submersible platform. Det vil bli benyttet en rimelig IMU for estimering av pitch, roll og yaw til å hjelpe med reguleringen av systemet. Prototypen vil styres med 4 motorer plassert i hjørnene på platformen, montert 45 grader på endene.

Prosjektet vil basere seg på at et system på kaia sender et kart over layouten på kaia, med informasjon om hvor fartøyet skal legge til kai.

Det vil også opprettes en GUI for overvåking av systemet som også vil fungere som systemet på kaia.

Forrige uke status

Software

Har fått implementert kalmanfilter for estimering av body velocities.

Har begynt testing av GUI. Mangler motor og hastighetsinformasjon fra server til client.

Hardware

Ingenting nytt på hardware siden av prosjektet denne uken

Generelt

Har jobbet videre med rapport og satt opp en generell struktur for informasjon som skal fylles inn.

Jobber med bearbeiding av nye målesett for mer presis modell av systemet.

Jobber med simulering og tuning av kontrollsystem.

Oppdagede problemer

Serverside GUI socket script har en feil som gjør at ruteren stopper.

Guidance systemet lar ikke platformen kjøre sidelengs og baklengs.

Nåværende status

Software

Små implementasjonsfeil på kalmanfilter. Skal bytte hvilke sensor meldinger som blir brukt for måledata.

Local path planner har blitt byttet ut til et nytt system som lar platformen bevege seg fritt i alle retninger.

Det ser ut til at GUI får ruteren til å overbelastes og stoppe.

Hardware

Har utført utbedringer på konstruksjon i forhold til kabelføringer og lignende.

Generelt

Har funnet gode modeller av systemet ved bruk av GA.

Har utført tester av autonom kjøring.

Oppdagede problemer

Kommunikasjon til GUI må endres for å begrense mengden data den sender.

Neste Uke

Fintuning av system og testing av endelig løsning.

Løse problematikk med GUI.

Lagge video av systemet.

Report Week 18

03/05/2020

Bachelorgruppe

Håkon Bjerkgaard Waldum

Ruben Natvik

Ruben Svedal Jørundland

Vebjørn Rimstad Wille

Prosjekt idé

Lage et system for autonom docking av skip vha primært Lidar. Det skal lages en fysisk prototype basert på design av en semi-submersible platform. Det vil bli benyttet en rimelig IMU for estimering av pitch, roll og yaw til å hjelpe med reguleringen av systemet. Prototypen vil styres med 4 motorer plassert i hjørnene på platformen, montert 45 grader på endene.

Prosjektet vil basere seg på at et system på kaia sender et kart over layouten på kaia, med informasjon om hvor fartøyet skal legge til kai.

Det vil også opprettes en GUI for overvåking av systemet som også vil fungere som systemet på kaia.

Forrige uke status

Software

Små implementasjonsfeil på kalmanfilter. Skal bytte hvilke sensor meldinger som blir brukt for måledata.

Local path planner har blitt byttet ut til et nytt system som lar platformen bevege seg fritt i alle retninger.

Det ser ut til at GUI får ruteren til å overbelastes og stoppe.

Hardware

Har utført utbedringer på konstruksjon i forhold til kabelføringer og lignende.

Generelt

Har funnet gode modeller av systemet ved bruk av GA.

Har utført tester av autonom kjøring.

Oppdagede problemer

Kommunikasjon til GUI må endres for å begrense mengden data den sender.

Nåværende status

Software

Software har blitt finjustert slik at plattformen nå fungerer så optimalt som vi klarer å få den til å gjøre.

Har fått byttet til laser scan matcher feedback for kalman filter. Fungerer mye bedre enn tidligere feedback system.

Hardware

Har kjøpt ny ruter. GUI fungerer nå som planlagt.

Generelt

Vi har fått kjørt mesteparten av planlagte systemtester. Dette inkluderer path following tester, go to dock tester, drift test, open vs closed loop feedback tester samt systemtester i større omgivelser.

Vi har også utført uoffisielle tester for kollisjonsunnvikelse, med dårlige resultater for TEB Local Planner systemet. Det ble derfor også utført kollisjonsunnvikelse tester med andre Local Planner system.

Oppdagede problemer

Local Planner systemene som har blitt testet fungerer dårlig for kollisjonsunnvikelse. Trolig har dette noe med at de er designet for raskere roboter enn vår platform da de har begrensinger på hvor langt frem i tid de kan simulere.

Neste Uke

Utføre endelige kollisjonsunnvikelsestester med TEB Local Planner

Jobbe med video for presentasjon

Skrive rapport

Report Week 19

10/05/2020

Bachelorgruppe

Håkon Bjerkgaard Waldum

Ruben Natvik

Ruben Svedal Jørundland

Vebjørn Rimstad Wille

Prosjekt idé

Lage et system for autonom docking av skip vha primært Lidar. Det skal lages en fysisk prototype basert på design av en semi-submersible platform. Det vil bli benyttet en rimelig IMU for estimering av pitch, roll og yaw til å hjelpe med reguleringen av systemet. Prototypen vil styres med 4 motorer plassert i hjørnene på platformen, montert 45 grader på endene.

Prosjektet vil basere seg på at et system på kaia sender et kart over layouten på kaia, med informasjon om hvor fartøyet skal legge til kai.

Det vil også opprettes en GUI for overvåking av systemet som også vil fungere som systemet på kaia.

Forrige uke status

Software

Software har blitt finjustert slik at plattformen nå fungerer så optimalt som vi klarer å få den til å gjøre.

Har fått byttet til laser scan matcher feedback for kalman filter. Fungerer mye bedre enn tidligere feedback system.

Hardware

Har kjøpt ny ruter. GUI fungerer nå som planlagt.

Generelt

Vi har fått kjørt mesteparten av planlagte systemtester. Dette inkluderer path following tester, go to dock tester, drift test, open vs closed loop feedback tester samt systemtester i større omgivelser.

Vi har også utført uoffisielle tester for kollisjonsunnvikelse, med dårlige resultater for TEB Local Planner systemet. Det ble derfor også utført kollisjonsunnvikelse tester med andre Local Planner system.

Oppdagede problemer

Local Planner systemene som har blitt testet fungerer dårlig for kollisjonsunnvikelse. Trolig har dette noe med at de er designet for raskere roboter enn vår platform da de har begrensinger på hvor langt frem i tid de kan simulere.

Nåværende status

Software

Ingen endringer

Hardware

Ingen endringer

Generelt

Har utført tester for kollisjonsunnvikelse.

Jobber med rapport.

Jobber med videopresentasjon

Oppdagede problemer

Neste Uke

Jobbe med rapport

Jobbe med videopresentasjon

Appendix E

STL Files

In this appendix pictures are shown of all 3D-models we have created and used for the construction of the prototype. The STL-files themselves can be found in the .zip-file included with the thesis.

Lidar Bracket

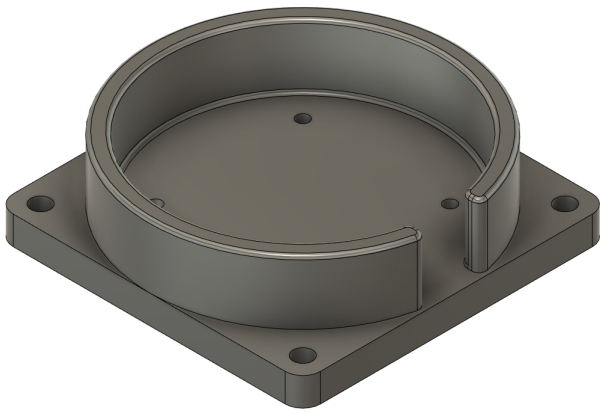


Figure E.1: Bracket created for the RPLidar A3

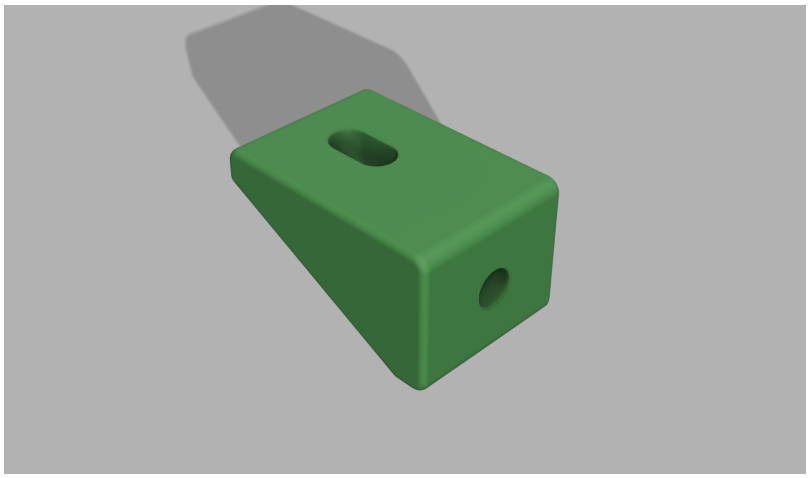


Figure E.2: Lower plate corner bracket

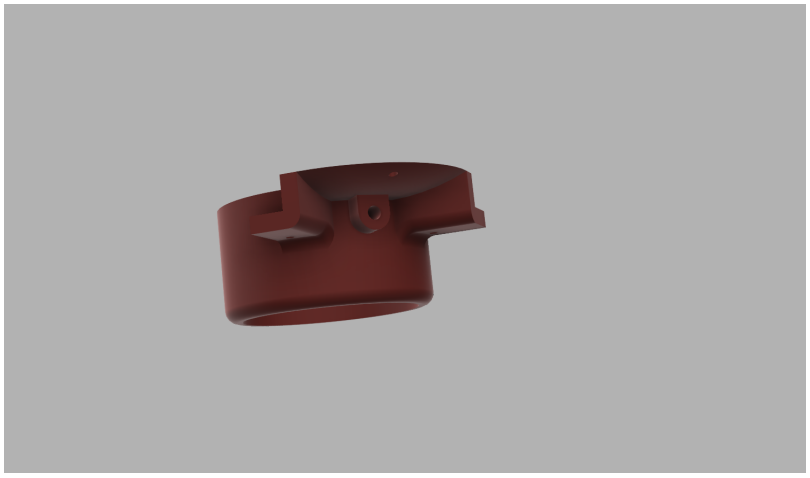


Figure E.3: Upper plate corner construction

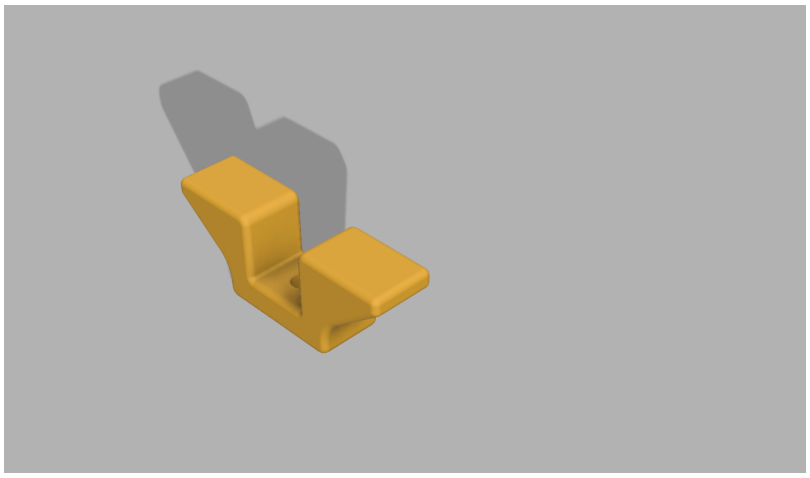


Figure E.4: Lower plate support bracket

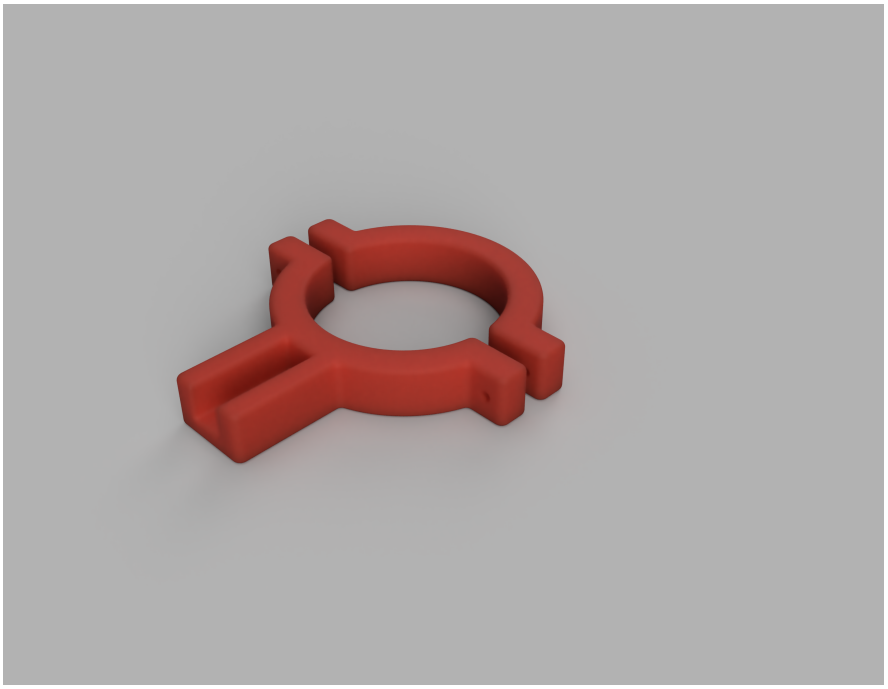


Figure E.5: Lower plate link/lock

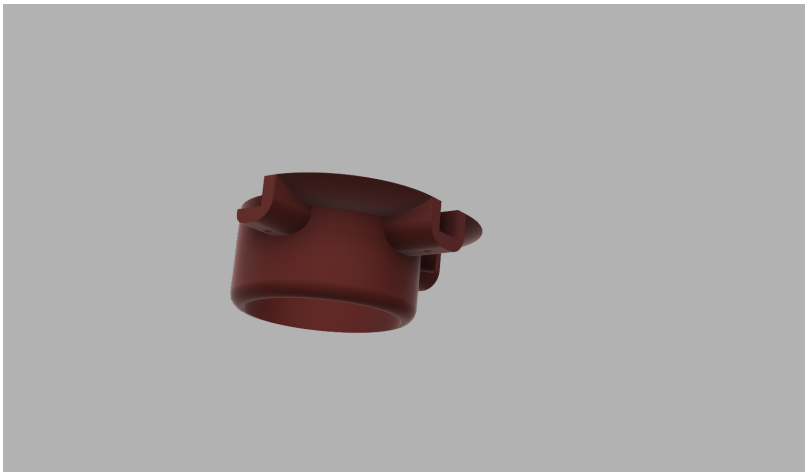


Figure E.6: Upper plate middle construction

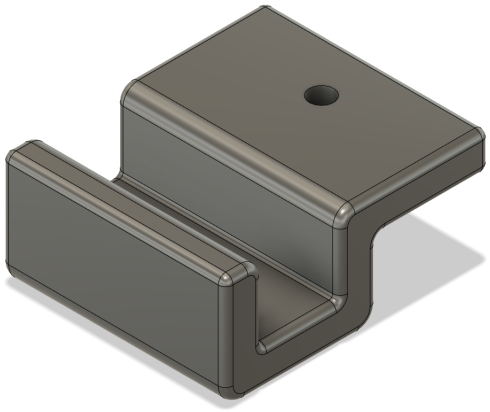


Figure E.7: Lower plate support bracket

Appendix F

Python Code & Config-Files

For the code we've decided to include all code we've made ourselves in this appendix. For the full code with all pre-made packages it can be found in the zip-file included with the project.

All code is present both on the Raspberry as well as the PC, but the split of the code made under is how things are actually run.

