CHALMERS
UNIVERSITY OF TECHNOLOGY


# Waypoint-based path following system for a jet ski

*Development, simulation and implementation of a path following control system for an autonomous driving jet ski that follows a lead boat*

Master's thesis in Systems, Control & Mechatronics

*In cooperation with the Swedish Sea Rescue Society*

Anton Bergholtz
Carl-Adam Hernvall

SJÖRÄDDNINGS SÄLLSKAPET

Department of Signals & Systems
Chalmers University of Technology
Gothenburg, Sweden 2015

Master's thesis EX024/2015

# Waypoint-based path following system for a jet ski

Development, simulation and implementation of a path following control system for an autonomous driving jet ski that follows a lead boat

Anton Bergholtz
Carl-Adam Hernvall


CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Signals & Systems
*Automation research group*
Chalmers University of Technology
Gothenburg, Sweden 2015

Waypoint based path following for a jet ski
Development, simulation and implementation of a path following control system for an autonomous driving jet ski that follows a lead boat

ANTON BERGHOLTZ
CARL-ADAM HERNVALL



Supervisor: Fredrik Falkman, Swedish Sea Rescue Society
Examiner: Petter Falkman, Department of Signals & Systems

Master's Thesis EX024/2015
Department of Signals & Systems
Automation Research Group
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: The cover shows how the Rescuerunner sends information via Wi-Fi and uses UDP protocol. The information is sent to the user program where reference throttle and steering are calculated and sent back to the Rescuerunner.

Printed by Reproservice Gothenburg, Sweden 2015

**Abstract**

This thesis has been focusing on developing and implementing a path following control system for an autonomous driving jet ski, making it able to follow the path of a lead boat. This has been achieved by deriving a mathematical model and describing the motion of the jet ski in a water environment in 6 degrees of freedom. In order to get ideas on how the actual implementation was going to be performed, the mathematical model was included as a part of a simulation model, in which the behavior of the jet ski following a predefined path consisting of a number of waypoints was simulated. Each waypoint was given a certain radius of acceptance and once the jet ski was within this tolerance it started heading towards the next waypoint. The heading and throttle of the jet ski was regulated using two PID-controllers. For the actual implementation a Rescuerunner, which is a kind of jet ski owned by the Swedish Sea Rescue Society, was used. The Rescuerunner was rebuilt with an Arduino microcontroller, that enabled the steering and throttle to be controlled digitally from a graphical user interface. The Arduino was also equipped with several shields that allowed it to connect to a Wi-Fi network and a GPS-device. Via the network, the Rescuerunner was allowed to send and receive GPS-positions from a similar Arduino microcontroller that was installed on the lead boat. As done in the simulation model, the GPS-positions that was dropped from the lead boat, were assigned a certain radius of acceptance and once the Rescuerunner was within this radius it starts aiming at the next waypoint. The path following control system has been tested with good results. It shows great tendencies on working as desired, although more testing and evaluation needs to be performed.

**Keywords: Path following, simulation, mathematical model, automatic control, sensor fusion, control engineering, UDP communication, microcontroller, Arduino, IMU, tilt compensated digital compass, autonomous jet ski, Rescuerunner, $I^2C$ communication, graphical user interface**

**Acknowledgements**

First of all, we would like to thank our supervisor and examiner Petter Falkman for his counselling. He arranged valuable meetings and supported us throughout the whole project. The meetings gave us new angles of incidence on how to approach upcoming and unpredictable problems that arose as the work went on. We would also like to thank our supervisor Fredrik Falkman at the Swedish Sea Rescue Society for the input he gave us regarding the desired behavior of the Rescuerunner, facilitating the implementation work a lot. Consecutively we would like to send a special thanks to Adam Andersson, Viktor Bäckman, Oscar Pantzare, Anders Sarvik and Oscar Granqvist who worked beside us on a similar project. The cooperation with them gave us a lot of valuable exchange. Finally, we would also like to send our thanks to Bo Egardt who took the time to help us in the earlier stages of the project, guiding us in the right direction.

Anton Bergholtz and Carl-Adam Hernvall,
Göteborg, June 2015

# Contents

# Glossary

**Arduino** The Arduino Mega is a microcontroller board.

**Boat fixed** Coordination system relative to the Rescuerunner.

**DoF** Degrees of Freedom is a way to describe translations and rotations of a body around a number of axes in a coordinate system.

**Earth fixed** Coordination system relative to earth.

**EB** The main vessel leading the platoon of the two vessels.

**GPS** A satellite navigation system that provides location and time information anywhere on or near the earth where there is an unobstructed line to four or more GPS-satellites.

**GUI** The graphical user interface is a program where the user can choose between different modes depending on which kind of application and behavior the user want to apply.

**IMU** An electronic device that measures a vessels velocity, gravitational force and heading by using a combination of an accelerometer, a gyroscope and in some cases a magnetometer.

**LOS** A term used to describe the straight line connecting two consecutive points.

**MATLAB** A programming language developed by MathWorks Inc., originally released in 1984.

**PID** A controller with a **P**roportional, an **I**ntegral and a **D**erivative term.

**RoA** The radius of the circle enclosing a waypoint.

**RR** A jet ski vessel used and developed by the Swedish sea rescue society.

**Simulink** A graphical programming envioronment for modeling and simulating multi-dynamic systems.

**SSRS** A voluntary organization that was started 1907 and since rescued over 25 000 ships and 70 000 people in need at sea.

**UDP** A data communication protocol that was designed by David P. Reed in 1980.

**WP** A latitude and longitude coordinate.

# Acronyms

**BF** Body fixed.

**DoF** Degrees of Freedom.

**EB** Ego boat.

**EF** Earth fixed.

**GND** Ground.

**GPS** Global Positioning System.

**GUI** Graphical User Interface.

**$I^2C$** Inter-Integrated Circuit.

**IMU** Inertial Measurement Unit.

**LOS** Line of Sight.

**LSB** Least significant bit.

**RoA** Radius of acceptance.

**RR** Rescuerunner.

**SCL** Serial Clock Line.

**SDA** Serial Data Line.

**SSRS** Swedish Sea Rescue Society.

**UDP** User Datagram Protocol.

**VCC** Voltage at the common collector.

**WP** Waypoint.

# List of Figures

# List of Tables

# 1

# Introduction

The Swedish Sea Rescue Society (SSRS) is a voluntary organization that was started 1907 and has since then rescued over 25 000 ships and 70 000 people in need at sea. In the past years SSRS has been growing rapidly and they currently consist of 68 rescue stations, 190 rescue boats and approximately 2000 voluntary lifeguards along the Swedish coast[1].

Larger rescue boats are good at making their way in rough seas. Once reaching the scene of the accident the size of these rescue boats is a problem. Because of their size they appeared to be inefficient when it comes to browsing accidental scenes where the water is shallow. In order to solve this problem the SSRS developed the Rescuerunner (RR), a small and flexible jet ski, which has now been situated at most of their rescue stations. Once this problem was solved a new one arose: it was complicated to adapt the larger rescue boats to carry the smaller RRs to the scene of the accident. This has led to an infrequent use of the RR. Currently the RRs are mostly being used for education and training operations in the vicinity of the rescue stations.

One solution to the problem stated above is to introduce an automated RR[2]. The purpose of implementing an autonomous RR is to handle situations where riding the RR is unwanted, e.g. when transporting the RR through rough sea to the scene of accident. Because of that, the goal was to create an application making the RR follow another larger boat (hereinafter referred to as an Ego Boat (EB)) to a certain position where it is

more needed. Such an invention will hopefully be of great help and make today's rescue missions even more safe and effective. The fact that the RR will be available at the scene of the accident, without exhausting the crew, will save time, which very often can correlate with saving lives. Another benefit is that there is no need to make adoptions for launch and recovery to existing larger boats, which means that the functionality and the performance is not compromised. When the RR is decoupled from the EB, the risk of hassles and extra cognitive effort of towing is minimized.