F.1 PC-Part of the Python Code for the Prototype

Kalman-filter

```
#!/usr/bin/env python
import csv
from filterpy.kalman import KalmanFilter
import rospy
import tf
from nav_msgs.msg import Odometry
from geometry_msgs.msg import PoseStamped, Twist,
    PoseWithCovarianceStamped
from math import cos, sin
import numpy as np
```

```
class Jallman:
    def __init__(self):

        with open('/home/ruben/luretriks-dev/src/autodock_regulation/
            config/force_curve.csv') as f:
            reader = csv.reader(f)
            force_curve_list = list(reader)
            force_curve_list = [[float(x), float(y)] for x, y in
                force_curve_list]
            self.force_curve = {}
            for x in force_curve_list:
                self.force_curve[x[0]] = x[1]

        with open('/home/ruben/luretriks-dev/src/autodock_regulation/
            config/physical_params.csv') as f:
            reader = csv.reader(f)
            self.physical_params = list(reader)
            self.physical_params = [float(x) for x in self.physical_params
                [0]]

        with open('/home/ruben/luretriks-dev/src/autodock_regulation/
            config/system.csv') as f:
            reader = csv.reader(f)
            system = list(reader)
            system = [[float(y) for y in x] for x in system]
            self.A = np.array([system[0], system[1], system[2]])
            self.B = np.array([system[3], system[4], system[5]])
            self.C = np.array([system[6], system[7], system[8]])
```

```
self.pub = rospy.Publisher("kalman", Twist, queue_size=10)
self.odom_pub = rospy.Publisher("odom", Odometry, queue_size=10)

self.last_pose = [[0, 0, 0], [0, 0, 0]]
self.vel_samples = [[0, 0, 0], [0, 0, 0], [0, 0, 0], [0, 0, 0],
   [0, 0, 0]]
self.vel_i = 0
self.last_ang = 0
self.last_pos_m = np.array([[0], [0], [1]])
self.lastime = rospy.Time.now()
self.first = True

self.last_vel = [[0, 0, 0], [0, 0, 0], [0, 0, 0], [0, 0, 0], [0,
   0, 0]]
self.last_poses = [[[0, 0, 0], [0, 0, 0, 0]], [[0, 0, 0], [0, 0,
   0, 0]], [[0, 0, 0], [0, 0, 0, 0]],
                   [[0, 0, 0], [0, 0, 0, 0]], [[0, 0, 0], [0, 0,
                      0, 0]]]

self.u = np.array([[0], [0], [0]])
self.filter = KalmanFilter(dim_x=3, dim_z=3, dim_u=3)
self.filter.x = np.array([[0], [0], [0]])
self.filter.F = self.A
self.filter.H = self.C
self.filter.P *= 1.
self.filter.R = np.array([[1, 0, 0], [0, 1, 0], [0, 0, 1]]) * 100
self.Q = np.array([[1, 0, 0], [0, 1, 0], [0, 0, 1]]) * 20.
self.filter.Q = self.Q
```

```
def handel_pose(self, msg):
    """
    Takes in a pose message and transforms it in to body coordinates
       then adds that to a global list
    :param msg: geometry_msgs/PoseStamped
    :return: None
    """
    position = msg.pose.position
    orientation = msg.pose.orientation
    _, _, ang = tf.transformations.euler_from_quaternion(
        (orientation.x, orientation.y, orientation.z, orientation.w))
    ang2 = ang
    ang3 = self.last_ang
    pos = np.array([[position.x], [position.y], [1]])
    rot_m0 = np.array([[cos(-self.last_ang), -sin(-self.last_ang), 0],
                       [sin(-self.last_ang), cos(-self.last_ang), 0],
                       [0, 0, 1]])
    trans_b_to_w = (-1 * rot_m0.dot(self.last_pos_m)).tolist()
    rot_m = np.array([[cos(-self.last_ang), -sin(-self.last_ang),
       trans_b_to_w[0][0]],
                      [sin(-self.last_ang), cos(-self.last_ang),
                         trans_b_to_w[1][0]],
                      [0, 0, 1]])
    new_pos = rot_m.dot(pos)
    dt = (msg.header.stamp - self.lastime).to_sec()
    ca = (ang2 - ang3)
    if abs(ca) > 3.14:
        if ang < 0:
            ang2 += 2 * np.pi
        else:
```

```
                ang3 += 2 * np.pi
        vr = (ang2 - ang3) / dt
        v = new_pos / dt
        v[2] = vr
        self.last_ang = ang
        self.last_pos_m = pos
        self.lastime = msg.header.stamp
        if not self.first:
            self.vel_samples.append(v)
            self.vel_samples.pop(0)
            self.vel_i += 1
        else:
            self.first = False

        position = [msg.pose.position.x, msg.pose.position.y, msg.pose.
           position.z]
        orientation = tf.transformations.euler_from_quaternion(
            (msg.pose.orientation.x, msg.pose.orientation.y, msg.pose.
               orientation.z, msg.pose.orientation.w))
        self.last_pose = [position, orientation]

    def constrain(self, value):
        """
        Constrain a value to +- 120
        :param value: number
        :return: constrained number
        """
        if value > 120:
            value = 120
        elif value < -120:
```

```
            value = -120
        return value


    def handel_force(self, msg):
        """
        Takes in a force vector and updates the forceparameter of the
            kalman filter
        :param msg: geometry_msgs/Twist
        :return: None
        """
        f = msg.linear
        # [forward_motors, rigt_motor, left_motor]
        force = [self.constrain(f.x), self.constrain(f.y), self.constrain(
            f.z)]
        forward = self.force_curve[round(force[0])]
        right = self.force_curve[round(force[1])]
        left = self.force_curve[round(force[2])]
        r = self.physical_params[2]
        fi = self.physical_params[3]
        x_force = forward * 2
        y_force = right - left
        n_force = r * (-right * cos(fi) - left * cos(fi))
        self.u = np.array([[x_force], [y_force], [n_force]])


    def publish(self, vel):
        """
        formats and publishes a nav_msgs/Odometry
        :param vel: velocity of the robot in body coordinates
        :return: None
        """
```

```
        odom = Odometry()
        odom.header.stamp = rospy.Time.now()
        odom.header.frame_id = "map"
        odom.child_frame_id = "base_link"

        odom.twist.twist.linear.x = vel[0]
        odom.twist.twist.linear.y = vel[1]
        odom.twist.twist.angular.z = vel[2]

        self.odom_pub.publish(odom)

    def run(self):
        """
        Runs the main loop of the program
        :return: None
        """
        rate = rospy.Rate(20)
        while not rospy.is_shutdown():
            rate.sleep()
            if self.vel_i >= 5:
                vel = [0, 0, 0]
                for x, y, r in self.vel_samples:
                    vel[0] += x
                    vel[1] += y
                    vel[2] += r
                vel = np.array(vel).reshape((3, -1)) / len(self.
                    vel_samples)
                self.filter.update(vel)
                self.vel_i = 0
            self.filter.predict(self.u, self.B, self.A, self.Q)
```

```
            t = Twist()
            t.linear.x = self.filter.x[0]
            t.linear.y = self.filter.x[1]
            t.angular.z = self.filter.x[2]
            self.pub.publish(t)
            self.publish([self.filter.x[0], self.filter.x[1], self.filter.
                x[2]])


if __name__ == "__main__":
    rospy.init_node('jallman', anonymous=True)
    j = Jallman()
    pose_topic = rospy.get_param("pose_topic", "slam_out_pose")
    rospy.Subscriber(pose_topic, PoseStamped, j.handel_pose)
    rospy.Subscriber("roboclaw", Twist, j.handel_force)
    j.run()
```

PID-Regulator

```
#!/usr/bin/env python

import rospy
from geometry_msgs.msg import Twist


class PID:

    def __init__(self, p, i, d, rate=20., out_lim=None, e_sum_lim=None):
        """
        :param p: P gain
```

```
        :param i: I gain
        :param d: D gain
        :param rate: Rate in seconds
        :param out_lim: maximum output
        :param e_sum_lim: maximum error sum value
        """
        if e_sum_lim is None:
            e_sum_lim = [-10., 10.]
        if out_lim is None:
            out_lim = [-50., 50.]
        self.out_lim = out_lim
        self.p = float(p)
        self.i = float(i)
        self.d = float(d)
        self.e0 = 0.
        self.e_sum = 0.
        self.t0 = 0.
        self.rate = rate
        self.e_sum_lim = e_sum_lim

    def pass_value(self, r, m, dt=None):
        """
        calculates the output from the controller
        :param r: setpoint
        :param m: measurement
        :param dt: time pased in sec
        :return: PID output
        """
        if dt is None:
            dt = 1. / float(self.rate)
```

```
        e = r - m
        de = e - self.e0
        self.e_sum += e
        self.e_sum = self._constrain(self.e_sum, self.e_sum_lim)
        self.e0 = e

        p = self.p * e
        i = self.i * self.e_sum * dt
        d = self.d * de / dt
        pid = self._constrain(p + i + d, self.out_lim)

        return pid

    def _constrain(self, v, c):
        """
        constrains the value.
        :param v: value to constrain
        :param c: constrain limit
        :return: constrained value
        """
        v = max(v, c[0])
        v = min(v, c[1])
        return float(v)


class Regulator:

    def __init__(self, pid_forward, pid_side, pid_turn, publisher_topic='
        roboclaw'):
```

```
        self.pid_forward = pid_forward
        self.pid_side = pid_side
        self.pid_turn = pid_turn
        self.publisher = rospy.Publisher(publisher_topic, Twist,
            queue_size=10)
        self.r = {'vx': 0, 'vy': 0, 'vr': 0}
        self.m = {'vx': 0, 'vy': 0, 'vr': 0}

    def handle_vel(self, msg):
        """
        saves the latest velocity message
        :param msg: geometry_msgs/Twist
        :return: None
        """
        self.r['vx'] = msg.linear.x
        self.r['vy'] = msg.linear.y
        self.r['vr'] = msg.angular.z

    def handle_measurement(self, msg):
        """
        saves the latest measurements from the motors
        :param msg: geometry_msgs/Twist
        :return: None
        """
        self.m['vx'] = msg.linear.x
        self.m['vy'] = msg.linear.y
        self.m['vr'] = msg.angular.z

    def constrain(self, value):
        """
```

```
        constrains the number to +- 100.
        :param value: number
        :return: constraind number
        """
        if value > 100:
            value = 100
        elif value < -100:
            value = -100
        return value

    def comp_for_bacwords(self, speed):
        """
        compensate for motors not being as efficient in reverse direction.
        :param speed: motor speed
        :return: speed * 1.3 if negaive
        """
        if speed < 0:
            return speed * 1.3
        else:
            return speed

    def run(self, dt=None):
        """
        runs the regulator
        :param dt: time pased
        :return: None
        """
        forward = self.pid_forward.pass_value(self.r['vx'], self.m['vx'],
            dt=dt)
        side = self.pid_side.pass_value(self.r['vy'], self.m['vy'], dt=dt)
```

```
        turn = self.pid_turn.pass_value(self.r['vr'], self.m['vr'], dt=dt)

        r, l = 0, 0
        r -= turn
        l -= turn
        r += side
        l -= side

        forward = self.constrain(forward)
        r = self.constrain(r)
        l = self.constrain(l)

        msg = Twist()
        msg.linear.x = self.comp_for_bacwords(forward)
        msg.linear.y = self.comp_for_bacwords(r)
        msg.linear.z = self.comp_for_bacwords(l)

        self.publisher.publish(msg)


def main():
    r = 20.  # frequency of rospy.Rate
    rate = rospy.Rate(int(r))
    pid_forward = PID(500., 500., 5., rate=r)
    pid_side = PID(500., 400., 5., rate=r)
    pid_turn = PID(500., 400., 5., rate=r)
    regulator = Regulator(pid_forward, pid_side, pid_turn)
    rospy.Subscriber('pid', Twist, regulator.handle_vel)
    rospy.Subscriber('kalman', Twist, regulator.handle_measurement)
```

```
    while not rospy.is_shutdown():
        rate.sleep()
        regulator.run()


if __name__ == '__main__':
    rospy.init_node('regulator', anonymous=True)
    main()
```

Logger

```
#!/usr/bin/env python
import datetime

import rospy
from sensor_msgs.msg import Imu, MagneticField
from std_msgs.msg import String
from geometry_msgs.msg import Pose2D, PoseWithCovarianceStamped, Twist
from nav_msgs.msg import Odometry, Path
import json

file_name = "/home/ruben/luretriks-dev/logs/log_{}.txt".format(str(
    datetime.datetime.now()))

saving = False

data_defalt = {"imu/data": [], "imu/raw": [], "mag": [], "Pose2D": [], "
    motors": [], "slam_pose": [], "kalman": [],
                "cmd_vel": [], "motor_output": [], "odom": [], "amcl_pose":
                    [], "globla_path": []}
```

```
data = dict(data_defalt)



def save():
    """
    saves logged data to disk
    :return:
    """
    print "saving"
    global saving, data, data_defalt
    saving = True
    data_to_save = data.copy()
    data = {"imu/data": [], "imu/raw": [], "mag": [], "Pose2D": [], "
       motors": [], "slam_pose": [], "kalman": [],
            "cmd_vel": [], "motor_output": [], "odom": [], "amcl_pose":
               [], "globla_path": []}
    saving = False
    with open(file_name, "a") as f:
        f.write(json.dumps(data_to_save) + "\n")



def format_time(stamp):
    """
    formats time to seconds
    :param stamp: rospy.time
    :return: time i seconds
    """
    return stamp.secs + stamp.nsecs * (10 ** -9)
```

```
def imuCallback(imuData):
    """
    formats and saves IMU data from the comp-filter to memory.
    :param imuData: sensor_msgs/Imu
    :return: None
    """
    global data, saving
    if not saving:
        collected_data = {}
        stamp = imuData.header.stamp
        collected_data["time"] = format_time(stamp)

        orientation = imuData.orientation
        collected_data["orientation"] = [orientation.x, orientation.y,
            orientation.z, orientation.w]

        angular_velocity = imuData.angular_velocity
        collected_data["angular_velocity"] = [angular_velocity.x,
            angular_velocity.y, angular_velocity.z]

        linear_acceleration = imuData.linear_acceleration
        collected_data["linear_acceleration"] = [linear_acceleration.x,
            linear_acceleration.y, linear_acceleration.z]

        data["imu/data"].append(collected_data)


def scan_match_calback(msg):
    """
    formats and saves position from scan matcher to memory
```

```
    :param msg: geometry_msgs/Pose_2d
    :return: None
    """
    global data, saving
    if not saving:
        collected_data = {}
        collected_data["pose"] = [msg.x, msg.y, msg.theta]
        stamp = rospy.Time.now()
        collected_data["time"] = format_time(stamp)

        data["Pose2D"].append(collected_data)


def handle_slam_pose(msg):
    """
    formats and saves pose from slam to memory.
    :param msg: geometry_msgs/PoseStamped
    :return: None
    """
    global data, saving
    if not saving:
        collected_data = {}
        stamp = msg.header.stamp
        collected_data["time"] = format_time(stamp)
        pose = msg.pose
        collected_data["covariance"] = pose.covariance
        pose = pose.pose
        collected_data["position"] = [pose.position.x, pose.position.y,
            pose.position.z]
        collected_data["orientation"] = [pose.orientation.x, pose.
```

```
            orientation.y, pose.orientation.z, pose.orientation.w]
        data["slam_pose"].append(collected_data)


def handle_sensor_mag(msg):
    """
    formats and saves magnetometer data to memory.
    :param msg: sensor_msgs/MagneticField
    :return: None
    """
    global data, saving
    if not saving:
        collected_data = {}
        stamp = msg.header.stamp
        collected_data["time"] = format_time(stamp)
        magnetic_field = msg.magnetic_field
        collected_data["magnetic_field"] = [magnetic_field.x,
            magnetic_field.y, magnetic_field.z]

        data["mag"].append(collected_data)


def handle_sensor_raw(msg):
    """
    formats and saves raw Imu data to memory
    :param msg: sensor_msgs/IMU
    :return: None
    """
    global data, saving
    if not saving:
```

```
        collected_data = {}
        stamp = msg.header.stamp
        collected_data["time"] = format_time(stamp)

        orientation = msg.orientation
        collected_data["orientation"] = [orientation.x, orientation.y,
            orientation.z, orientation.w]

        angular_velocity = msg.angular_velocity
        collected_data["angular_velocity"] = [angular_velocity.x,
            angular_velocity.y, angular_velocity.z]

        linear_acceleration = msg.linear_acceleration
        collected_data["linear_acceleration"] = [linear_acceleration.x,
            linear_acceleration.y, linear_acceleration.z]

        data["imu/raw"].append(collected_data)


def motor_calback(msg):
    """
    formats and saves motor speed commands to memory
    :param msg: geometry_msgs/Twist
    :return: None
    """
    global data, saving
    if not saving:
        collected_data = {}
        forward_left_motor = msg.linear.x
        forward_right_motor = msg.linear.x
```

```
        right_turning_motor = msg.linear.y
        left_turning_motor = msg.linear.z
        stamp = rospy.Time.now()
        collected_data["time"] = format_time(stamp)

        collected_data["fl"] = forward_left_motor
        collected_data["fr"] = forward_right_motor
        collected_data["rt"] = right_turning_motor
        collected_data["lt"] = left_turning_motor

        data["motors"].append(collected_data)


def handle_kalman(msg):
    """
    formats and saves velocity data to memory
    :param msg: geometry_msgs/Twist
    :return: None
    """
    global data, saving
    if not saving:
        collected_data = {}
        stamp = rospy.Time.now()
        collected_data["time"] = format_time(stamp)
        x = msg.linear.x
        y = msg.linear.y
        n = msg.angular.z
        collected_data["x"] = x
        collected_data["y"] = y
        collected_data["n"] = n
```

```
        data["kalman"].append(collected_data)


def handle_cmd_vel(msg):
    """
    formats and saves desired velocity to memory
    :param msg: geometry_msgs/Twist
    :return: None
    """
    global data, saving
    if not saving:
        collected_data = {}
        stamp = rospy.Time.now()
        collected_data["time"] = format_time(stamp)
        x = msg.linear.x
        y = msg.linear.y
        n = msg.angular.z
        collected_data["vx"] = x
        collected_data["vy"] = y
        collected_data["vr"] = n
        data["cmd_vel"].append(collected_data)


def handle_motor_out(msg):
    """
      formats and saves motor speed commands to memory
      :param msg: geometry_msgs/Twist
      :return: None
      """
    global data, saving
```

```
    if not saving:
        collected_data = {}
        stamp = rospy.Time.now()
        collected_data["time"] = format_time(stamp)
        x = msg.linear.x
        y = msg.linear.y
        n = msg.angular.z
        collected_data["vx"] = x
        collected_data["vy"] = y
        collected_data["vr"] = n
        data["motor_output"].append(collected_data)


def handle_odom(msg):
    """
    formats and saves odometry data to memory
    :param msg: nav_msgs/Odometry
    :return: None
    """
    global data, saving
    if not saving:
        collected_data = {}
        stamp = msg.header.stamp
        collected_data["time"] = format_time(stamp)
        x = msg.twist.twist.linear.x
        y = msg.twist.twist.linear.y
        n = msg.twist.twist.angular.z
        collected_data["vx"] = x
        collected_data["vy"] = y
        collected_data["vr"] = n
```

```
        pose = msg.pose.pose
        collected_data["position"] = [pose.position.x, pose.position.y,
            pose.position.z]
        collected_data["orientation"] = [pose.orientation.x, pose.
            orientation.y, pose.orientation.z, pose.orientation.w]
        data["odom"].append(collected_data)


def handle_amcl(msg):
    """
    formats and saves the pose from AMCL to memory
    :param msg: geometry_msgs/PoseWitCovarianceStamped
    :return: None
    """
    global data, saving
    if not saving:
        collected_data = {}
        stamp = msg.header.stamp
        collected_data["time"] = format_time(stamp)
        pose = msg.pose
        collected_data["covariance"] = pose.covariance
        pose = pose.pose
        collected_data["position"] = [pose.position.x, pose.position.y,
            pose.position.z]
        collected_data["orientation"] = [pose.orientation.x, pose.
            orientation.y, pose.orientation.z, pose.orientation.w]
        data["amcl_pose"].append(collected_data)
```

```
def handle_path(msg):
    """
    formats and saves the global path to memory
    :param msg: geometry_msgs/Path
    :return: None
    """
    global data, saving
    if not saving:
        collected_data = {}
        stamp = msg.header.stamp
        collected_data["time"] = format_time(stamp)
        path = []
        for p in msg.poses:
            pos = p.pose.position
            ori = p.pose.orientation
            d = {"position": [pos.x, pos.y, pos.z], "orientation": [ori.x,
                ori.y, ori.z, ori.w]}
            path.append(d)
        collected_data["path"] = path
        data["globla_path"].append(collected_data)


def main():
    global last_time, data
    with open(file_name, "w+") as f:
        f.write("")
    rospy.init_node("logger", anonymous=True)
    rospy.Subscriber("imu/data", Imu, imuCallback)
    rospy.Subscriber("pose2D", Pose2D, scan_match_calback)
    rospy.Subscriber("roboclaw", Twist, motor_calback)
```

```
    rospy.Subscriber("poseupdate", PoseWithCovarianceStamped,
        handle_slam_pose)
    rospy.Subscriber("kalman", Twist, handle_kalman)
    rospy.Subscriber("cmd_vel", Twist, handle_cmd_vel)
    rospy.Subscriber("motor_output", Twist, handle_motor_out)
    rospy.Subscriber("odom", Odometry, handle_odom)
    rospy.Subscriber("amcl_pose", PoseWithCovarianceStamped, handle_amcl)
    rospy.Subscriber("move_base/TebLocalPlannerROS/global_plan", Path,
        handle_path)
    rate = rospy.Rate(0.2)
    while not rospy.is_shutdown():
        save()
        rate.sleep()
    save()


if __name__ == "__main__":
    main()
```