## 1.1 Context

This project is a cooperation between the SSRS and the department of Signals & Systems at Chalmers University of Technology. The idea was to make the most of the practical knowledge of how the RR is used today. Identify its pros and cons and further discuss how to make the new application as efficient as possible. Combining this practical knowledge from SSRS with the technical knowledge from Chalmers to create and develop an autonomous RR. A path following algorithm and a control system, enabling the RR to follow an EB are then developed within the framework of this cooperation. This project resulted in a master's thesis, which was carried out by the authors during the first half of 2015.

## 1.2 General objectives

The general objectives of this thesis are

1. Deriving a mathematical model for the RR.

2. Developing a simulation environment for the RR, in which it is able to autonomously follow a path, consisting of waypoints (WP) fed to its control system.

3. Development of a communication system which enables wireless data transfer between the RR and EB.

4. Assess the results gathered from the simulation and use them to redesign the RR for automatic navigation.

## 1.3 Contributions

The main contributions of this thesis are listed below.

- Development of a mathematical model for a RR that can be used in simulations for testing its behavior in path following applications.
- Development of a decision making algorithm, making it able for the system to determine what to do when a WP is missed.
- Derivation of a tilt compensated heading angle
- Development of a control system for the EB-RR system using microcontrollers.
- Development of a communication system for the EB-RR system using microcontrollers.

## 1.4 Constraints

To be able to complete this thesis within the limited time frame certain things had to be neglected. Therefore the simulation model is constructed without environmental impacts such as winds, waves and drift taken into account. The path following system is only be tested for low velocities and neither have laws regarding unmanned vehicles at sea been taken into consideration. In order not to violate any sea traffic regulations, there was always someone on board the RR being able to stop the engine if something unforeseen would happen, when the path following control system was tested. On the contrary to the simulation model the actual implementation of the path following system used during the empirical tests only allows 3 Degrees of Freedom (DoF) to be controlled.

## 1.5 Thesis organisation

The report starts with describing some prerequisites regarding modeling of ocean vehicles, consecutively the approach of deriving a mathematical model for the RR is thoroughly described, see Flow diagram 1.1. It is also mentioned what simplifications are made and what is being neglected in the model and for what reasons. This chapter is followed by a simulations chapter, carefully describing how the mathematical model of the RR is implemented in a simulation environment. The chapter is also, among other

things, describing how the simulations are carried out, how the path following algorithm is developed and how the PID-controllers for heading and speed are implemented. The simulations chapter is followed by a chapter describing how the simulation environment is translated for use in a real system, using Arduino Mega microcontrollers. This chapter also includes sensor fusion algorithm and filtering, e.g. how the yaw angle of the RR is calculated by combining an accelerometer, a gyroscope and a magnetometer. A result chapter follows this chapter, where the results from the simulation and the result from the tests performed on the actual application are presented. Finally the results from this thesis work are discussed, conclusions are drawn and proposal for future work are mentioned.

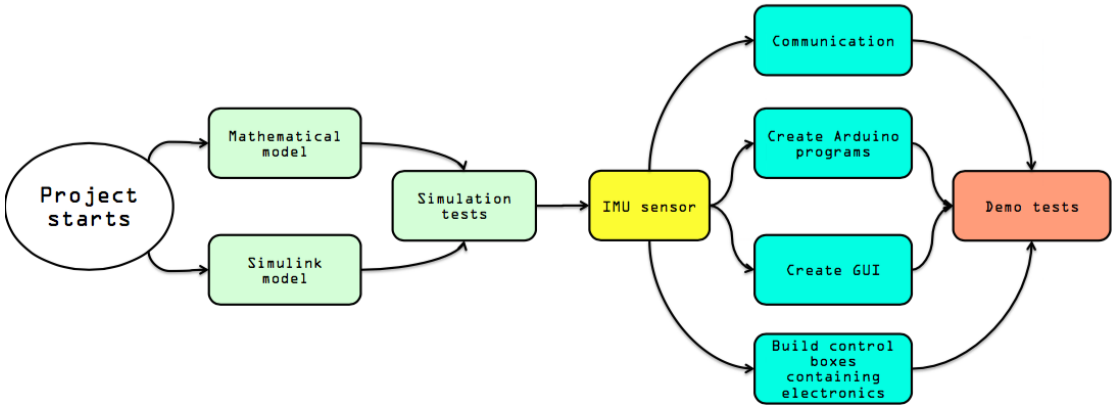



**Diagram 1.1:** Flow diagram is showing an overview how the work of this thesis will be carried out.

# 2

# Modeling

In this chapter a mathematical model of the RR is presented. First, the kinematics of the RR is demonstrated, and how it is positioned and orientated with respect to the Body Fixed (BF) coordinate system. Further, the nonlinear dynamic equation of motion is stated followed by the rigid body dynamics. In this chapter a state space model in 6 DoF is derived. For simulation purposes however, the state space model is simplified to 3 DoF.

## 2.1 Kinematics

To be able to describe the position and orientation of the RR as a rigid body, it is necessary to use six independent coordinates, see Tab. 2.1, where the RR can be described in 6 DoF. Note, further on, vectors will be notated with **bold** font.

### 2.1.1 Coordinate systems

The motion of the RR is based on a 6 DoF model, and uses two coordinate systems, one EF (Earth fixed) and one Boat fixed[3], see Fig. 2.1. The BF coordinate axes are defined as $X_{BF}$ (longitudinal axis), $Y_{BF}$ (transverse axis) and $Z_{BF}$ (normal axis). The EF coordinate axes are defined as $X_{EF}$, $Y_{EF}$ and $Z_{EF}$. The RRs position and orientation will be expressed relative to the BF frame and described by the vector $\boldsymbol{\nu}$, also called the *Euler rate vector*. The linear and angular velocity vector $\boldsymbol{\eta}$ is formulated from the

**Table 2.1:** Notation of variables describing the RR in a 6 DoF environment

| | **6 DoF** | | **Forces and moments** | **Linear and angular vel.** | **Position and Euler angles** |
|---|---|---|---|---|---|
| 1. | Motions in the x-direction | (surge) | X | u | x |
| 2. | Motions in the y-direction | (sway) | Y | v | y |
| 3. | Motions in the z-direction | (heave) | Z | w | z |
| 4. | Rotations around the x-axis | (roll) | K | p | $\phi$ |
| 5. | Rotations around the y-axis | (pitch) | M | q | $\theta$ |
| 6. | Rotations around the z-axis | (yaw) | N | r | $\psi$ |