GUI Communication

```
#!/usr/bin/env python
import json
import socket

import rospy
import numpy as np
import cv2
from nav_msgs.msg import OccupancyGrid
import tf2_ros
```

```
import copy
import base64


class MapToGui:
    def __init__(self):
        self.rate = rospy.Rate(5)
        self.ready = False

        self.sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        self.sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
        serverAddress = ('192.168.0.103', 5007)
        self.sock.bind(serverAddress)
        self.sock.setblocking(False)
        self.resipients = []

        self.map = None
        self.map_gray = None
        self.map_meta_data = None

        self.tfBuffer = tf2_ros.Buffer()
        self.listener = tf2_ros.TransformListener(self.tfBuffer)

    def crope_img(self, im, meta_data, scale=2.0, gray_copy=None):
        """
        recursively crops the image so only gray pixels are removed.
        will stop trying to crop the image if white or gray pixels are
            getting lost, updates the map metadata.
        :param im: Image
        :param meta_data: meta data for the image
```

```
    :param scale: how much each pass should try to remove
    :param gray_copy: if cropping a picture in color put in gray scale
        image here to reduce computational time
    :return: cropped image, updated meta data
    """
    meta_data = copy.deepcopy(meta_data)
    if not gray_copy is None:
        crop_im = gray_copy
    else:
        crop_im = im
    center_h = int(meta_data.height / 2)
    center_w = int(meta_data.width / 2)
    dim_h = int(meta_data.height / scale)
    dim_w = int(meta_data.width / scale)
    crop = True
    for x in range(dim_h):
        if not crop_im[int(center_h - dim_h / scale) + x, int(center_w
            - dim_w / scale)] == 155.0:
            crop = False
            break
        if not crop_im[int(center_h - dim_h / scale) + x, int(center_w
            + dim_w / scale)] == 155.0:
            crop = False
            break
    if crop:
        for x in range(dim_w):
            if not crop_im[int(center_h - dim_h / scale), int(center_w
                - dim_w / scale) + x] == 155.0:
                crop = False
                break
```

```
                if not crop_im[int(center_h + dim_h / scale), int(center_w
                    - dim_w / scale) + x] == 155.0:
                    crop = False
                    break

        if crop:
            im = im[int(center_h - dim_h / scale):int(center_h + dim_h /
                scale),
                  int(center_w - dim_w / scale):int(center_w + dim_w /
                      scale)]
            meta_data.height = meta_data.height / scale
            meta_data.width = meta_data.width / scale
            meta_data.origin.position.x = meta_data.origin.position.x /
                scale
            meta_data.origin.position.y = meta_data.origin.position.y /
                scale
            if not gray_copy is None:
                gray_copy = gray_copy[int(center_h - dim_h / scale):int(
                    center_h + dim_h / scale),
                                      int(center_w - dim_w / scale):int(center_w +
                                          dim_w / scale)]
            im, meta_data = self.crope_img(im, meta_data, scale, gray_copy
                )

        return im, meta_data

    def recolore_img(self, im):
        """
        recolors the image to the right gray scale 0-255
        :param im: Image
```

```
        :return: Image in right scale
        """
        im = np.where(im == 100, 255, im)
        im = np.where(im == -1, 100, im)
        return 255 - im

    def handle_o_grid(self, msg):
        """
        recieve the map and save it to memory
        :param msg: nav_msgs/OccupancyGrid
        :return: None
        """
        im = np.array(msg.data)
        im = np.reshape(im, (msg.info.width, msg.info.height))
        im = self.recolore_img(im)
        im = im.astype(np.float32)
        im, msg.info = self.crope_img(im, msg.info)
        im_c = cv2.cvtColor(im, cv2.COLOR_GRAY2RGB)

        self.ready = False
        self.map = im_c
        self.map_gray = im
        self.map_meta_data = msg.info
        self.ready = True

    def calc_robot_pos(self, transform, map_meta_data):
        """
        finds the pixel on the map where the robot is located
        :param transform: position of robot in map coordinate frame
        :param map_meta_data: meta data of the map
```

```
        :return: Position of robot
        """
        x_offset = int(0 - map_meta_data.origin.position.x / map_meta_data
            .resolution)
        y_offset = int(0 - map_meta_data.origin.position.y / map_meta_data
            .resolution)
        t = transform.transform.translation
        return [int(t.x / map_meta_data.resolution + x_offset), int(t.y /
            map_meta_data.resolution + y_offset),
                int(t.z)]

    def send(self, im):
        """
        send a udp message wit an image of the map containing the robot
        :param im: image with robot drawn on
        :return: None
        """
        nco = [int(cv2.IMWRITE_JPEG_QUALITY), 70]
        im = cv2.imencode(".jpg", im, nco)[1]
        im = base64.b64encode(im)
        for r in self.resipients:
            self.sock.sendto(im, r)

    def main(self):
        """
        main loop of the program
        :return:
        """
        while not rospy.is_shutdown():
            try:
```

```
                message, address = self.sock.recvfrom(1024)
                if address not in self.resipients:
                    self.resipients.append(address)
                    print("map␣to␣gui␣has␣Conection␣from:␣{},␣message:␣{}"
                        .format(address, message))
            except socket.error as e:
                pass

            if self.ready:
                self.rate.sleep()
                try:
                    base_link_trans = self.tfBuffer.lookup_transform("map"
                        , 'base_link', rospy.Time())
                except (tf2_ros.LookupException, tf2_ros.
                    ConnectivityException, tf2_ros.ExtrapolationException)
                    as e:
                    rospy.logwarn(e)
                    continue
                pose = self.calc_robot_pos(base_link_trans, self.
                    map_meta_data)
                im = cv2.circle(self.map.copy(), (pose[0], pose[1]), 5,
                    (0, 255, 0))
                #im, _ = self.crope_img(im, self.map_meta_data, 1.5, self.
                    map_gray)
                im = im[850:512+1024,650:512+700,:]
                im = cv2.resize(im, (800, 600))
                self.send(im)


if __name__ == "__main__":
```

```
    rospy.init_node('test_yplp', anonymous=True)
    m = MapToGui()
    rospy.Subscriber('map', OccupancyGrid, m.handle_o_grid)
    m.main()
    # rospy.spin()
```

```
#!/usr/bin/env python

import socket
import json
import rospy
import thread
from std_msgs.msg import String

"""
This is a script that is incomplete, and should be finished to actually
    enable communication from the GUI to the program
to change driving modes etc.
"""

class TcpServer:

    def __init__(self, ip, port):
        self.addr = (ip, port)
        self.shutdown = False
        self.publisher = rospy.Publisher('ctrl_mode', String, queue_size
            =10)

    def parse_msg(self, msg):
```

```
        """
        {topic: system_cmd, data: {recipient: ..., parameter_name: ....,
            value: ...}}
        """
        data = msg.decode('utf-8')
        message = String.data = data
        self.publisher.publish(message)

    def main(self):
        rate = rospy.Rate(5)
        with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as soc:
            soc.bind(self.addr)
            while not rospy.is_shutdown():
                soc.listen(5)
                connection, address = soc.accept()
                with connection:
                    connection.settimeout(None)
                    while connection:
                        data = connection.recv(1024)
                        if not data:
                            break
                        self.parse_msg(data)
                        rate.sleep()


if __name__ == '__main__':
    rospy.init_node('tcp', anonymous=True)
    t = TcpServer('192.168.0.103', 5015)
    t.main()
```

```
#!/usr/bin/env python
import socket
import json
import rospy
from sensor_msgs.msg import LaserScan
from geometry_msgs.msg import Twist
import random
import numpy as np

hostIp = rospy.get_param("ip", {"ip": socket.gethostname()})
hostIp = hostIp["ip"]

port = rospy.get_param("port", {"port": "5005"})
port = int(port["port"])
print("server started on host: {}, port: {}".format(hostIp, port))

s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
s.setblocking(False)
serverAddress = ('192.168.0.103', 5005)
s.bind(serverAddress)
resipients = []

ranges = []
angles = []
redy = True

forward = 0
right = 0
left = 0
```

```
def motor_calback(msg):
    """
    formats and saves the latest motor output message
    :param msg: geometry_msgs/Twist
    :return: None
    """
    global forward, right, left
    forward = int(msg.linear.x)
    right = int(msg.linear.y)
    left = int(msg.linear.z)


def handel_laser(data):
    """
    formats and saves the latest laserScan message
    :param data: sensor_msgs/LaseScan
    :return: None
    """
    global ranges, angles, redy, forward, right, left
    redy = False
    ranges = data.ranges
    aMin = data.angle_min
    aMax = data.angle_max
    increment = data.angle_increment
    angles = [x for x in np.arange(aMin, aMax, increment)]
    redy = True
```

```
# print(ranges_json)


def main():
    """
    publishes laser scan and motor state with a rate of 20Hz
    :return: None
    """
    global ranges, angles, redy
    rospy.init_node('scan_listener', anonymous=True)
    rospy.Subscriber("scan", LaserScan, handel_laser)
    rospy.Subscriber("roboclaw", Twist, motor_calback)
    rate = rospy.Rate(20)
    while not rospy.is_shutdown():
        try:
            message, address = s.recvfrom(1024)
            if address not in resipients:
                resipients.append(address)
                print("Laser␣pub␣has␣Conection␣from:␣{},␣message:␣{}".
                    format(address, message))
        except socket.error as e:
            pass
        if redy:
            dictionary = {
                "Lidar": {
                    "Ranges": ranges,
                    "Angles": angles
                },
                "Motor": {
                    "Motor1": forward,
                    "Motor2": forward,
```

```
                    "Motor3": right,
                    "Motor4": left
                }
            }
            ranges_json = json.dumps(dictionary)
            for r in resipients:
                s.sendto(ranges_json.encode(), r)

        rate.sleep()


if __name__ == "__main__":
    main()
```

```
#!/usr/bin/env python
import rospy
from geometry_msgs.msg import Twist
import socket
import json
from threading import Thread


class UdpServer:

    def __init__(self, host, port, buffer_size=1024):
        self.addr = (host, port)
        self.buffer_size = buffer_size
        self.available = {'kalman': True, 'roboclaw': True}
        self.data = {
```

```
            'kalman': {'speedX': 0, 'speedY': 0, 'speedTheta': 0},
            'motors': {'forwardLeft': 0, 'forwardRight': 0, 'turnLeft': 0,
                'turnRight': 0}
        }
        self.socket = None
        self.client_list = []
        self.shutdown = False

    def kalman_callback(self, msg):
        """
        formats and saves the latest vessel speed message
        :param msg: geometry_msgs/Twist
        :return: None
        """
        if self.available['kalman']:
            self.available['kalman'] = False
            self.data['kalman']['speedX'] = msg.linear.x
            self.data['kalman']['speedY'] = msg.linear.y
            self.data['kalman']['speedTheta'] = msg.angular.z
            self.available['kalman'] = True

    def roboclaw_callback(self, msg):
        """
        formats and saves the latest motor speed message
        :param msg: geometry_msgs/Twist
        :return: None
        """
        if self.available['roboclaw']:
            self.available['roboclaw'] = False
            self.data['motors']['forwardLeft'] = msg.linear.x
```

```
            self.data['motors']['forwardRight'] = msg.linear.x
            self.data['motors']['turnRight'] = msg.linear.y
            self.data['motors']['turnLeft'] = msg.linear.z
            self.available['roboclaw'] = True

    def set_socket(self, socket):
        self.socket = socket
        self.socket.settimeout(1)

    def accept_client(self):
        """
        registers new address to send to
        :return: None
        """
        while not self.shutdown:
            if self.socket is not None:
                try:
                    msg, client_addr = s.recvfrom(self.buffer_size)
                    if client_addr not in self.client_list:
                        self.client_list.append(client_addr)
                        print("vessel␣pub␣has␣Conection␣from:␣{},␣message:
                            ␣{}".format(client_addr, msg))
                except socket.error as e:
                    pass


if __name__ == '__main__':
    rospy.init_node('vessel_udp_publisher', anonymous=True)
    server = UdpServer('192.168.0.103', 5006)
    welcome_thread = Thread(target=server.accept_client)
```

```
    rate = rospy.Rate(4)
    rospy.Subscriber("roboclaw", Twist, server.roboclaw_callback)
    rospy.Subscriber("kalman", Twist, server.kalman_callback)
    s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    s.bind(server.addr)
    server.set_socket(s)
    welcome_thread.start()
    while not rospy.is_shutdown():
        rate.sleep()

        while not server.available['kalman']:
            pass
        server.available['kalman'] = False
        while not server.available['roboclaw']:
            pass
        server.available['roboclaw'] = False
        data = json.dumps(server.data).encode('utf-8')
        for c in server.client_list:
            s.sendto(data, c)

        server.available['kalman'] = True
        server.available['roboclaw'] = True
    server.shutdown = True
```

Sending Positions

```
#!/usr/bin/env python
import time

import rospy
```

```
import math

import actionlib
from move_base_msgs.msg import MoveBaseAction, MoveBaseGoal
from actionlib_msgs.msg import GoalStatus
from geometry_msgs.msg import Pose, Point, Quaternion,
    PoseWithCovarianceStamped
import tf


class GoToDock:
    def __init__(self):
        self.client = actionlib.SimpleActionClient('move_base',
            MoveBaseAction)
        self.init_pose_publisher = rospy.Publisher('initialpose',
            PoseWithCovarianceStamped, queue_size=10)

        self.client.wait_for_server()

    def sendGoal(self, pos, orientation):
        """
        Sends goal to action server
        :param pos: position [x, y, z]
        :param orientation: orientation in quaternions [x, y, z, w]
        :return: None
        """
        goal = MoveBaseGoal()
        goal.target_pose.header.frame_id = 'map'
        goal.target_pose.header.stamp = rospy.Time.now()
```

```
        goal.target_pose.pose.position.x = pos[0]
        goal.target_pose.pose.position.y = pos[1]
        goal.target_pose.pose.position.z = pos[2]

        goal.target_pose.pose.orientation.x = orientation[0]
        goal.target_pose.pose.orientation.y = orientation[1]
        goal.target_pose.pose.orientation.z = orientation[2]
        goal.target_pose.pose.orientation.w = orientation[3]

        self.client.send_goal(goal, self.done_cb, self.active_cb, self.
            feedback_cb)
        print "goal sent"
        wait = self.client.wait_for_result()

    def active_cb(self):
        """
        Active callback function
        Not currently in use
        :return: None
        """
        print "active"
        # print "status: {}".format(status)

    def feedback_cb(self, feedback):
        """
        Feedback callback function
        Not currently in use
        :param feedback: feedback from action server
        :return: None
        """
```

```
        return
        # print "feedback: {}".format(feedback)


    def done_cb(self, result, yolo):
        """
        Callback function used when goal is reached
        :param result: Result feedback from server
        :param yolo: Not in use
        :return: None
        """
        if result == 0:
            print "not␣prossest"
        elif result == 1:
            print "beeing␣prosset"
        elif result == 2:
            print "cancle␣request"
        elif result == 3:
            print "sucside"
        elif result == 4:
            print "aborted"
        elif result == 5:
            print "rejected"
        elif result == 6:
            print "cancle␣2"
        elif result == 7:
            print "canceling"
        elif result == 8:
            print "canceld"
        elif result == 9:
            print "gole␣lost"
```

```
        print "yolo: {}".format(yolo)

    def send_init_pose(self, pos, orientation):
        """
        Publishes initial pose to AMCL via topic initialpose
        :param pos: position in [x, y, z]
        :param orientation: orientation in quaternions [x, y, z, w]
        :return: None
        """
        pose = PoseWithCovarianceStamped()
        pose.header.frame_id = 'map'
        pose.header.stamp = rospy.Time.now()

        pose.pose.pose.position.x = pos[0]
        pose.pose.pose.position.y = pos[1]
        pose.pose.pose.position.z = pos[2]

        pose.pose.pose.orientation.x = orientation[0]
        pose.pose.pose.orientation.y = orientation[1]
        pose.pose.pose.orientation.z = orientation[2]
        pose.pose.pose.orientation.w = orientation[3]

        self.init_pose_publisher.publish(pose)


if __name__ == "__main__":
    rospy.init_node('go_to_dock', anonymous=False)
    gtd = GoToDock()
    time.sleep(1)
```

```
    # gtd.send_init_pose((-1.56, -0.45, 0), (0, 0, -0.71, 0.69))
    # time.sleep(1)

    q = tf.transformations.quaternion_from_euler(0, 0, 0)

    gtd.sendGoal((-2.6, 10.4, 0), (0, 0, -0.69, 0.72))

    gtd.sendGoal((-1.35, 10.4, 0), (0, 0, -0.69, 0.72))
```

Transform Odom to Base

```
#!/usr/bin/env python
import time

import rospy
import tf
from sensor_msgs.msg import Imu
from std_msgs.msg import String
import numpy as np
import json


class imuTransformer():
    """
    In dire need of rework. Many different functions were added to this
        node,
    that are not in use.
    """

    def __init__(self):
```

```
        imuTopic = rospy.get_param("imu_topic")
        name = rospy.get_param('name')
        config_file = rospy.get_param('config_file_odom')
        rospy.init_node(name, anonymous=True)
        rospy.Subscriber(imuTopic, Imu, self.imuCallback)
        self.calibrate_reply_publisher = rospy.Publisher('calibrate/reply'
            , String, queue_size=10)
        rospy.Subscriber("calibrate", String, self.handle_config)
        self.initial_quaternion = []
        self.first = True
        with open(config_file, 'r') as cf:
            config_data = json.load(cf)
        self.g_offset = config_data['g_offset']

        self.ready = False
        self.really_ready = False
        self.acc = [0, 0, 0]
        self.vel = [0, 0, 0]
        self.distance = [0, 0, 0]
        self.t0 = 0
        self.timeStart = time.time()
        rospy.Subscriber("imu/sensor_raw", Imu, self.sensor_callback)

    def imuCallback(self, imuData):
        """
        Saves the latest Imu message to memory.
        publishes all frames needed to run slam
        :param imuData: sensor_msgs/Imu
        :return: None
        """
```

```
        if self.ready:
            orientation = imuData.orientation
            if self.first:
                self.first = False
                self.initial_quaternion = orientation
                time_stamp = imuData.header.stamp
                self.t0 = time_stamp.to_sec()

            else:
                rotation_matrix = self.createRotationMatrix(orientation)
                NED_acceleration = self.remove_gravity_from_rot_matrix(
                    rotation_matrix, self.acc, self.g_offset)
                time_stamp = imuData.header.stamp
                t = time_stamp.to_sec()
                dt = t - self.t0
                self.t0 = t
                self.vel = [a*dt + v0 for a, v0 in zip(NED_acceleration,
                    self.vel)]
                self.distance = [v*dt + s0 for v, s0 in zip(self.vel, self
                    .distance)]
                #print NED_acceleration
                #print time.time() - self.timeStart

            orientation.x = orientation.x - self.initial_quaternion.x
            orientation.y = orientation.y - self.initial_quaternion.y
            orientation.z = orientation.z - self.initial_quaternion.z
            orientation.w = orientation.w - self.initial_quaternion.w

            roll0 = 0
            pitch0 = 0
```

```
            yaw0 = 0
            q0 = tf.transformations.quaternion_from_euler(roll0, pitch0,
                yaw0)
            roll, pitch, yaw = tf.transformations.euler_from_quaternion(
                (orientation.x, orientation.y, orientation.z, orientation.
                    w))
            yaw_q = tf.transformations.quaternion_from_euler(0, 0, yaw)
            q = tf.transformations.quaternion_from_euler(roll, pitch, 0)
            br = tf.TransformBroadcaster()

            br.sendTransform((0, 0, 0), yaw_q, rospy.Time.now(), '
                base_footprint', 'odom')
            br.sendTransform((0, 0, 0.25), q0, rospy.Time.now(), '
                base_stabilized', 'base_footprint')
            br.sendTransform((0, 0, 0), q, rospy.Time.now(), 'base_link',
                'base_stabilized')

            #br.sendTransform((0, 0, 0), q0, rospy.Time.now(),'base_link',
                'base_stabilized')

    def sensor_callback(self, msg):
        """
        Callback function receiving raw IMU-data
        :param msg: IMU-data
        :return: None
        """
        self.acc = [msg.linear_acceleration.x, msg.linear_acceleration.y,
            msg.linear_acceleration.z]
        self.ready = True
```

```
def handle_config(self, msg):
    """
    Not currently in use.
    Was intended to be used to remove gravity from NED-matrix
    :param msg: std_msgs/String
    :return:
    """
    name = rospy.get_name()[1:]
    msg = msg.data
    reset_file_name = rospy.get_param('reset_file_odom')
    config_file_name = rospy.get_param('config_file_odom')
    if msg == "reset":
        with open(reset_file_name, 'r') as rf:
            reset_file_data = json.load(rf)
        with open(config_file_name, 'w') as cf:
            json.dump(reset_file_data, cf)
        self.calibrate_reply_publisher.publish(name + "::ok")
    else:
        sender, calibraton = msg.split("::")
        if sender == name:
            rospy.loginfo(name + "␣config␣received")
            with open(config_file_name, 'r') as cf:
                json_calibration = json.loads(calibraton)
                json_file = json.load(cf)
                for key in json_calibration.keys():
                    json_file[key] = json_calibration[key]
            with open(config_file_name, 'w') as cf:
                json.dump(json_file, cf)
            self.calibrate_reply_publisher.publish(name + "::ok")
    with open(config_file_name, 'r') as cf:
```

```
            config_data = json.load(cf)
            self.g_offset = config_data['g_offset']

    def createRotationMatrix(self, orientation):
        """
        Creates a rotation matrix from quaternions
        :param orientation: orientation in quaternions [x, y, z, w]
        :return: rotation matrix
        """
        # Extract information from orientation
        qx, qy, qz, qw = [orientation.w, orientation.x, orientation.y,
            orientation.z]

        rotation_matrix = np.array([
            [2 * (qx ** 2) - 1 + 2 * (qy ** 2), 2 * qy * qz + 2 * qx * qw,
                2 * qy * qw - 2 * qx * qz],
            [2 * qy * qz - 2 * qx * qw, 2 * (qx ** 2) - 1 + 2 * (qz ** 2),
                2 * qz * qw + 2 * qx * qy],
            [2 * qy * qw + 2 * qx * qz, 2 * qz * qw - 2 * qx * qy, 2 * (qx
                ** 2) - 1 + 2 * (qw ** 2)]
        ]).reshape(3, 3)

        return rotation_matrix

    def remove_gravity_from_rot_matrix(self, rotation_matrix, raw_acc,
        grav_offset):
        """
        Removes gravity from the rotation matrix.
        Not currently in use
        :param rotation_matrix: rotation matrix [3x3]
```

```
        :param raw_acc: raw acceleration data from accelerometer
        :param grav_offset: gravity offset
        :return: NED-matrix compensated for gravity
        """
        # Need inverse of Rot Matrix
        rotation_matrix_inverse = np.linalg.inv(rotation_matrix)

        # Multiply raw data from accel with inverse of rot matrix
        acc_NED = rotation_matrix_inverse.dot(raw_acc).tolist()

        # Remove gravity from North East Down-matrix
        acc_NED_compensated = [acc - offs for acc, offs in zip(acc_NED,
            grav_offset)]

        return acc_NED_compensated

    def run(self):
        """
        Runs program
        :return:
        """
        # while not rospy.is_shutdown():
        #       br = tf.TransformBroadcaster()
        #       br.sendTransform((0, 0, 0), (0, 0, 0, 0), rospy.Time.now()
            , 'base_link', 'Odom')
        rospy.spin()


if __name__ == "__main__":
    time.sleep(5)
```

```
    imutrf = imuTransformer()
    imutrf.run()
```

Joystick to Velocity Commands

```
#!/usr/bin/env python

import rospy
from geometry_msgs.msg import Twist
from sensor_msgs.msg import Joy


class JoyHandler:

    def __init__(self, pub):
        self.pub = pub

    def handel_joy(self, msg):
        """
        converts a sensor_msgs/joy to a geometry_msgs/Twist in the range
            between -0.2 to 0.2
        :param msg: sensor_msgs/joy
        :return: None
        """
        scaler_l = 0.2
        scaler_a = 0.2

        axis = msg.axes
        r_trigger = (axis[2] + 1) / 2
        l_trigger = (axis[5] + 1) / 2
        # linear motion
```

```
        forward = r_trigger* scaler_l
        forward -= l_trigger * scaler_l



        # turning motion
        turn = axis[3] * scaler_a

        # lateral shift
        lat = axis[0] * scaler_l

        t = Twist()
        t.linear.x = forward
        t.linear.y = lat
        t.angular.z = turn

        self.pub.publish(t)



if __name__ == "__main__":
    rospy.init_node("joy_to_vel", anonymous=True)
    pub = rospy.Publisher('joy_vel', Twist, queue_size=10)
    handler = JoyHandler(pub)
    rospy.Subscriber("joy", Joy, handler.handel_joy)
    rospy.spin()
```

Safety Publisher

```
#!/usr/bin/env python
from std_msgs.msg import String
import rospy
```

```
if __name__ == "__main__":
    """
    sends alive messages to the safety-listener node
    """
    rospy.init_node("safety_publisher", anonymous=True)
    publisher = rospy.Publisher('alive', String, queue_size=10)

    rate = rospy.Rate(1)  # Hz
    while not rospy.is_shutdown():
        publisher.publish("alive")
        rate.sleep()
```

F.2 Raspberry Pi-part of the Python Code for the Prototype

IMU-Filter

```
#!/usr/bin/env python

import rospy
import json
from sensor_msgs.msg import Imu, MagneticField
from std_msgs.msg import String


class Filter:

    def __init__(self, b, a=None, k=1):
        """
        Creates a filter object
        :param b: numerator coefficients of the transfer function (coeffs
            of X)
```

```
        :param a: denominator coefficients of the transfer function (
            coeffs of Y)
        :param k: output gain (default 1)
        """
        if not a:
            a = [1]
        self.b = b
        self.a = a
        self.k = k
        self.input = [0] * len(b)
        self.output = [0] * (len(a) - 1)

    def filter_value(self, x_new):
        """
        Passes a single value through the filter
        TODO: Find a better way to do the calculation than to use pop and
            insert for self.a[0]
        :param x_new: the value to be passed through the filter
        :return: the filter's output value
        """
        self.input = self._shift_list(self.input, x_new)
        a0 = self.a.pop(0)
        y_new = a0 * (
                sum([b * x for b, x in zip(self.b, self.input)])
                - sum([a * y for a, y in zip(self.a, self.output)])
        )
        self.a.insert(0, a0)
        if len(self.output) > 0:
            self.output = self._shift_list(self.output, y_new)
        return y_new
```

```
    def filter_list(self, xn):
        """
        Passes a list of values through the filter
        :param xn: the input vector
        :return: the filter output vector
        """
        y = []
        for x in xn:
            y.append(self.filter_value(x))
        return y

    def clear(self):
        """
        Clear the filter's stored input and output list
        :return: None
        """
        self.input = [0] * len(self.input)
        self.output = [0] * len(self.output)

    def _shift_list(self, lst, val):
        """
        Removes the last value in a list an puts in a value in the first
            position
        :param lst: list
        :param val: value
        :return: shifted list
        """
        lst.pop()
        lst.insert(0, val)
```

```
        return lst


class MultiChannelFilter:

    def __init__(self, channels, b, a=None, k=1):
        self.channels = channels
        self.filters = []
        for i in range(self.channels):
            self.filters.append(Filter(b, a=a, k=k))

    def filter_values(self, values):
        """
        Filters values
        :param values: values to filter
        :return: filtered values
        """
        filtered_values = [0] * self.channels
        if len(values) == self.channels:
            for i in range(self.channels):
                filtered_values[i] = self.filters[i].filter_value(values[i
                    ])
        else:
            filtered_values = None
        return filtered_values


class SensorFilter:

    def __init__(self, acc_num, gyro_num, mag_num,
```

```
            acc_offset=None, gyro_offset=None, mag_offset=None,
            acc_cv=None, gyro_cv=None, mag_cv=None):
        if mag_cv is None:
            mag_cv = [0] * 9
        if gyro_cv is None:
            gyro_cv = [0] * 9
        if acc_cv is None:
            acc_cv = [0] * 9
        if mag_offset is None:
            mag_offset = [0, 0, 0]
        if gyro_offset is None:
            gyro_offset = [0, 0, 0]
        if acc_offset is None:
            acc_offset = [0, 0, 0]

        self.mag_cv = mag_cv[0] + mag_cv[1] + mag_cv[2]
        self.gyro_cv = gyro_cv[0] + gyro_cv[1] + gyro_cv[2]
        self.acc_cv = acc_cv[0] + acc_cv[1] + acc_cv[2]
        self.mag_offset = mag_offset
        self.gyro_offset = gyro_offset
        self.acc_offset = acc_offset
        self.acc_filter = MultiChannelFilter(3, acc_num)
        self.gyro_filter = MultiChannelFilter(3, gyro_num)
        self.mag_filter = MultiChannelFilter(3, mag_num)
        self.raw_publisher = rospy.Publisher('imu/data_raw', Imu,
            queue_size=10)
        self.mag_publisher = rospy.Publisher('imu/mag', MagneticField,
            queue_size=10)
        self.calibrate_reply_publisher = rospy.Publisher('calibrate/reply'
            , String, queue_size=10)
```

```
def handle_sensor_raw(self, msg):
    """
    Filters the raw imu data and publishes the filtered data
    :param msg: imu-data
    :return: None
    """
    gyro_raw = [msg.angular_velocity.x, msg.angular_velocity.y, msg.
        angular_velocity.z]
    acc_raw = [msg.linear_acceleration.x, msg.linear_acceleration.y,
        msg.linear_acceleration.z]
    gyro = [g - o for g, o in zip(gyro_raw, self.gyro_offset)]
    acc = [a - o for a, o in zip(acc_raw, self.acc_offset)]
    gyro_data = self.gyro_filter.filter_values(gyro)
    acc_data = self.acc_filter.filter_values(acc)
    new_msg = self.format_raw(acc_data, gyro_data)
    self.raw_publisher.publish(new_msg)

def handle_sensor_mag(self, msg):
    """
    Filters the raw magnetometer data and publishes the filtered data
    :param msg: Magnetometer-data
    :return: None
    """
    mag_raw = [msg.magnetic_field.x, msg.magnetic_field.y, msg.
        magnetic_field.z]
    mag = [m - o for m, o in zip(mag_raw, self.mag_offset)]
    mag_data = self.mag_filter.filter_values(mag)
    new_msg = self.format_mag(mag_data)
    self.mag_publisher.publish(new_msg)
```

```
def handle_config(self, msg):
    """
    Receives and updates the config from the calibrate-script
    :param msg: std_msgs/String
    :return: None
    """
    name = rospy.get_name()[1:]
    msg = msg.data
    config_file_name = rospy.get_param('config_file_filter')
    reset_file_name = rospy.get_param('reset_file_filter')

    if msg == "reset":
        with open(reset_file_name, 'r') as rf:
            reset_file_data = json.load(rf)
        with open(config_file_name, 'w') as cf:
            json.dump(reset_file_data, cf)
        self.calibrate_reply_publisher.publish(name + "::ok")
    else:
        sender, calibraton = msg.split("::")

        if sender == name:
            rospy.loginfo(name + " config received")
            with open(config_file_name, 'r') as cf:
                json_calibration = json.loads(calibraton)
                json_file = json.load(cf)
                for key in json_calibration.keys():
                    json_file[key] = json_calibration[key]
            with open(config_file_name, 'w') as cf:
                json.dump(json_file, cf)
```

```
                self.calibrate_reply_publisher.publish(name + "::ok")
        with open(config_file_name, 'r') as cf:
            config_data = json.load(cf)

            self.gyro_offset = config_data['gyro_offset']
            mag_cv = config_data['cv_mag']
            gyro_cv = config_data['cv_angular']
            acc_cv = config_data['cv_linear']
            self.mag_cv = mag_cv[0] + mag_cv[1] + mag_cv[2]
            self.gyro_cv = gyro_cv[0] + gyro_cv[1] + gyro_cv[2]
            self.acc_cv = acc_cv[0] + acc_cv[1] + acc_cv[2]

    def format_raw(self, acc_data, gyro_data):
        """
        Formats ROS IMU-message
        :param acc_data: accelerometer data
        :param gyro_data: gyroscope data
        :return: formated message
        """
        msg = Imu()
        msg.header.frame_id = 'imu_raw'
        msg.header.stamp = rospy.Time.now()
        msg.angular_velocity.x = gyro_data[0]
        msg.angular_velocity.y = gyro_data[1]
        msg.angular_velocity.z = gyro_data[2]
        msg.angular_velocity_covariance = self.gyro_cv
        msg.linear_acceleration.x = acc_data[0]
        msg.linear_acceleration.y = acc_data[1]
        msg.linear_acceleration.z = acc_data[2]
        msg.linear_acceleration_covariance = self.acc_cv
```

```
        return msg


    def format_mag(self, mag_data):
        """
        Formats magnetometer-message
        :param mag_data: magnetometer data
        :return: formated message
        """
        msg = MagneticField()
        msg.header.frame_id = 'imu_mag'
        msg.header.stamp = rospy.Time.now()
        msg.magnetic_field.x = mag_data[0]
        msg.magnetic_field.y = mag_data[1]
        msg.magnetic_field.z = mag_data[2]
        msg.magnetic_field_covariance = self.mag_cv
        return msg


def main():
    """
    Runs the filter
    :return:
    """
    rospy.init_node('imu_publisher_node', anonymous=True)
    filter_file = rospy.get_param("filter_file")
    calibration_file = rospy.get_param("config_file_filter")
    with open(filter_file) as json_file:
        filter_config = json.load(json_file)
    acc_filter = filter_config['accelerometer']
    gyro_filter = filter_config['gyroscope']
```

```
    mag_filter = filter_config['magnetometer']

    with open(calibration_file) as cal_file:
        calibration_config = json.load(cal_file)
    sensor_filter = SensorFilter(
        acc_filter['num'],
        gyro_filter['num'],
        mag_filter['num'],
        gyro_offset=calibration_config['gyro_offset'],
        acc_cv=calibration_config['cv_linear'],
        gyro_cv=calibration_config['cv_angular'],
        mag_cv=calibration_config['cv_mag']
    )
    rospy.Subscriber('imu/sensor_raw', Imu, sensor_filter.
        handle_sensor_raw)
    rospy.Subscriber('imu/sensor_mag', MagneticField, sensor_filter.
        handle_sensor_mag)
    rospy.Subscriber("calibrate", String, sensor_filter.handle_config)
    rospy.spin()


if __name__ == '__main__':
    main()
```