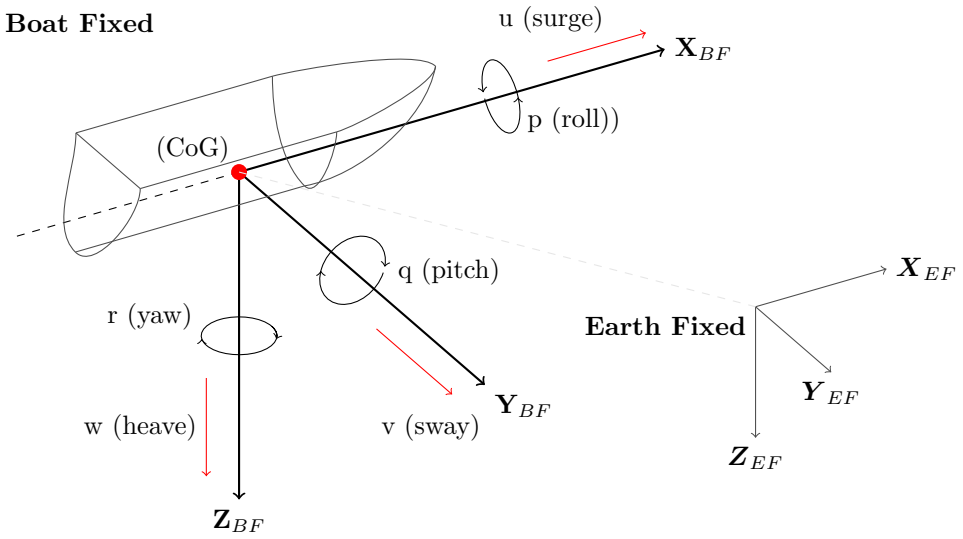



**Figure 2.1:** Illustration of motions and rotations affecting the RR in 6 DoF. The motions as well as the rotations are described along the $x-$, $y-$ and $z-$axis respectively.

BF coordinate system. The forces acting on the RR in its three dimensions is referred to as $\boldsymbol{\tau}_1$. The same goes for the moments, where $\boldsymbol{\tau}_2$ will describe the moments around the three axes. Together they will form the forces and moments vector $\boldsymbol{\tau}$. The following notations will therefore be used throughout this thesis, see Tab. 2.2.

**Table 2.2:** Vector notations of the RR in 6 DoF

| | | |
|---|---|---|
| $\boldsymbol{\eta} = [\boldsymbol{\eta}_1^\top, \boldsymbol{\eta}_2^\top]^\top$, | $\boldsymbol{\eta}_1 = [x,y,z]^\top$, | $\boldsymbol{\eta}_2 = [\phi,\theta,\psi]^\top$, |
| $\boldsymbol{\nu} = [\boldsymbol{\nu}_1^\top, \boldsymbol{\nu}_2^\top]^\top$, | $\boldsymbol{\nu}_1 = [u,v,w]^\top$, | $\boldsymbol{\nu}_2 = [p,q,r]^\top$, |
| $\boldsymbol{\tau} = [\boldsymbol{\tau}_1^\top, \boldsymbol{\tau}_2^\top]^\top$ | $\boldsymbol{\tau}_1 = [X,Y,Z]$ | $\boldsymbol{\tau}_1 = [K,M,N]$. |

### 2.1.2 Coordinate transformation

In order to describe the position and movement of the RR relative to the earth-fixed coordinate system a velocity transformation can be used. According to [3] the following expression for this transformation can be derived

$$\dot{\boldsymbol{\eta}_1} = \mathbf{J}_1(\boldsymbol{\eta_2})\boldsymbol{\nu_1}. \tag{2.1}$$

Here $\mathbf{J}_1(\boldsymbol{\eta_2})$ is a transformation matrix for expressing to so called *Euler angles*, i.e. roll ($\phi$) , pitch ($\theta$) and yaw ($\psi$),. With the help of *"Euler's Theorem of Rotation"* as described in [3], the rotation matrices can be derived as

$$\boldsymbol{R}_x(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & c(\phi) & s(\phi) \\ 0 & -s(\phi) & c(\phi) \end{bmatrix}, \boldsymbol{R}_y(\theta) = \begin{bmatrix} c(\theta) & 0 & -s(\theta) \\ 0 & 1 & 0 \\ s(\theta) & 0 & c(\theta) \end{bmatrix}, \boldsymbol{R}_z(\psi) = \begin{bmatrix} c(\psi) & s(\psi) & 0 \\ -s(\psi) & c(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.2}$$

where $s(\cdot) = \sin(\cdot)$ and $c(\cdot) = \cos(\cdot)$ and the three rotation matrices $R_x, R_y$ and $R_z$ about the rotation angle $\phi, \theta$ and $\psi$ respectively.