IMU-Link

```
#!/usr/bin/env python
import time
from multiprocessing import TimeoutError

import rospy
```

```
from sensor_msgs.msg import Imu, MagneticField
import serial
import numpy as np


class ImuSensor:
    """
    A class for communicating with an inertial measurement unit over
        serial
    """

    def __init__(self, serial_port):
        self.serial_port = serial_port
        self.last_time = time.time()
        self.timeout = 5

    def get_imu_data(self, expected_params=19):
        """
        Receives IMU-data from the Arduino and extracts and separates it
        :param expected_params: amount of datapoints
        :return: data from imu1, imu2, imu3, timestamp
        """
        while self.serial_port.in_waiting < 1 and not rospy.is_shutdown():
            pass
        items = []
        while len(items) is not expected_params and not rospy.is_shutdown
            ():
            data = list(bytearray(self.serial_port.readline()))
            string = ''.join(chr(i) for i in data)
            items = string.strip().split(',')
```

```
        raw = [float(s) for s in items]

        imu1 = {"acc": raw[0:3], "gyro": raw[3:6], "mag": raw[6:9]}
        imu2 = {"acc": raw[9:12], "gyro": raw[12:15]}
        imu3 = {"acc": raw[15:18]}
        t = raw[18]
        return imu1, imu2, imu3, t

    def get_imu_data_single(self, expected_params=10):
        """
        Receives IMU-data from the Arduino and extracts it
        :param expected_params: amount of data-points
        :return: accelerometer_data, gyroscope_data, magnetometer_data,
           timestamp
        """
        self.last_time = time.time()
        while self.serial_port.in_waiting < 1 and not rospy.is_shutdown():
            if time.time() - self.last_time > self.timeout:
                raise TimeoutError("Arduino timed out")
        items = []
        while len(items) is not expected_params and not rospy.is_shutdown
           ():
            if time.time() - self.last_time > self.timeout:
                raise TimeoutError("Arduino timed out")

            data = list(bytearray(self.serial_port.readline()))
            string = ''.join(chr(i) for i in data)
            items = string.strip().split(',')
        raw = [float(s) for s in items]
```

```
        acc_data = raw[0:3]
        gyro_data = raw[3:6]
        mag_data = raw[6:9]
        t = raw[9]

        acc_data = [acc_data[1], -acc_data[0], acc_data[2]]
        gyro_data = [gyro_data[1], -gyro_data[0], gyro_data[2]]
        mag_data = [mag_data[1], -mag_data[0], mag_data[2]]
        return acc_data, gyro_data, mag_data, t


def format_raw(acc_data, gyro_data):
    """
    Formats the data for publishing
    :param acc_data: accelerometer_data
    :param gyro_data: gyroscope_data
    :return: formatted message
    """
    msg = Imu()
    msg.header.frame_id = 'sensor_raw'
    msg.header.stamp = rospy.Time.now()
    msg.angular_velocity.x = gyro_data[0]
    msg.angular_velocity.y = gyro_data[1]
    msg.angular_velocity.z = gyro_data[2]
    msg.linear_acceleration.x = acc_data[0]
    msg.linear_acceleration.y = acc_data[1]
    msg.linear_acceleration.z = acc_data[2]
    return msg
```

```
def format_mag(mag_data):
    """
    Formats magnetometer data for publishing
    :param mag_data: magnetometer_data
    :return: formatted message
    """
    msg = MagneticField()
    msg.header.frame_id = 'sensor_mag'
    msg.header.stamp = rospy.Time.now()
    msg.magnetic_field.x = mag_data[0]
    msg.magnetic_field.y = mag_data[1]
    msg.magnetic_field.z = mag_data[2]
    return msg


window_x = []
window_y = []
window_z = []


def running_average_data(imu1, imu2, imu3):
    """
    Takes running average of data from three different IMU's
    :param imu1: acc, gyro, mag
    :param imu2: acc, gyro
    :param imu3: acc
    :return: average_accel, average_gyro, average_mag
    """
    avg_acc = [sum(x) / 3 for x in zip(imu1["acc"], imu2["acc"], imu3["acc
        "])]
```

```
    avg_gyro = [sum(x) / 1 for x in zip(imu1["gyro"])]
    avg_mag = [sum(x) / 1 for x in zip(imu1["mag"])]

    window_x.append(avg_acc[0])
    window_y.append(avg_acc[1])
    window_z.append(avg_acc[2])
    if len(window_x) > 9:
        window_x.pop(0)
        window_y.pop(0)
        window_z.pop(0)

    avg_x = float(sum(window_x)) / float(len(window_x))
    avg_y = float(sum(window_y)) / float(len(window_y))
    avg_z = float(sum(window_z)) / float(len(window_z))

    avg_acc_filt = [avg_x, avg_y, avg_z]

    return avg_acc_filt, avg_gyro, avg_mag


def average_data(imu1, imu2, imu3):
    """
    Takes average of data from three IMU's
    :param imu1: acc, gyro, mag
    :param imu2: acc, gyro
    :param imu3: acc
    :return: averaged acc_data, gyro_data, mag_data
    """
    avg_acc = [sum(x) / 3 for x in zip(imu1["acc"], imu2["acc"], imu3["acc
        "])]
```

```
    avg_gyro = [sum(x) / 1 for x in zip(imu1["gyro"])]
    avg_mag = [sum(x) / 1 for x in zip(imu1["mag"])]

    return avg_acc, avg_gyro, avg_mag


def median_data(imu1, imu2, imu3):
    """
    Takes the median data between three IMU's
    :param imu1: acc, gyro, mag
    :param imu2: acc, gyro
    :param imu3: acc
    :return: median acc_data, gyro_data, mag_data
    """
    med_acc = [np.median([imu1["acc"][0], imu2["acc"][0], imu3["acc"][0]])
        ,
               np.median([imu1["acc"][1], imu2["acc"][1], imu3["acc"][1]])
                   ,
               np.median([imu1["acc"][2], imu2["acc"][2], imu3["acc"][2]])
                   ]
    med_gyro = imu1["gyro"]
    med_mag = imu1["mag"]

    return med_acc, med_gyro, med_mag


def main():
    """
    Runs the program. Boolean in start to say if data from three IMU's or
        just a single IMU
```

```
    :return: None
    """
    tripleImu = False
    singleImu = True
    rospy.init_node('sensor_publisher_node', anonymous=True)
    port = rospy.get_param("serial_port", "/dev/ttyUSB0")
    baud_rate = rospy.get_param("serial_baud_rate", 115200)

    while not rospy.is_shutdown():
        try:
            with serial.Serial(port, baud_rate, timeout=0.5) as
                serial_port:
                sensor = ImuSensor(serial_port)
                raw_publisher = rospy.Publisher('imu/sensor_raw', Imu,
                    queue_size=10)
                mag_publisher = rospy.Publisher('imu/sensor_mag',
                    MagneticField, queue_size=10)
                # rate = rospy.Rate(50)
                while not rospy.is_shutdown():
                    if tripleImu:
                        imu1, imu2, imu3, t = sensor.get_imu_data()
                        acc_data, gyro_data, mag_data =
                            running_average_data(imu1, imu2, imu3)
                        raw_publisher.publish(format_raw(acc_data,
                            gyro_data))
                        mag_publisher.publish(format_mag(mag_data))
                    if singleImu:
                        acc_data, gyro_data, mag_data, t = sensor.
                            get_imu_data_single()
                        raw_publisher.publish(format_raw(acc_data,
```

```
                                gyro_data))
                            mag_publisher.publish(format_mag(mag_data))

        except TimeoutError as e:
            rospy.logerr(e.message)
        except serial.serialutil.SerialException as e:
            rospy.logfatal(e)
            exit(126)


if __name__ == '__main__':
    '''
    Parameters:
    raw_frame_name (default 'raw_imu_frame')
    mag_frame_name (default 'mag_imu_frame')
    serial_port (default '/dev/ttyUSB0')
    serial_baud_rate (default 115200)
    '''
    main()
```

Motor Control

```
#!/usr/bin/env python

import random
import serial
import time
import rospy
from geometry_msgs.msg import Twist
from sensor_msgs.msg import Joy
from std_msgs.msg import String
```

```
import json



class MotorController:
    def __init__(self):
        """
        establishes communication with both roboclaws
        """
        self.rc = Roboclaw("/dev/ttyACM0", 230400)
        self.rc2 = Roboclaw("/dev/ttyACM1", 230400)
        self.addressFB = 0x80
        self.addressLR = 0x81
        self.rc.Open()
        self.rc2.Open()
        self.ready = True
        try:
            versionFB = self.rc.ReadVersion(self.addressFB)
            versionLR = self.rc2.ReadVersion(self.addressLR)
        except AttributeError as e:
            rospy.logfatal(e.message)
            exit(126)


        self.a_button_last_state = 0
        self.b_button_last_state = 0
        self.scaler = 50


        if not versionFB[0] and not versionLR[0]:
            self.rc = Roboclaw("/dev/ttyACM1", 230400)
            self.rc2 = Roboclaw("/dev/ttyACM0", 230400)
            self.rc.Open()
```

```
            self.rc2.Open()
            versionFB = self.rc.ReadVersion(self.addressFB)
            versionLR = self.rc2.ReadVersion(self.addressLR)

        if not versionFB[0] or not versionLR[0]:
            if not versionFB[0] and versionLR[0]:
                rospy.logfatal("GETVERSION Failed, Only Roboclaw FB")
            elif versionFB[0] and not versionLR[0]:
                rospy.logfatal("GETVERSION Failed, Only Roboclaw LR")
            else:
                rospy.logfatal("GETVERSION Failed, Both Roboclaws after
                    trying switch")
            exit(1)
        else:
            print("RoboFB: " + repr(versionFB[1]) + ", RoboLR: " + repr(
                versionLR[1]))
            print("Car main battery voltage at start of script: ", self.rc
                .ReadMainBatteryVoltage(self.addressFB))

        self.rc.ForwardM1(self.addressFB, 0)
        self.rc.ForwardM2(self.addressFB, 0)
        self.rc2.ForwardM1(self.addressLR, 0)
        self.rc2.ForwardM2(self.addressLR, 0)

        self.timer = time.time()

        rospy.Subscriber("roboclaw", Twist, self.handle_speed)

    def handle_speed(self, msg):
        """
```

```
        Receives the latest motor speed message and sends it to the
            roboclaws
        :param msg: geometry_msgs/Twist
        :return: None
        """
        forward_left_motor = int(msg.linear.x)
        forward_right_motor = int(msg.linear.x)
        right_turning_motor = int(msg.linear.y)
        left_turning_motor = int(msg.linear.z)

        print "Forward: " + str(forward_left_motor)
        print "Right: " + str(right_turning_motor)
        print "Left: " + str(left_turning_motor)

        if forward_left_motor > 0:
            self.rc.ForwardM1(self.addressFB, self.constrain(
                forward_left_motor))
        else:
            self.rc.BackwardM1(self.addressFB, abs(self.constrain(
                forward_left_motor)))

        if forward_right_motor > 0:
            self.rc.ForwardM2(self.addressFB, self.constrain(
                forward_right_motor))
        else:
            self.rc.BackwardM2(self.addressFB, abs(self.constrain(
                forward_right_motor)))

        if right_turning_motor > 0:
            self.rc2.ForwardM1(self.addressLR, self.constrain(
```

```
                right_turning_motor))
        else:
            self.rc2.BackwardM1(self.addressLR, abs(self.constrain(
                right_turning_motor)))

        if left_turning_motor > 0:
            self.rc2.ForwardM2(self.addressLR, self.constrain(
                left_turning_motor))
        else:
            self.rc2.BackwardM2(self.addressLR, abs(self.constrain(
                left_turning_motor)))


    def constrain(self, value):
        """
        Constrains values between -127 and 127
        :param value: value to be constrained
        :return: constrained value
        """
        if value > 127:
            value = 127
        elif value < -127:
            value = -127
        return value
```

Safety Listener

```
#!/usr/bin/env python
import time
import rospy
from sensor_msgs.msg import Joy
```

```
from geometry_msgs.msg import Vector3, Twist
from std_msgs.msg import String


class Watcher:

    def __init__(self):
        self.t0 = time.time()
        self.timeout = True

        self.mode = "manual"
        self.idle = False

        self.new_joy_msg = False
        self.vel_from_joy = Twist()

        self.vel_from_cmd_vel = [[0, 0, 0], [0, 0, 0], [0, 0, 0], [0, 0,
            0], [0, 0, 0]]
        self.new_vel_msg = False

        self.scaler = 50
        self.a_button_last_state = 0
        self.b_button_last_state = 0
        self.y_button_last_state = 0

        self.motor_publisher = rospy.Publisher('pid', Twist, queue_size
            =10)
        self.roboclaw_publisher = rospy.Publisher('roboclaw', Twist,
            queue_size=10)
```

```
    def handle_joy(self, msg):
        """
        Gets the button presses from the xbox controller.
        Sends manual or autonomous control
        :param msg: sensor_msgs/joy
        :return: None
        """
        buttons = msg.buttons

        y_button = buttons[2]

        if y_button and not self.y_button_last_state:
            if self.mode == "manual":
                self.mode = "auto"
            else:
                self.mode = "manual"

        self.y_button_last_state = y_button

    def handle_cmd_vel(self, msg):
        """
        saves the latest velocity message from the nav stack
        :param msg: geometry_msgs/Twist
        :return: None
        """
        vel = [msg.linear.x, msg.linear.y, msg.angular.z]
        self.vel_from_cmd_vel.append(vel)
        self.vel_from_cmd_vel.pop(0)
        if vel[0] == 0 and vel[1] == 0 and vel[2] == 0:
            for i in range(5):
```

```
                self.vel_from_cmd_vel.append(vel)
                self.vel_from_cmd_vel.pop(0)
        self.new_vel_msg = True

    def handle_joy_vel(self, msg):
        """
        saves the latest velocity message from the joy_to_vel node
        :param msg: geometry_msgs/Twist
        :return: None
        """
        self.vel_from_joy = msg
        self.new_joy_msg = True

    def handel_timeout(self, msg):
        """
        resets timer if a new message from safety_publisher is received
        :param msg: standard_msgs/String
        :return: None
        """
        if msg.data == "alive":
            self.t0 = time.time()

    def handle_sys_cmd(self, msg):
        """
        changes operating mode depending on the command from the GUI
        Currently not in use since TCP has not been established with the
            GUI
        :param msg: standard_msgs/String
        :return:
        """
```

```
        if msg.data == "manual":
            self.mode = "manual"
        elif msg.data == "auto":
            self.mode = "auto"
        elif msg.data == "idle":
            self.idle = True
        elif msg.data == "run":
            self.idle = False

    def main(self):
        """
        forwards velocity messages depending on the operating mode of the
            robot
        or sends zero velocity if no connection to pc
        :return: None
        """
        rate = rospy.Rate(10)  # Hz
        while not rospy.is_shutdown():
            rate.sleep()

            if time.time() - self.t0 > 2:
                self.timeout = True
            else:
                self.timeout = False

            if self.timeout:
                msg = Twist()
                msg.linear.x = 0
                msg.linear.y = 0
                msg.linear.z = 0
```

```
                self.roboclaw_publisher.publish(msg)
                rospy.logwarn("Timeout, stoping all motors")
            elif self.idle:
                msg = Twist()
                msg.linear.x = 0
                msg.linear.y = 0
                msg.angular.z = 0
                self.motor_publisher.publish(msg)
            elif self.mode == "manual":
                if self.new_joy_msg:
                    self.motor_publisher.publish(self.vel_from_joy)
                    self.new_joy_msg = False
            elif self.mode == "auto":
                if self.new_vel_msg:
                    vel = [0, 0, 0]
                    for x, y, r in self.vel_from_cmd_vel:
                        vel[0] += x
                        vel[1] += y
                        vel[2] += r
                    vel = [float(c) / len(self.vel_from_cmd_vel) for c in
                       vel]
                    t = Twist()
                    t.linear.x = vel[0]
                    t.linear.y = vel[1]
                    t.angular.z = vel[2]
                    self.motor_publisher.publish(t)
                    self.new_vel_msg = False


if __name__ == '__main__':
```

```
    rospy.init_node("safety_watcher", anonymous=True)
    watch = Watcher()
    rospy.Subscriber("joy", Joy, watch.handle_joy)
    rospy.Subscriber("cmd_vel", Twist, watch.handle_cmd_vel)
    rospy.Subscriber("joy_vel", Twist, watch.handle_joy_vel)
    rospy.Subscriber("ctrl_mode", String, watch.handle_sys_cmd)
    rospy.Subscriber("alive", String, watch.handel_timeout)
    watch.main()
```

F.3 Config-files (ROS)

Parameters for localplanner DWA

```
TrajectoryPlannerROS:
  max_vel_x: 0.2
  min_vel_x: -0.2
  escape_vel: -0.2
  max_vel_theta: 0.2
  min_vel_theta: -0.2
  min_in_place_vel_theta: 0.05

  y_vels: [-0.2, -0.05, 0.05, 0.2]

  acc_lim_theta: 0.05
  acc_lim_x: 0.04
  acc_lim_y: 0.04

  yaw_goal_tolerance: 0.17
  xy_goal_tolerance: 0.2
  latch_xy_goal_tolerance: false
```

```
  holonomic_robot: true

  meter_scoring: true
  sim_time: 6
  sim_granularity: 0.5
  angular_sim_granularity: 0.1
  vx_samples: 5
  vtheta_samples: 25
  heading_scoring: false
  heading_timestep_: 6
  pdist_scale: 0.6
  gdist_scale: 0.8
  occddist_scale: 0.01

  global_frame_id: map

  dwa: true

  oscillation_reset_dist: 0.15
  #heading_lookahead: 0.325
```

Parameters for localplanner TEB

```
TebLocalPlannerROS:

  #Misc Params
  odom_topic: odom
  map_frame: map

  # Speed params
  acc_lim_x: 0.04
```

```
  acc_lim_theta: 0.05
  max_vel_x: 0.2
  max_vel_x_backwards: 0.2
  max_vel_theta: 0.2
  max_vel_y: 0.2
  acc_lim_y: 0.04

  # Footprint
  footprint_model:
    type: "polygon"
    vertices: [[0.5, -0.35], [-0.5, -0.35], [-0.5, 0.35], [0.5, 0.35]]

  # Goal Tolerances
  xy_goal_tolerance: 0.2
  yaw_goal_tolerance: 0.17

  # Trajectory Config
  min_samples: 3
  global_plan_overwrite_orientation: true
  max_global_plan_lookahead_dist: 3.0
  dt_ref: 0.3
  global_plan_viapoint_sep: 1

  # Obstacle Params
  min_obstacle_dist: 0.5
  include_costmap_obstacles: true
  costmap_obstacles_behind_robot_dist: 1.0
  inflation_dist: 0.6
  obstacle_association_force_inclusion_factor: 1.5
  obstacle_association_cutoff_factor: 5
```

```
  obstacle_poses_affected: 30

  # Optimization Params
  weight_max_vel_x: 2.0
  weight_max_vel_theta: 1.0
  weight_acc_lim_x: 1.0
  weight_acc_lim_theta: 1.0

  penalty_epsilon: 0.01

  weight_kinematics_nh: 5.0
  weight_kinematics_forward_drive: 1.0

  weight_obstacle: 50.0
  weight_viapoint: 0.01
  weight_inflation: 0.1
  weight_adapt_factor: 2.0

  # Parallell Planning
  enable_homotopy_class_planning: true
  enable_multithreading: true

  max_number_classes: 4
  selection_cost_hysteresis: 1.0

  viapoints_all_candidates: true
```

Common Costmap Parameters

```
footprint: [[0.5, -0.35], [-0.5, -0.35], [-0.5, 0.35], [0.5, 0.35]]
```

```
global_frame: map
robot_base_frame: base_link
#robot_radius: ir_of_robot

#inflation_radius: 0.2
#
#observation_sources: laser_scan_sensor
#
#laser_scan_sensor: {sensor_frame: laser, data_type: LaserScan, topic:
   scan, marking: true, clearing: true}
```

Global Costmap Parameters

```
global_costmap:
  update_frequency: 5.0
  static_map: true
  transform_tolerance: 0.9

  plugins:
    - {name: static_layer, type: "costmap_2d::StaticLayer"}
    - {name: inflation_layer, type: "costmap_2d::InflationLayer"}

  static_layer:
    map_topic: /map
    subscribe_to_updates: true

  inflation_layer:
    inflation_radius: 0.7
    cost_scaling_factor: 1.0
```

Local Costmap Parameters

```
local_costmap:
  transform_tolerance: 0.9
  update_frequency: 5.0
  publish_frequency: 2.0
  static_map: false
  rolling_window: true
  width: 4.0
  height: 4.0
  resolution: 0.05

  plugins:
    - {name: obstacles, type: "costmap_2d::ObstacleLayer"}
    - {name: inflation_layer, type: "costmap_2d::InflationLayer"}

  obstacles:
    observation_sources: laser
    laser: {data_type: LaserScan, sensor_frame: laser, topic: scan,
       marking: true, clearing: true}
    obstacle_range: 5
    raytrace_range: 3.0

  inflation_layer:
    inflation_radius: 0.7
    cost_scaling_factor: 1.0
```

Move Base Parameters

```
controller_frequency: 10

recovery_behaviors: [    {
                          name: conservative_reset,
```

```
                    type: clear_costmap_recovery/
                        ClearCostmapRecovery
                  }, {
                    name: aggressive_reset,
                    type: clear_costmap_recovery/
                        ClearCostmapRecovery


                  }]
clearing_rotation_allowed: false
```

Appendix G

GUI Code

G.1 Python Code

Flask app

```
import math
from flask import Flask, render_template, url_for, jsonify, request
from div import RosComunication
from threading import Lock, Thread
from div import randomData
from div.ClientUdpSocket import ClientUdpSocket
from div.ClientTcpSocket import ClientTcpSocket
from div import Parser
import cv2
import json
import numpy as np
import os

debug = False
rubenIp = '192.168.0.103'
localHost = "127.0.0.1"
```

```
serverIp = rubenIp

lidarPort = 5005
vesselInfoDataPort = 5006
mapPort = 5007

rosComuncationPort = 5015



UDPLidarSocket = ClientUdpSocket(serverIp, lidarPort, debug=False)
lidarSocketThread = Thread(target=UDPLidarSocket.run)
lidarSocketThread.start()

UDPVesselInfoSocket = ClientUdpSocket(serverIp, vesselInfoDataPort, debug=
    False)
vesselSocketTread = Thread(target= UDPVesselInfoSocket.run)
vesselSocketTread.start()

UDPMapSocket = ClientUdpSocket(serverIp, mapPort, debug=False)
mapSocketThread = Thread(target=UDPMapSocket.run)
mapSocketThread.start()

TCPRosComunicationSocket = ClientTcpSocket(serverIp, rosComuncationPort,
    debug=False)
rosComuncationThread = Thread(target=TCPRosComunicationSocket.run)
rosComuncationThread.start()

app = Flask(__name__)
@app.route('/', methods=["GET", "POST"])
```

```
def homepage():
    """
    sends the homepage for the GUI
    :return: returns the rendered template for the homepage of the GUI
    """
    if request.method == "POST":
        if "Emergency_btn" in request.form:
            print(RosComunication.sendEMERGENCYCommand())
            data = json.dumps(RosComunication.sendEMERGENCYCommand())
            TCPRosComunicationSocket.sendData(data)

        elif "Start_btn" in request.form:
            print(RosComunication.sendStartCommand())
            data = json.dumps(RosComunication.sendStartCommand())
            TCPRosComunicationSocket.sendData(data)

        elif "Stop_btn" in request.form:
            print(RosComunication.sendStopCommand())
            data = json.dumps(RosComunication.sendStopCommand())
            TCPRosComunicationSocket.sendData(data)

        elif "Auto_btn" in request.form:
            print(RosComunication.sendAutonomousCommand())
            data = json.dumps(RosComunication.sendAutonomousCommand())
            TCPRosComunicationSocket.sendData(data)

        elif "Manually_btn" in request.form:
            print(RosComunication.sendManuallyCommand())
            data = json.dumps(RosComunication.sendManuallyCommand())
            TCPRosComunicationSocket.sendData(data)
```

```
    return render_template("index.html")


@app.route("/nav/", methods=["GET", "POST"])
def naviPage():
    """
    sends the navigation page to the GUI.

    :return: returns the navigation page to the GUI
    """
    lidarData = [[0] * 360, [0] * 360]
    motorSpeed1 = 0
    motorspeed2 = 0
    motorSpeed3 = 0
    motorSpeed4 = 0
    vesselSpeedX = 0
    vesselSpeedY = 0
    vesselSpeedTheta = 0

    if request.method == "POST":
        if "Emergency_btn" in request.form:
            print(RosComunication.sendStopCommand())
            data = json.dumps(RosComunication.sendEMERGENCYCommand())
            TCPRosComunicationSocket.sendData(data)
    return render_template("navi.html",
                           lidarData=lidarData,
                           motorSpeed1=motorSpeed1,
                           motorspeed2=motorspeed2,
                           motorSpeed3=motorSpeed3,
```

```
                                motorSpeed4=motorSpeed4,
                                vesselSpeedX = vesselSpeedX,
                                vesselSpeedY = vesselSpeedY,
                                vesselSpeedTheta = vesselSpeedTheta)


@app.route("/nav/update/", methods=["GET", "POST"])
def updateNaviPage():
    """
    sends the updated parameters to the GUI Navigation Page
    :return: returns a json object containing the updated parameters.
    """
    if request.method == "POST":
        if "Emergency_btn" in request.form:
            print(RosComunication.sendEMERGENCYCommand())
            data = json.dumps(RosComunication.sendEMERGENCYCommand())
            TCPRosComunicationSocket.sendData(data)

############_Retrive_data_and_compute_it_#########
    lidarUdpData = UDPLidarSocket.retriveData()
    if debug == True:
        print("lidar Data:")
        print(lidarUdpData)
    lidarData = Parser.lidarDataParser(lidarUdpData)

    vesselUdpData = UDPVesselInfoSocket.retriveData()
    if debug == True:
        print("vessel Data:")
        print(vesselUdpData)
    vesselData = Parser.vesselInfoDataParser(vesselUdpData)
```

```
    mapUdpData = UDPMapSocket.retriveData()
    if debug == True:
        print("map Data:")
        print(mapUdpData)
    image = Parser.mapDataParser(mapUdpData)
    appRoot = os.path.dirname(os.path.abspath(__file__))
    target = os.path.join(appRoot, "static\\map.jpg")
    if image is not None:
        if debug == True:
            print("URL path for image: ")
            print(target)
        cv2.imwrite(target, image)

################_RANDOM_Data_Test################
    # lidarData = randomData.getRandomLidarData(12)
    # motorSpeed1 = randomData.getRandomSpeedData(10)
    # motorSpeed2 = randomData.getRandomSpeedData(10)
    # motorSpeed3 = randomData.getRandomSpeedData(10)
    # motorSpeed4 = randomData.getRandomSpeedData(10)
    #
    # vesselSpeedX = randomData.getRandomSpeedData(10)
    # vesselSpeedY = randomData.getRandomSpeedData(10)
    # vesselSpeedTheta = randomData.getRandomSpeedData(10)
################################################

    return jsonify(points=lidarData[0][::-1],
                   angle=lidarData[1][::-1],
                   motor1=vesselData[0],
                   motor2=vesselData[1],
```

```
                    motor3=vesselData[2],
                    motor4=vesselData[3],
                    vesselSpeedX = vesselData[4],
                    vesselSpeedY = vesselData[5],
                    vesselSpeedTheta = vesselData[6])
                    #need to add update for vessel speed in x,y and theta
                        direction

@app.route("/adv/", methods = ["GET", "POST"])
def advPage():
    """
    sends the advanced page to the GUI.
    :return: returns the rendered template to the GUI.
    """
    if request.method == "POST":
        if "Emergency_btn" in request.form:
            print(RosComunication.sendEMERGENCYCommand())
            data = json.dumps(RosComunication.sendEMERGENCYCommand())
            TCPRosComunicationSocket.sendData(data)

    return render_template("advanced.html")

@app.route("/adv/update/", methods =["GET","POST"])
def updateAdvPage():
    """
    sends the updated parameters to the GUI advanced page.
    :return: returns a json object containing the updated parameters.
    """
    if request.method == "POST":
        if "Emergency_btn" in request.form:
```

```
            print(RosComunication.sendEMERGENCYCommand())
            data = json.dumps(RosComunication.sendEMERGENCYCommand())
            TCPRosComunicationSocket.sendData(data)

    return jsonify()


if __name__ == "__main__":
    app.run(static_url_path= "/static", Debug=True)
```

Parsers

```
import json
import math
import numpy as np
import cv2
import base64
import os
from flask import url_for
debug = True

def lidarDataParser(dataToParse):
    """
    pares the retrieved Lidar data from the udp server to a list with
       points and ranges and filters out "inf" values.
    :param dataToParse: the unparsed retrieved Lidar data from the UDP.
    :return: returns a list of lists with the points and ranges without "
       inf" values
    """
    if dataToParse is not None:
```

```
        parsedData = str(dataToParse.decode("utf-8"))
        parsedData = json.loads(parsedData)
        lidarRanges = parsedData["Lidar"]["Ranges"]
        lidarAngles = parsedData["Lidar"]["Angles"]
        lidarData = [[], []]
        for r, a in zip(lidarRanges, lidarAngles):
            if r != float("inf"):
                lidarData[0].append(r)
                lidarData[1].append(math.degrees(a))
    else:
        lidarData = [[0] * 360, [0] * 360]

    return lidarData


def vesselInfoDataParser(dataToParse):
    """
    pares the retrieved vessel info data from the udp server into a list
       of seven lists containing all motor information,
    and Kalman generated vessel velocity.
    :param dataToParse: the retrived UDP Data from the server.
    :return: a list of seven lists containing all motor information,
    and Kalman generated vessel velocity.
    """

    if dataToParse is not None:
        if debug == True:
            print("recived␣Vessel␣data:␣")
            print(dataToParse)
        parsedData = str(dataToParse.decode("utf-8"))
```

```
        parsedData = json.loads(parsedData)
        forwardRightMotor = parsedData["motors"]["forwardRight"]
        turnLeftMotor = parsedData["motors"]["turnLeft"]
        turnRightMotor = parsedData["motors"]["turnRight"]
        forwardLeftMotor = parsedData["motors"]["forwardLeft"]
        vesselXDir = parsedData["kalman"]["speedX"]
        vesselYDir = parsedData["kalman"]["speedY"]
        vesselThetaDir = parsedData["kalman"]["speedTheta"]
        vesselData = [ int(turnLeftMotor), int(forwardRightMotor), int(
            forwardLeftMotor), int(turnRightMotor), vesselXDir, vesselYDir,
             vesselThetaDir]

    else:
        vesselData = [0, 0, 0, 0, 0, 0, 0]

    return vesselData


# def mapDataParser(dataToParse, savingPath):
#     if dataToParse is not None:
#         mapMetaData = dataToParse.pop()
#         if type(mapMetaData) is not dict:
#             mapMetaData = {"color": "RGB", "width": 800, "height": 600}
   #default meta data
#
#         image = np.array(dataToParse)
#         print(image)
#         if image.size == 144000:
#             image = image.reshape((mapMetaData["height"], mapMetaData["
   width"], 3))
```

```
#                 cv2.imwrite(savingPath, image)

def mapDataParser(dataToParse):
    """
    pares the map data retrieved from the UDP server.
    :param dataToParse: The map data retrieved form the UDP server.
    :return: returns the decoded image.
    """
    if debug == True:
        print("recived map data data: ")
        print(dataToParse)

    stringData = dataToParse
    restoredImage = None
    if stringData is not None:
        orginalImage = base64.b64decode(stringData)
        pngAsNp = np.frombuffer(orginalImage, dtype=np.uint8)
        restoredImage = cv2.imdecode(pngAsNp, flags=1)
    return(restoredImage)
```

ROS Communication

```
def sendStartCommand():
    """
    construct the Start command as a dictionary.
    :return: returns the Start command as a dictionary.
    """
    message ={
```

```
        "topic": "ctrl_mode",
        "value": "run"
    }
    return message

def sendStopCommand():
    """
    construct the Stop command as a dictionary.
    :return: returns the Stop command as a dictionary.
    """
    message = {
        "topic": "ctrl_mode",
        "value": "Stop"
    }
    return message

def sendAutonomousCommand():
    """
    construct the Autonomous command as a dictionary.
    :return: returns the Autonomous command as a dictionary.
    """
    message = {
        "topic": "ctrl_mode",
        "value": "Auto"
    }
    return message

def sendManuallyCommand():
    """
    construct the Manually command as a dictionary.
```

```
    :return: returns the Manually command as a dictionary.
    """
    message = {
        "topic": "ctrl_mode",
        "value": "manual"
    }
    return message


def sendEMERGENCYCommand():
    """
    construct the Emergency stop command as a dictionary.
    :return: returns the Emergency stop command as a dictionary.
    """
    message = {
        "topic": "ctrl_mode",
        "value": "stop"
    }
    return message
```

GUI sockets

UDP socket

```
import socket
import time
import json
from threading import Lock, Thread


class ClientUdpSocket:
    def __init__(self, ip, port,debug=False):
```

```
        """
        creates an UDP connection with specified host at a selected port
        :param ip: the ip that the udp wants to connect to.
        :param port: the port that the udp wants to connect to.
        :param debug: set to true if debug enabled.
        """
        self.socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        self.data = None
        self.ip = ip
        self.port = port
        self.lock = Lock()
        self.shutdown = False
        self.new_data = False
        self.debug = debug

    def run(self):
        """
        runs the socket communication with the desierd server.

        :return: none.
        """
        contact = False
        timeout = 3
        self.socket.settimeout(timeout)
        while not contact:
            self.socket.sendto(b'h', (self.ip, self.port))
            try:
                data, addr = self.socket.recvfrom(2 ** 16)
            except socket.timeout:
                if self.debug == True:
```

```
                print("Exception before connection")
            continue
        except ConnectionResetError:
            if self.debug == True:
                print("No socket avalible!")
            continue

        # if no response: continue
        with self.lock:
            self.data = data
        contact = True
    while not self.shutdown:
        if self.debug == True:
            print('got contact')
        try:
            data, addr = self.socket.recvfrom(2 ** 16)
            with self.lock:
                self.data = data
                self.new_data = True
        except socket.timeout:
            if self.debug == True:
                print("Exception after connection")

    self.socket.sendto(b'END', (self.ip, self.port))

def runMap(self):
    """
    test function that is not in use.
    :return: none.
    """
```

```
        save = False
        start = False
        stop = False
        imData = []
        contact = False
        timeout = 3
        self.socket.settimeout(timeout)
        while not contact:
            self.socket.sendto(b'h', (self.ip, self.port))
            try:
                data, addr = self.socket.recvfrom(2 ** 16)
            except socket.timeout:
                if self.debug == True:
                    print("Exception before connection")
                continue
            except ConnectionResetError:
                if self.debug == True:
                    print("No socket avalible!")
                continue

            # if no response: continue
            contact = True
        while not self.shutdown:
            if self.debug == True:
                print('got contact')

            try:
                data, addr = self.socket.recvfrom(2 ** 16)

                if not start and data == b"start::":
```

```
            print("start recived")
            save = True
            start = True
            stop = False

        elif save and data == b"end::":
            print("end recived")
            with self.lock:
                self.data = imData
                self.new_data = True
                save = False
                stop = True
                start = False
                imData = []

        elif not stop and data == b"start::":
            print("end packet lost!")
            imData = []

        elif save == True:
            packet = json.loads(data)

            if type(packet) is list:
                #for item in packet:
                    #imData.append(item)
                imData.append(packet)

            elif type(packet) is dict:
                imData.append(packet)
```

```
                else:
                    imData = []


            except socket.timeout:
                if self.debug == True:
                    print("Exception after connection")

        self.socket.sendto(b'END', (self.ip, self.port))


    def retriveData(self):
        """
        retrives the data from the connected UDP server.

        :return: returns the retrived data from the UDP server.
        """
        data = None
        with self.lock:
            data = self.data
            self.new_data = False

        return data


if __name__ == '__main__':
    """
    test function to test the communcation.
    """
    client_socket = ClientUdpSocket('127.0.0.1', 5005)
```

```
    thread = Thread(target=client_socket.run)
    thread.start()

    for i in range(9):
        while not client_socket.new_data:
            pass
        print(client_socket.retriveData())

    client_socket.shutdown = True
    time.sleep(1)
```

TCP socket

```
import socket
import time
from threading import Lock, Thread


class ClientTcpSocket:


    def __init__(self, ip, port, debug=False):
        """
        creats a TCP socket object

        :param ip: desierd ip to connect to
        :param port: type of port to connect to
        :param debug: set to true if you want do debugg
```

```
        """
        self.socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.ip = ip
        self.port = port
        self.debug = debug
        self.data = None
        self.isConnected = False
        self.isDisconnected = False


    def run(self):
        """
        starts the socket communication
        :return: none.
        """

        while not self.isConnected:
            self.connect()
        while not self.isDisconnected:
            time.sleep(1)


    def connect(self):
        """
        tries to connect to desierd ip and port, and throws exception if
           no connection is avalible

        :return:none.
        """
        try:
```

```
            self.socket.connect((self.ip, self.port))
            if self.debug:
                print("got connection!")
            self.isConnected = True

        except OSError:
            if self.debug:
                print("no socket available!")
            pass

    def shutdown(self):
        """
        shuts down the connection from the server.

        :return:none
        """
        self.socket.close()
        self.isDisconnected = True

    def sendData(self, data: str):
        """
        sends desired data to the server.

        :param data: desired data to send to server
        :return:none.
        """
        if self.isConnected:
            data = data
            self.socket.sendall(data.encode())
        else:
```

```
            if self.debug:
                print("cant send message, no connection established!")
            pass
```

G.2 HTML Code

Base Page

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <link rel="stylesheet" href="{{ url_for('static',filename='css/base.
        CSS') }}">
    <link rel="shortcut icon" href="{{ url_for('static', filename='
        favicon3.ico') }}">
    <script src="{{ url_for('static',filename='Js/jquery-3.4.1.min.js') }}"
        ></script>
    <script src="{{ url_for('static', filename='Js/base.js') }}"></script>
    <script src="{{ url_for('static', filename='Js/plotly-latest.min.js') }
        }"></script>
    {% block head %}{% endblock %}

</head>

<body>

    <header>
```

```
<div class="flex-header">

    <div class="groupPhoto">
        <img src="{{ url_for('static',filename= 'favicon3.ico') }}
            " width="120" height="120" >
    </div>

    <div class ="homeBox">

        <form id="homebox" method="GET" action="/">
            <p1 class="text" id="naviTopic"> Home:</p1>
            <input class="btnLayout" id="homeBtn" type="submit"
                value="go to Home">
        </form>

    </div>

    <div class="navBox">

        <form id="navbox" method="GET" action="/nav">
            <p1 class="text" id="naviTopic"> navigation:</p1>
            <input class="btnLayout" id="naviBtn" type="submit"
                value="go to Navigation">
        </form>

    </div>


    <div class="tecBox">
```

```
                <form id="tecbox" method="GET" action="/adv">
                    <p2 class="text" id ="tecTopic"> Advanced:</p2>
                    <input class="btnLayout" id="tecBtn" type="submit"
                        value="go to Advanced">
                </form>

            </div>


        </div>

    </header>

    <div class="contactModal" id="contactModal"> <!-- will add a class of
        active-->

        <div class ="modal-header active">
            <div class="ModalTitle">Contact information</div>
            <button data-close-button class="modalClose" id="closeModal">&
                times;</button>
        </div>

        <div class="modal-body">
            <dl class = "mainList">
            <dl class="info" id ="hkList">
                <dt>H kon Bjerkgaard Waldum</dt>
                <dt>haakonbw@stud.ntnu.no</dt>
            </dl>
```