### 2.1.3 Linear Velocity Transformation

The transformation matrix is described by three rotations matrices and can be written as

$$\boldsymbol{J_1}(\boldsymbol{\eta_2}) = \boldsymbol{R}_x^\top(\phi)\boldsymbol{R}_y^\top(\theta)\boldsymbol{R}_z^\top(\psi). \tag{2.3}$$

It is important to have in mind that these rotations are not arbitrary, but are often set according to the $xyz$-convention in control applications specified in terms of the Euler

angles as described in section 2.1.2. So will be done also in this thesis. From what is written in 2.3 the inverse transformation is possible to express as

$$\boldsymbol{J_1}^{-1}(\boldsymbol{\eta_2}) = \boldsymbol{J_1}^{\top}(\boldsymbol{\eta_2}) = \boldsymbol{R}_x(\phi)\boldsymbol{R}_y(\theta)\boldsymbol{R}_z(\psi). \tag{2.4}$$

Subsequently expanding the linear transformation matrix results in the following expression

$$\boldsymbol{J_1}(\boldsymbol{\eta_2}) = \begin{bmatrix} c(\theta)c(\psi) & c(\psi)s(\theta)s(\phi) - c(\phi)s(\psi) & c(\phi)c(\psi)s(\theta) + s(\phi)s(\psi) \\ c(\theta)s(\psi) & c(\phi)c(\psi) + s(\theta)s(\phi)s(\psi) & c(\phi)s(\theta)s(\psi) - c(\psi)s(\phi) \\ -s(\theta) & c(\theta)s(\phi) & c(\theta)c(\phi) \end{bmatrix}. \tag{2.5}$$

In Fig. 2.2-2.4 it is described what happens with the roll, pitch and yaw when rotating them around the $z-$,$y-$ and $x-$axis respectively. When considering rotation around the $z-$axis the new coordinate system $X_{BF,3}Y_{BF,3}Z_{BF,3}$ arises[3]. This rotation describes the behavior of the RR when rotating the yaw angle $\psi$ about the $z-$axis (Fig. 2.2). The same goes for the coordinate system $X_{BF,2}Y_{BF,2}Z_{BF,2}$ when rotating the pitch angle $\theta$ about the $y-$axis (Fig. 2.3), and the coordinate system $X_{BF,1}Y_{BF,1}Z_{BF,1}$ when rotating the roll angle $\phi$ about the $x-$axis (Fig. 2.4).