```
            <dl class= "info" id = "rnkList">
                <dt>Ruben Ole Berg Natvik</dt>
                <dt>ronatvik@stud.ntnu.no</dt>
            </dl>

            <dl class ="info" id = "rsjList">
                <dt>Ruben Svedal J rundland</dt>
                <dt>rubensj@stud.ntnu.no</dt>
            </dl>

            <dl class = "info" id = "vrwList">
                <dt>Vebj rn Rimstad Wille</dt>
                <dt>Vebjorrw@ntnu.no</dt>
            </dl>

        </dl>
        </div>
    </div>

    <div  id="overlay"></div> <!--will add a class of active-->

        {% block body %}{% endblock %}

    <footer>
        <div class = "flex-footer">s

            <div class = "ModalBox">
                <button data-modal-target="#contactModal" class="
                    ContactBtn" id="contactBtn" >contact</button>
            </div>
```

```
            <div class = "EMERGENCY">
                <form action ="#" method ="post">
                    <input class="EMERGENCYBnt" id="emergBtn" type="submit
                        " name="Emergency_btn" value="EMERGENCY␣STOP!">
                </form>
            </div>



        </div>
    </footer>


</body>
</html>
```

Home Page

```
{% extends "base.html"%}

{% block head %}

    <link rel="stylesheet" href="{{␣url_for('static',␣filename='css/index.
        css')␣}}">
    <script src="{{␣url_for('static',␣filename='Js/Index.js')␣}}"></script
        >
    <title>home page</title>

{% endblock %}
```

```
{% block body %}
 <div class="FlexBody">

     <div class = "FlexHeader">
            <h1 class = "topicText" >welcome to Home</h1>
     </div>


     <div class ="flexPannel">

         <div class = "FlexOnOffTopic">
             <h2 class ="PannelText" id ="toggleTxt">Start/stop the vessel
                 </h2>
         </div>

         <form action ="#" method ="post">
             <div class="FlexOnButton">
                 <input class="Buttons" id="startBtn" type="submit" name="
                     Start_btn"  value="Start">
             </div>

             <div class ="FlexOffButton">
                 <input class="Buttons" id="stopBtn" type="submit" name="
                     Stop_btn" value="Stop">
             </div>
         </form>

         <div class ="FlexToggleTopic">
             <h3 class ="PannelText" id="selectTxt"> please select driver
```

```
                mode </h3>
            </div>

            <form action="#" method="post">
                <div class="FlexAutonomous">
                    <input class="Buttons" id="autoBtn" type="submit" name="
                        Auto_btn" value="Autonomous">
                </div>
                <div class="FlexManually">
                    <input class="Buttons" id="manuallyBtn" type="submit"
                        name="Manually_btn" value="Manually control">
                </div>
            </form>

            <div class="FlexAboutBtn">
                <input class="About" id="aboutBtn" type="submit" name="
                    About_btn" value="About">
            </div>

        </div>
    </div>


<script>

</script>

{% endblock %}
```

Navigation Page

```
{% extends "base.html" %}

{% block head %}

    <link rel="stylesheet" href="{{ url_for('static',filename = 'css/navi.
        CSS') }}">

    <script src="{{ url_for('static', filename='Js/Navi.js') }}"></script>

    <title>Navigation</title>

{% endblock %}

{% block body %}

    <div class="naviBodyFlex">
        <div class = "topicBox">
            <h1 class = "topicText" >welcome to navigation</h1>
        </div>

        <div class = "graphs">
            <div class = "flexCharts">
                <div class = "charts">
                    <div id="pointCloud"></div> <!--where the plot for the
                        LIDAR is-->
                </div>
            </div>
            <div class = "flexGauge">
```

```
            <div class ="motorGauge">
                <div id = "speedMotor1"></div>  <!--where the plot for
                    the motor1 is-->
                <div id = "speedMotor2"></div>  <!--where the plot for
                    the motor1 is-->
                <div id = "speedMotor3"></div>  <!--where the plot for
                    the motor1 is-->
                <div id = "speedMotor4"></div>  <!--where the plot for
                    the motor1 is-->
            </div>
        </div>
        <div class = "FlexMap">
            <div id ="Map">
                <img id ="mapId" src = "{{url_for('static', filename='
                   map.jpg') }}">  <!-- MAP GOES HERE-->
            </div>
        </div>
    </div>

    <div class = "flexVesselSpeed">
        <div id ="speedX"></div> <!-- where the speed x is plotted-->
        <div id ="speedY"></div> <!-- where the speed y is plotted-->
        <div id ="speedTheta"></div> <!-- where the speed theta is
           plotted-->
    </div>
</div>

<script>

    navi = new Navi();
```

```
        let minSpeed = -100;
        let maxSpeed = 100; //depending on max speed of given motor
        let minVesselSpeed = -0.5;
        let maxVesselSpeed = 0.5; //the max speed of the vessel
        let updateFrequency = 250; //in milli seconds

//////////////////// MOTOR1 ////////////////////////////////
        /*
        plots the motor gauge for motor 1
         */
        let motor1 = navi.plotSpeed({{ motorSpeed1 }},minSpeed, maxSpeed,"
           Speed␣Motor␣1");
        Plotly.react("speedMotor1",motor1[0],motor1[1]);
////////////////////////////////////////////////////////////

//////////////////// MOTOR2 ////////////////////////////////
         /*
        plots the motor gauge for motor 2
         */
        let motor2 = navi.plotSpeed({{ motorspeed2 }}, minSpeed, maxSpeed,
            "Speed␣Motor␣2");

        Plotly.react("speedMotor2",motor2[0],motor2[1]);
////////////////////////////////////////////////////////////

//////////////////// MOTOR3 ////////////////////////////////
         /*
        plots the motor gauge for motor 3
         */
        let motor3 = navi.plotSpeed({{ motorSpeed3 }},minSpeed, maxSpeed,"
```

```
            Speed␣Motor␣3");
        Plotly.react("speedMotor3",motor3[0],motor3[1]);
////////////////////////////////////////////////////////////

////////////////////// MOTOR4 ////////////////////////////////
         /*
        plots the motor gauge for motor 4
         */
        let motor4 = navi.plotSpeed({{ motorSpeed4 }},minSpeed, maxSpeed,"
            speed␣Motor␣4");
        Plotly.react("speedMotor4",motor4[0],motor4[1]);
////////////////////////////////////////////////////////////




////////////////////// LidarPointMap  ////////////////////////
         /*
        plots the Lidar chart.
         */
        let lidarList = navi.plotLidar({{lidarData}});
        Plotly.plot("pointCloud",lidarList[0],lidarList[1]);
////////////////////////////////////////////////////////////

////////////////////// vesselSpeeds  ////////////////////////
         /*
        plots the speed gauge for velocity in x, y and theta.
         */

        let vesselSpeedX = navi.plotSpeed({{vesselSpeedX}},maxVesselSpeed,
```

```
            "speed in X dir");
        Plotly.plot("speedX",vesselSpeedX[0],vesselSpeedX[1]);

        let vesselSpeedY = navi.plotSpeed({{ vesselSpeedY }},
            maxVesselSpeed,"speed in Y dir");
        Plotly.plot("speedY",vesselSpeedY[0],vesselSpeedY[1]);

        let vesselSpeedTheta = navi.plotSpeed({{vesselSpeedTheta}},
            maxVesselSpeed,"speed in Theta dir");
        Plotly.plot("speedTheta",vesselSpeedTheta[0],vesselSpeedTheta[1]);

////////////////////////////////////////////////////////////

////////////// update Graphs//////////////////////////

        $(document).ready(function () {
            setInterval(function(){
                /*
                creates an interval function that gets recalled every
                    updateFrequency.
                it gets called when the pages is loaded and re plotts all
                    the charts for motors, lidar, chart and
                velocity for the vessel.
                 */

                    $.getJSON("/nav/update/",function (updatedData) {
                        let updatedDataset= [updatedData.points,
                            updatedData.angle];
                        let lidarUpdate = navi.plotLidar(updatedDataset);
```

```
let updatedMotor1 = navi.plotSpeed(updatedData.
    motor1,minSpeed, maxSpeed,"Turn␣Left␣Motor");
let updatedMotor2 = navi.plotSpeed(updatedData.
    motor2,minSpeed, maxSpeed,"Fwd␣Right␣Motor");
let updatedMotor3 = navi.plotSpeed(updatedData.
    motor3,minSpeed, maxSpeed,"Fwd␣Left␣Motor");
let updatedMotor4 = navi.plotSpeed(updatedData.
    motor4,minSpeed, maxSpeed,"Turn␣Right␣Motor");

let updatedSpeedX = navi.plotSpeed(updatedData.
    vesselSpeedX, minVesselSpeed, maxVesselSpeed,"
    Speed␣in␣X␣dir");
let updatedSpeedY  = navi.plotSpeed(updatedData.
    vesselSpeedY, minVesselSpeed, maxVesselSpeed,"
    Speed␣in␣Y␣dir");
let updatedSpeedTheta = navi.plotSpeed(updatedData
    .vesselSpeedTheta, minVesselSpeed,
    maxVesselSpeed,"Speed␣in␣Theta␣dir")


let d = new Date()
let path = "{{␣url_for('static',␣filename='map.jpg
    ')}}?"
$("#mapId").attr("src", path + d.getTime());

Plotly.plot("pointCloud",lidarUpdate[0],
    lidarUpdate[1]);

Plotly.newPlot("speedMotor1",updatedMotor1[0],
    updatedMotor1[1]);
```

```
                    Plotly.newPlot("speedMotor2",updatedMotor2[0],
                        updatedMotor2[1]);
                    Plotly.newPlot("speedMotor3",updatedMotor3[0],
                        updatedMotor3[1]);
                    Plotly.newPlot("speedMotor4",updatedMotor4[0],
                        updatedMotor4[1]);

                    Plotly.newPlot("speedX",updatedSpeedX[0],
                        updatedSpeedX[1]);
                    Plotly.newPlot("speedY",updatedSpeedY[0],
                        updatedSpeedX[1]);
                    Plotly.newPlot("speedTheta",updatedSpeedTheta[0],
                        updatedSpeedTheta[1]);
                });

        }, updateFrequency);



        //Might want to impliment one more set intervall function for
            speeds, to desync them from lidar update.
    });
////////////////////////////////////////////////////
</script>


{% endblock %}
```

Advanced Page

```
{% extends "base.html" %}

{% block head %}

    <link rel="stylesheet" href="{{ url_for('static', filename='css/
        advanced.css') }}">
    <title>Advanced</title>

{% endblock %}

{% block body %}
<div class = "FlexBody">

    <div class="FlexHeader">
        <h1 class = "topicText">welcome to advanced info!</h1>
    </div>

    <div class="FlexInfoPannels">

        <div class ="XboxPannel">
            <div class="XboxText">
                <h2 class ="bodyText">Manual control info</h2>
            </div>

            <div class="XboxInfo">


            </div>

        </div>
```

```
        <div class="PIDPannel">
            <div class="PidText">
                <h3 class="bodyText">controller info</h3>
            </div>

        </div>

    </div>



</div>



{% endblock %}
```

G.3 CSS Code

Style for Base page

```
.flex-header {
    background-color: #7f8c8d;
    display: flex;
    position: fixed;
    flex-direction: row;
    justify-content: flex-end;
    left: 0;
```

```
    top: 0;
    z-index: 5;
    width: 100%;
    margin: 0;
}


.flex-footer {
    background-color: #7f8c8d;
    position: fixed;
    left: 0;
    bottom: 0;
    width: 100%;
    padding: 10px 0px;
    margin: 0;

    display: flex;
    flex-direction: row;
    justify-content: space-between;
    align-items: baseline;

}

.EMERGENCY{
    order: 0;
}

.ModalBox{
    order: 1;
```

```
}

.groupPhoto {
    order: 1;

    flex-basis: 4000px;
    margin: 10px 0px 3px 10px;
    padding: 0;

}

.homeBox {
    order: 2;

    text-align: center;
    margin: 3px;
    padding: 0;
}

.navBox {
    order: 3;

    text-align: center;
    margin: 3px;
    padding: 0;

}


.tecBox {
```

```
    order: 4;

    text-align: center;
    margin: 3px;
    padding: 0;
}


.text {
    color: white;
    font-size: large;
    font-family: sans-serif;
    height: 15px;
}


.btnLayout {
    border: 0;
    background: rgb(255, 255, 255);
    display: block;
    margin: 20px auto;
    text-align: center;
    border: 3px solid black;
    padding: 10px 5px;
    width: 200px;
    transition: 0.3s;
    cursor: pointer;
}

.btnLayout:hover {
```

```
    background: #177199;
}



.ContactBtn {
    border: 0;
    background: rgb(255, 255, 255);
    display: block;
    margin: 3px 3px 0px 3px ;
    text-align: center;
    border: 3px solid black;
    padding: 10px 5px;
    width: 100px;
    transition: 0.3s;
}

.ContactBtn:hover{
    background: #177199;
    cursor: pointer;
}
.EMERGENCYBnt{
    border: 0;
    background: red;
    display: block;
    margin: 3px 3px 0px 3px ;
    text-align: center;
    color: white;
    border: 3px solid black;
    padding: 10px 5px;
    width: 300px;
```

```
    height: 50px;
    transition: 0.3s;

}

.EMERGENCYBnt:hover{
    background: darkred;
    cursor: pointer;
}

.contactModal {
    position: fixed;
    top: 50%;
    left: 50%;
    transform: translate(-50%, -50%) scale(0);
    transition: 200ms   ease-in-out;
    border: 2px solid black;
    border-radius: 25px;
    z-index: 10;
    background-color: #177199;
    width: 500px;
    height:500px;
}

.contactModal.active{
    transform: translate(-50%, -50%) scale(1);
}

.modal-header{
    padding:10px 15px;
```

```
    display: flex;
    justify-content: space-between;
    align-items: center;
    border-bottom: 2px solid black;
}

.modal-header .ModalTitle{
    font-size:25px;
    font-weight:bold;
    color: white;

}

.modal-header .modalClose{
    cursor: pointer;
    border:none;
    outline: none;
    background: none;
    font-size: 1.25rem;
    font-weight: bold;
}

.modal-body{
    text-align: center;

}

.modal-body .info{
    font-family: sans-serif;
    font-size: 20px;
```

```
    color:white;
}

#overlay{
    position: fixed;
    opacity: 0;
    transition: 200ms ease-in-out;
    top:0;
    left:0;
    right: 0;
    bottom: 0;
    background-color: rgba(0,0,0,.5);
    pointer-events: none;
}

#overlay.active{
    opacity: 1;
    pointer-events: all;
}
```

Style for Index page

```
.FlexBody{
    margin: 150px 0px 10px 0px;
    display: flex;
    flex-direction: column;
    flex-wrap: wrap;
    background: white;
}
```

```
.flexPannel{
    order: 1;
    display:flex;
    flex-direction: column;
    flex-wrap: wrap;

    border:solid black;
    background: #485F6E;
    align-self: center;
    justify-content: space-around;
    align-items: center;

    width: 700px;
    height: 1000px;
}

.FlexOnOffTopic{
    order:0;
}

.FlexOnButton{
    order:1;
}

.FlexOffButton{
    order:2;
}

.FlexToggleTopic{
    order:0;
```

```
}

.FlexAutonomous{
    order:4;
}

.FlexManually{
    order:5;
}

.FlexAboutBtn{
    order:6;
}

.PannelText{
    color: white;
    text-align: center;
    font-family: serif;
    font-size: 50px;
}


.FlexHeader{
    order:1;
    height:75px;
    width: 400px;
    margin: 0px 0px 7px 0px;
    border: solid black;
    background-color: rgb(235, 235, 235); /*change dependent on the final
        backgorund color */
```

```
    text-align: center;
}

.topicText{
    margin: 0px 0px 0px 0px;
    font-family: serif;
    color: black;
}

.Buttons{
    border: 0;
    background: rgb(255, 255, 255);
    display: block;
    margin: 20px auto;
    text-align: center;
    font-size: 30px;
    border: 3px solid black;
    padding: 10px 5px;
    width: 300px;
    height: 75px;
    transition: 0.3s;
    cursor: pointer;
}

#startBtn:hover{
    background: #3A7734;
    cursor: pointer;
}

#stopBtn:hover{
```

```
    background: red;
    cursor: pointer;
}

#autoBtn:hover{
    background: #177199;
    cursor:pointer;
}

#manuallyBtn:hover{
    background: #177199;
    cursor:pointer;
}

#homeBtn{
    background: #485F6E;     /*change dependent on the final backgorund
        color */
}
```

Style for Navigation page

```
{% extends "base.html" %}

{% block head %}

    <link rel="stylesheet" href="{{ url_for('static',filename = 'css/navi.
        CSS')}}">

    <script src="{{ url_for('static', filename='Js/Navi.js') }}"></script>
```

```
    <title>Navigation</title>

{% endblock %}

{% block body %}

    <div class="naviBodyFlex">
        <div class = "topicBox">
            <h1 class = "topicText" >welcome to navigation</h1>
        </div>

        <div class = "graphs">
            <div class = "flexCharts">
                <div class = "charts">
                    <div id="pointCloud"></div> <!--where the plot for the
                        LIDAR is-->
                </div>
            </div>
            <div class = "flexGauge">
                <div class ="motorGauge">
                    <div id = "speedMotor1"></div>  <!--where the plot for
                        the motor1 is-->
                    <div id = "speedMotor2"></div>  <!--where the plot for
                        the motor1 is-->
                    <div id = "speedMotor3"></div>  <!--where the plot for
                        the motor1 is-->
                    <div id = "speedMotor4"></div>  <!--where the plot for
                        the motor1 is-->
                </div>
            </div>
```

```
            <div class = "FlexMap">
                <div id ="Map">
                    <img id ="mapId" src = "{{url_for('static', filename='
                        map.jpg') }}">  <!-- MAP GOES HERE-->
                </div>
            </div>
        </div>

        <div class = "flexVesselSpeed">
            <div id ="speedX"></div> <!-- where the speed x is plotted-->
            <div id ="speedY"></div> <!-- where the speed y is plotted-->
            <div id ="speedTheta"></div> <!-- where the speed theta is
                plotted-->
        </div>
    </div>

    <script>

        navi = new Navi();
        let minSpeed = -100;
        let maxSpeed = 100; //depending on max speed of given motor
        let minVesselSpeed = -0.5;
        let maxVesselSpeed = 0.5; //the max speed of the vessel
        let updateFrequency = 250; //in milli seconds

////////////////////// MOTOR1 ////////////////////////////////
        /*
        plots the motor gauge for motor 1
         */
        let motor1 = navi.plotSpeed({{ motorSpeed1 }},minSpeed, maxSpeed,"
```