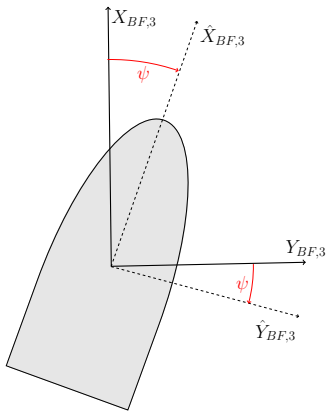



**Figure 2.2:** Rotation over the heading angle $\psi$ about the $z$-axis.

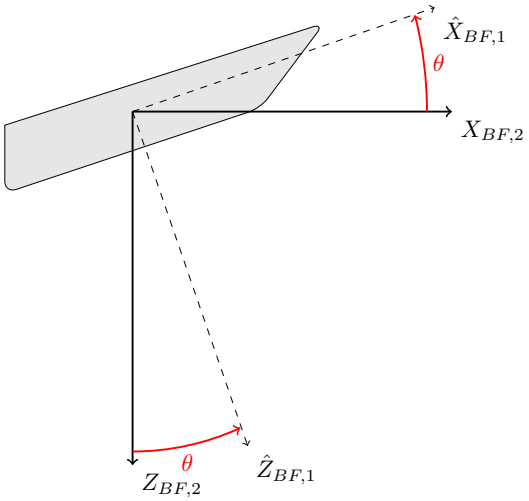



**Figure 2.3:** Rotation over the pitch angle $\theta$ about the $y$-axis.

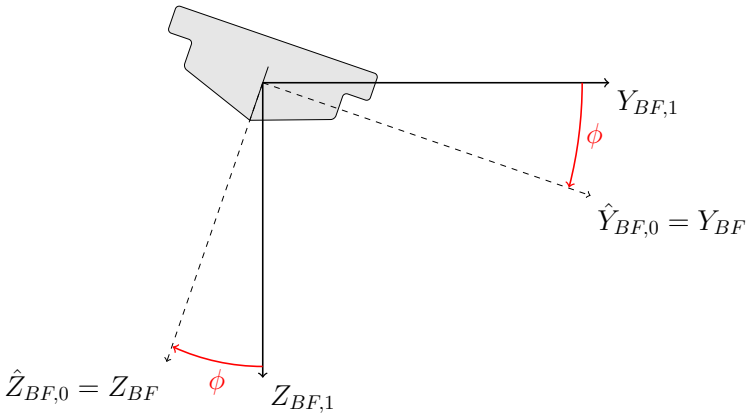



**Figure 2.4:** Rotation over the heading angle $\phi$ about the $x$-axis.

### 2.1.4 Angular Velocity Transformation

The same approach is used as in section 2.1.3 when calculating the transformation matrix $\boldsymbol{J_2}(\eta_2)$, now with the angular velocity vector $\boldsymbol{\nu}_2 = [p,q,r]^\top$ and the Euler rate vector $\boldsymbol{\eta}_2 = [\phi,\theta,\psi]^\top$. The orientation of the RR can be described by the BF reference frame with respect to the BF frame, see Eq. 2.6.

$$\boldsymbol{\nu_2} = \begin{bmatrix} \dot{\phi} \\ 0 \\ 0 \end{bmatrix} + \boldsymbol{R}_x(\phi) \begin{bmatrix} 0 \\ \dot{\theta} \\ 0 \end{bmatrix} + \boldsymbol{R}_x(\phi)\boldsymbol{R}_y(\theta) \begin{bmatrix} 0 \\ 0 \\ \dot{\psi} \end{bmatrix} = \boldsymbol{J}_2^{-1}(\boldsymbol{\eta})\dot{\boldsymbol{\eta}}. \tag{2.6}$$

The Eq. 2.6 can be expanded in terms of $\eta_2$ by

$$\dot{\boldsymbol{\eta}}_{\mathbf{2}} = \boldsymbol{J_2}(\eta_2)\boldsymbol{\nu}, \tag{2.7}$$

with the corresponding inverse transformation matrix in Eq. 2.8.

$$\boldsymbol{J}_2^{-1}(\eta_2) = \begin{bmatrix} 1 & 0 & -s(\theta) \\ 0 & c(\theta) & c(\theta)s(\psi) \\ 0 & -s(\phi) & c(\theta)c(\phi) \end{bmatrix} \Rightarrow \boldsymbol{J}_2(\eta_2) \begin{bmatrix} 1 & s(\phi)c(\theta) & c(\phi)t(\theta) \\ 0 & c(\phi) & -s(\phi) \\ 0 & s(\phi)/c(\theta) & c(\phi)c(\theta) \end{bmatrix}, \tag{2.8}$$

where $s(\cdot) = \sin(\cdot)$, $c(\cdot) = \cos(\cdot)$ and $t(\cdot) = \tan(\cdot)$. With all components derived the overall kinematics can be describe by the vector in Eq. 2.9.

$$\begin{bmatrix} \dot{\boldsymbol{\eta}}_{\mathbf{1}} \\ \dot{\boldsymbol{\eta}}_{\mathbf{2}} \end{bmatrix} = \begin{bmatrix} \boldsymbol{J}_1(\boldsymbol{\eta}_2) & \mathbf{0}_{3X3} \\ \mathbf{0}_{3X3} & \boldsymbol{J}_2(\boldsymbol{\eta}_2) \end{bmatrix} \begin{bmatrix} \boldsymbol{\nu}_1 \\ \boldsymbol{\nu}_2 \end{bmatrix} \Rightarrow \dot{\boldsymbol{\eta}} = \boldsymbol{J}(\boldsymbol{\eta})\boldsymbol{\nu}. \tag{2.9}$$