```
            Speed Motor 1");
        Plotly.react("speedMotor1",motor1[0],motor1[1]);
////////////////////////////////////////////////////////////

////////////////////// MOTOR2 ////////////////////////////////
         /*
        plots the motor gauge for motor 2
         */
        let motor2 = navi.plotSpeed({{ motorspeed2 }}, minSpeed, maxSpeed,
            "Speed Motor 2");

        Plotly.react("speedMotor2",motor2[0],motor2[1]);
////////////////////////////////////////////////////////////

////////////////////// MOTOR3 ////////////////////////////////
         /*
        plots the motor gauge for motor 3
         */
        let motor3 = navi.plotSpeed({{ motorSpeed3 }},minSpeed, maxSpeed,"
           Speed Motor 3");
        Plotly.react("speedMotor3",motor3[0],motor3[1]);
////////////////////////////////////////////////////////////

////////////////////// MOTOR4 ////////////////////////////////
         /*
        plots the motor gauge for motor 4
         */
        let motor4 = navi.plotSpeed({{ motorSpeed4 }},minSpeed, maxSpeed,"
           speed Motor 4");
        Plotly.react("speedMotor4",motor4[0],motor4[1]);
```

```
/////////////////////////////////////////////////////////////




///////////////////// LidarPointMap ////////////////////////
        /*
        plots the Lidar chart.
        */
        let lidarList = navi.plotLidar({{lidarData}});
        Plotly.plot("pointCloud",lidarList[0],lidarList[1]);
/////////////////////////////////////////////////////////////

///////////////////// vesselSpeeds ////////////////////////
        /*
        plots the speed gauge for velocity in x, y and theta.
        */

        let vesselSpeedX = navi.plotSpeed({{vesselSpeedX}},maxVesselSpeed
            ,"speed in X dir");
        Plotly.plot("speedX",vesselSpeedX[0],vesselSpeedX[1]);

        let vesselSpeedY = navi.plotSpeed({{ vesselSpeedY }},
            maxVesselSpeed,"speed in Y dir");
        Plotly.plot("speedY",vesselSpeedY[0],vesselSpeedY[1]);

        let vesselSpeedTheta = navi.plotSpeed({{vesselSpeedTheta}},
            maxVesselSpeed,"speed in Theta dir");
        Plotly.plot("speedTheta",vesselSpeedTheta[0],vesselSpeedTheta[1]);
```

```
//////////////////////////////////////////////////////////

////////////// update Graphs//////////////////////////

        $(document).ready(function () {
            setInterval(function(){
                /*
                creates an interval function that gets recalled every
                   updateFrequency.
                it gets called when the pages is loaded and re plotts all
                   the charts for motors, lidar, chart and
                velocity for the vessel.
                 */

                    $.getJSON("/nav/update/",function (updatedData) {
                        let updatedDataset= [updatedData.points,
                           updatedData.angle];
                        let lidarUpdate = navi.plotLidar(updatedDataset);

                        let updatedMotor1 = navi.plotSpeed(updatedData.
                           motor1,minSpeed, maxSpeed,"Turn Left Motor");
                        let updatedMotor2 = navi.plotSpeed(updatedData.
                           motor2,minSpeed, maxSpeed,"Fwd Right Motor");
                        let updatedMotor3 = navi.plotSpeed(updatedData.
                           motor3,minSpeed, maxSpeed,"Fwd Left Motor");
                        let updatedMotor4 = navi.plotSpeed(updatedData.
                           motor4,minSpeed, maxSpeed,"Turn Right Motor");

                        let updatedSpeedX = navi.plotSpeed(updatedData.
                           vesselSpeedX, minVesselSpeed, maxVesselSpeed,"
```

```
    Speed in X dir");
let updatedSpeedY  = navi.plotSpeed(updatedData.
    vesselSpeedY, minVesselSpeed, maxVesselSpeed,"
    Speed in Y dir");
let updatedSpeedTheta = navi.plotSpeed(updatedData
    .vesselSpeedTheta, minVesselSpeed,
    maxVesselSpeed,"Speed in Theta dir")



let d = new Date()
let path = "{{ url_for('static', filename='map.jpg
    ')}}?"
$("#mapId").attr("src", path + d.getTime());

Plotly.plot("pointCloud",lidarUpdate[0],
    lidarUpdate[1]);

Plotly.newPlot("speedMotor1",updatedMotor1[0],
    updatedMotor1[1]);
Plotly.newPlot("speedMotor2",updatedMotor2[0],
    updatedMotor2[1]);
Plotly.newPlot("speedMotor3",updatedMotor3[0],
    updatedMotor3[1]);
Plotly.newPlot("speedMotor4",updatedMotor4[0],
    updatedMotor4[1]);

Plotly.newPlot("speedX",updatedSpeedX[0],
    updatedSpeedX[1]);
Plotly.newPlot("speedY",updatedSpeedY[0],
    updatedSpeedX[1]);
```

```
                    Plotly.newPlot("speedTheta",updatedSpeedTheta[0],
                        updatedSpeedTheta[1]);
                });

        }, updateFrequency);



        //Might want to impliment one more set intervall function for
            speeds, to desync them from lidar update.
    });
////////////////////////////////////////////////////////
    </script>


{% endblock %}
```

Style for advanced page

```
.FlexBody{
    margin: 150px 0px 10px 0px;
    display: flex;
    flex-direction: column;
    flex-wrap: wrap;
    background: white;
}
.FlexHeader{
    order:0;
    height:75px;
    width: 400px;
```

```
    margin: 0px 0px 7px 0px;
    border: solid black;
    background-color: rgb(235, 235, 235); /*change dependent on the final
       backgorund color */
    text-align: center;
}

.topicText{
    margin: 0px 0px 0px 0px;
    font-family: serif;
    color: black;
}

.bodyText{
    font-family: sans-serif;
    color:white;
    font-size: 30px;

}

.FlexInfoPannels{
    order:1;
    display: flex;
    flex-direction: Row;
    flex-wrap: wrap;
    justify-content: flex-start;
    width: 100%;
}

.XboxPannel{
```

```
    order: 0;
    display:flex;
    flex-direction: column;
    flex-wrap: wrap;
    justify-content: flex-start;

    border: solid black;
    background: #485F6E;
    align-self: flex-start;
    align-items: center;
    margin:0px 30px 0px 0px;
    width: 700px;
    height: 500px;
}

.XboxText{
    order:0;
    align-self: flex-start;
}

.XboxInfo{
    order:1;
    display: flex;
    flex-direction: row;
    border:solid black;
    align-self: flex-start;
    height:150px;
    width:600px;
    background: rgb(235, 235, 235);
}
```

```
.PIDPannel{
    order: 1;
    display:flex;
    flex-direction: column;
    flex-wrap: wrap;

    border: solid black;
    background: #485F6E;
    align-self: flex-start;
    justify-content: space-around;
    align-items: center;
    margin: 0px 0px 0px 0px;
    width: 700px;
    height: 500px;

}

#tecBtn{
    background: #485F6E;
}
```

G.4 JavaScript

Base page JavaScript

```
$(document).ready(function () {

    /*
    creates a function that runs after the page has loaded with the help
```

```
        of jQuery.
    it creates the action to the about button, that make it pops up when
        pressed.

    return: returns none.
     */

    const openModalButton = document.getElementById("contactBtn");
    const closeModalButton = document.getElementById("closeModal");
    const overlay = document.getElementById("overlay");
    const contactModal = document.getElementById("contactModal");

    openModalButton.addEventListener("click",()=>{
        openModal(contactModal);
    });

    closeModalButton.addEventListener("click", ()=>{
        closeModal(contactModal);
    });

function openModal(modal) {
        if (modal == null) return
        modal.classList.add("active");
        overlay.classList.add("active");
    }

    function closeModal(modal) {
        if (modal == null) return;
        modal.classList.remove("active");
        overlay.classList.remove("active");
```

```
    }


});
```

Navigation page JavaScript

```
class Navi {

    constructor() {

    }

    plotLidar(data) {
        /*
        creates a standardisation for the lidar data layup for plotly.js
            lib
        @param {object} data - contains the data to be plotted by the
            plotly.js lib
        return - returns the lidar data and the layout to the plot.
         */

        let lidarData = [{
            type: "scatter",
            mode: "markers",
            marker: {
                color: 'rgb(27,158,119)',
                size: 10,
            },
            name: "lidar Points",
```

```
            r: data[0],
            t: data[1],
        }];

        let layout = {
            height: 600,
            width: 600,
            title: {text: 'Lidar Map'},
            font: {size: 5},
            plot_bgcolor: 'rgb(225, 225, 225)',
            angularaxis: {tickcolor: 'rgb(253,253,255)'},
            paper_bgcolor: "rgb(235, 235, 235)"
        };

        return [lidarData, layout];
    }

    plotSpeed(speedData, minSpeed, maxSpeed, indicatorName ) {
        /*
        computes the data to be displayed in a speed gauge plotted by
           plotly.js alongside with the layup for the
        plotly.js chart.

        @param {object} SpeedData     - the data to be plotted.
        @param {object} minSpeed      - minimum threshold value for speed.
        @param {object} maxSpeed      - maximum threshold value for speed.
        @param {object} indicatorName - The name that wil be displayed
           over the indicator.
        return - returns the motor data to be plotted alongside with the
           layout of the plot.
```

```
        */
        let topSpeed = maxSpeed;
        let speed = speedData;

        let motorData = [{
            domain: {x: [0, 1], y: [0, 1]},
            value: speed,
            title: {text: indicatorName},
            type: "indicator",
            mode: "gauge+number",
            gauge:{
                axis: {range: [minSpeed, topSpeed]},

            }
        }];

        let motorLayout = {
            width: 300,
            height: 300,
            paper_bgcolor: "rgb(235, 235, 235)"
        };
        return [motorData, motorLayout]
    }
/*
    plotVesselSpeed(speedData, maxSpeed, speedDirection){
        let speed = speedData;
        let maxDisplaySpeed = maxSpeed;

        let vesselSpeedData = [{
            domain: {x: [0,1], y: [0,1]},
```

```
            value:speed,
            title: {text:speedDirection},
            type: "indicator",
            mode: "gauge",
            gauge: {
                axis: {range: [null, maxDisplaySpeed]},
                },
        }];

        let vesselSpeedLayout = {
            height: 300,
            width: 500,
            paper_bgcolor: "rgb(235, 235, 235)",
        };

        return [vesselSpeedData, vesselSpeedLayout]
    }

  */
}
```

Appendix H

Arduino Code

```
#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_LSM303_U.h>
#include <Adafruit_BMP085_U.h>
#include <Adafruit_L3GD20_U.h>
#include <Adafruit_10DOF.h>

/* Assigning a unique ID to the sensors */
Adafruit_LSM303_Accel_Unified accel = Adafruit_LSM303_Accel_Unified(30301)
    ;
Adafruit_LSM303_Mag_Unified   mag   = Adafruit_LSM303_Mag_Unified(30302);
Adafruit_BMP085_Unified       bmp   = Adafruit_BMP085_Unified(18001);
Adafruit_L3GD20_Unified       gyro  = Adafruit_L3GD20_Unified(20);

void setup(void)
{
  Serial.begin(115200);

  accel.begin();
  gyro.begin();
```

```
  mag.begin();

}

void loop(void)
{

  sensors_event_t event;

  /* accelerometer values in m/s^2 */
  accel.getEvent(&event);
  float imu1_acc_x = event.acceleration.x;
  float imu1_acc_y = event.acceleration.y;
  float imu1_acc_z = event.acceleration.z;

  /* gyrocope values in rad/s */
  gyro.getEvent(&event);
  float imu1_gyro_x = event.gyro.x;
  float imu1_gyro_y = event.gyro.y;
  float imu1_gyro_z = event.gyro.z;

  /* magnetic vector values are in micro-Tesla (uT) */
  mag.getEvent(&event);
  float imu1_magn_x = event.magnetic.x;
  float imu1_magn_y = event.magnetic.y;
  float imu1_magn_z = event.magnetic.z;

  /* Printing all values */
  Serial.print(imu1_acc_x);Serial.print(",");
  Serial.print(imu1_acc_y);Serial.print(",");
```

```
  Serial.print(imu1_acc_z);Serial.print(",");

  Serial.print(imu1_gyro_x);Serial.print(",");
  Serial.print(imu1_gyro_y);Serial.print(",");
  Serial.print(imu1_gyro_z);Serial.print(",");

  Serial.print(imu1_magn_x);Serial.print(",");
  Serial.print(imu1_magn_y);Serial.print(",");
  Serial.print(imu1_magn_z);Serial.print(",");

  Serial.println(millis());

}
```

Appendix I

User Manual

I.1 How to run the Boat

To get the boat up and running, you have to first of all install ROS on an Ubuntu-computer. We recommend dual boot because the virtual machine is problematic with networking.

To install ROS Melodic on your computer, follow this guide: http://wiki.ros.org/melodic/Installation/Ubuntu. Choose the Desktop-full install.

When ROS is installed, its time to clone down the repository, use this link: https://github.com/VeslRuben/Luretriks-Dev.git. The repository has some ROS Packets that require some dependencies to build. Run these commands in a terminal to install the dependencies:

```
sudo apt install qt4-default
```

Needed for hector slam to run

```
sudo apt-get install libsdl-image1.2-dev
sudo apt-get install libsdl-dev
```

These are needed for the map server to run

```
sudo apt install libgsl-dev
```

Needed for the laser scan-matcher to run

Now you need to follow this guide by FranekStark to install csm, a dependancy for the laser scan matcher. https://github.com/ccny-ros-pkg/scan_tools/issues/63#issuecomment-486194571, note that this repository should NOT be put in the SRC-folder of the project, but rather be put inside the catkin workspace.

Now all the dependencies of the project should be installed, and its time to catkin_make the project. Open a new terminal with CTRL+ALT+T and change into the project directory using:

```
cd [path to folder]
```

After this you can type

```
catkin_make
```

This will build the entire project, and will take quite some time the first time it is done.

Now everything related to ROS is installed and we are almost done with the installation. The last step is to install an NTP-server. Since the Raspberry Pi 4 does not keep its time when powered off, we need to re-synchronize it (this happens automatically when connected to the internet).

```
sudo apt-get update
sudo apt-get install ntp
```

Use these two to install the NTP-server on your PC. After this it's time to tell the Raspberry that it should use your PC to synchronize its time. First you need to connect to the wifi router on the boat. After this, you have to ssh into the raspberry by typing

```
ssh pi@192.168.0.102
```

into the terminal. The password for the raspberry is "Luretriks". Now type:

```
sudo nano /etc/hosts
```

and change the IP-address of "NTP-server-host" to the IP-address of your machine. This needs to be done every time the Raspberry Pi has been shut down for some amount of time. If the time is not synchronized, ROS will mistake messages as old and disregard them completely.

Now all of the installation is done!

Now some last minute setup that needs to be done when using a new computer. We need to edit the .sh files used for starting up the different parts of the program. First in the terminal you used to ssh into the raspberry, type:

```
cd luretriks-dev
```

From here, type:

```
nano shells/raspberry.sh
```

Here you need to edit the "ROS_MASTER_URI" to the ip of your computer, and the ROS_IP to the ip of the raspberry. Now the raspberry is up to speed, and its only your PC left.

Open a new terminal and "cd" into the project directory. Use nano and edit all the files in the shells directory as we did on the raspberry only here the "ROS_IP" is the IP of your PC.

Now it's soon time to run the damn thing!

The last configuration that has to be done is telling the system if the feedback to the kalman-filter is coming from SLAM or laser scan matcher. The feedback comes from laser scan matcher when running the AMCL-porogram, and SLAM if running the SLAM-program.

From the project directory navigate in to "src/launch/launch" and edit the "run_pc.launch"-file. Comment out the parameter "SLAM_OUT_POSE" and comment in "POSE_STAMPED" if

running AMCL, and the other way if running SLAM.

Now we are all set to run the boat!

First connect your PC to the internet using your phone or something else. Then open a terminal and type:

```
sudo service ntp restart
```

to update your NTP-server. Give it a few seconds and check if all went well using:

```
sudo service ntp status
```

If no errors all is good and you can connect back to the boats network.

Open a terminal and type

```
roscore
```

to start a ROS Master on your PC.

Then you can SSH into the raspberry, then

```
cd luretriks-dev
```

to get into the catkin workspace. Now type:

```
./shells/raspberry.sh
```

This will synch the raspberry's time with your PC and then run the program on the raspberry, this is all that needs to be done on the raspberry.

Now that the raspberry is running what it needs, but your PC is running nothing, it activates its "disconnected"-behaviour and stops all the motors from running.

Time to start up the remote control for the boat from the PC. This script tells the Raspberry that the PC is connected and activates the motors. This makes it possible to manually control the boat if a Wired XBox-controller is connected when the script is started. Open a new terminal, and from the catkin workspace type:

```
./shells/pc.sh
```

This runs the program.

Now the boat can be remotely controlled, but it has no clue where it is in the world, and the kalman filter has no feedback because of this.

Last thing to do is to launch the mapping and navigation to make the robot localize itself and run autonomously. Open a new terminal and from the catkin workspace type:

```
./shells/mapping.sh
```

To run up the slam localization. Or you can type:

```
./shells/amcl.sh
```

To start up the AMCL. Both of the programs fires up the navigation-stack and gives the possibility to make it run autonomously. Note: Remember to change the kalman-filter feedback parameter discussed further up to reflect what you are running.

To run AMCL with a new map you first have to use SLAM to map out the area. Start up SLAM and drive carefully around to get the most accurate map. When you are satisfied with the map SLAM has created, save the map by typing:

```
rosrun map_server map_saver -f /home/[username]/map_name
```

This saves the map in your home folder. To use the map, move the files created (yaml and BMP) in to the maps directory in the launch packet (luretriks-dev/src/launch/maps). Now open the "run-amcl.launch"-file and edit the path to the map parameter under map-server.

I.2 Tips & Tricks

I.2.1 Can't Find Executable File When Trying Run .py or .launch-files

All script-files need to be made executable to be able to run. This is done on all files used in the project, but if for some reason a file lacks this or you add a new one you need to make it executable by typing:

```
sudo chmod +x [path/to/file]
```

I.2.2 Error about sensor readings being from the past

When the Raspberry Pi-script is started, pay attention to the first few lines to make sure the time gets synchronized with the PC. If this fails, everything will look normal at the start, but SLAM and AMCL will not work since they think all the sensor messages are very old.

I.2.3 OpenCv wrong version

If you encounter an error with openCV being the wrong version, heres the solution:

In the folder vision_opencv in luretriks-dev there's a shell-file for installing opencv 3. Run this if catkin_make complains about the wrong version of opencv.

Navigate to this folder, then type:

```
./cv.sh
```

Other than that you can contact your supervisor to contact us if there is anything specific.