## 2.2 Nonlinear dynamic equation of motion

To be able to describe the behavior of the RR in a water environment the equations of motion, described by Fossen [3] is a powerful tool. The Eq. 2.10 describes the 6 DoF nonlinear dynamic equation of motion with matrix explanations in Tab. 2.3

$$\boldsymbol{M}\dot{\boldsymbol{\nu}} + \boldsymbol{C}(\boldsymbol{\nu})\boldsymbol{\nu} + \boldsymbol{D}(\boldsymbol{\nu})\boldsymbol{\nu} + \boldsymbol{g}(\boldsymbol{\eta}) = \boldsymbol{\tau}, \tag{2.10}$$

## 2.3 Rigid body dynamics

In the following section it will be described how the rigid body parameters from Eq. 2.10 are calculated. The first part will show how to calculate the inertia matrix $\boldsymbol{M}_{RB}$ followed by the Coriolis and centripetal terms.

**Table 2.3:** Matrix notations of the nonlinear dynamic equations.

| | | |
|---|---|---|
| $\boldsymbol{M}$ | $= \boldsymbol{M_{RB}} + \boldsymbol{M_A}$ | : Inertia matrix for a rigid body, including added mass |
| $\boldsymbol{C}(\boldsymbol{\nu})$ | $= \boldsymbol{C_{RB}}(\boldsymbol{\nu}) + \boldsymbol{C_A}(\boldsymbol{\nu})$ | : Coriolis and centripetal forces, including added mass |
| $\boldsymbol{D}(\boldsymbol{\nu})$ | | : Hydrodynamic damping matrix |
| $\boldsymbol{g}(\boldsymbol{\eta})$ | | : Gravitational forces and moments |
| $\boldsymbol{\tau}$ | | : Control inputs as forces and moments |

### 2.3.1 The inertia matrix

As mentioned by Krammer[4] the inertia matrix looks as follows

$$\boldsymbol{M}_{RB} = \begin{bmatrix} m\boldsymbol{I}_{3\times 3} & 0_{3\times 3} \\ 0_{3\times 3} & \boldsymbol{I}_0 \end{bmatrix}, \tag{2.11}$$

where $\boldsymbol{I}_0$ is the inertia tensor with respect to the origin, i.e.

$$\boldsymbol{I}_0 = \begin{bmatrix} I_{xx} & -I_{xy} & -I_{xz} \\ -I_{yx} & I_{yy} & -I_{yz} \\ -I_{zx} & -I_{zy} & I_{zz} \end{bmatrix}. \tag{2.12}$$

Here, $I_{xy} = I_{yx} = I_{xz} = I_{zx} = I_{yz} = I_{zy} = 0$[4], where the moments from the inertia about centroidal axes $x$,$y$ and $z$ can be written as

$$I_{xx} = (\frac{1}{2} - \frac{1}{2} - \frac{16}{9\pi^2}mr^2), \tag{2.13}$$

$$I_{yy} = (\frac{1}{4} - \frac{16}{9\pi^2}mr^2) + \frac{1}{12}mL^2, \tag{2.14}$$

$$I_{zz} = \frac{1}{4}mr^2 + \frac{1}{12}mL^2. \tag{2.15}$$

### 2.3.2 Coriolis and centripetal terms

The matrix describing the Coriolis and centripetal forces can be written as below

$$\boldsymbol{C}_{RB}(\boldsymbol{\nu}) = \begin{bmatrix} m\boldsymbol{S}(\boldsymbol{\nu}_2) & -m\boldsymbol{S}(\boldsymbol{\nu}_2)\boldsymbol{S}(\boldsymbol{r}_G) \\ m\boldsymbol{S}(\boldsymbol{r}_G)\boldsymbol{S}(\boldsymbol{\nu}_2) & -\boldsymbol{S}(\boldsymbol{I}_0\boldsymbol{\nu}_2) \end{bmatrix}, \tag{2.16}$$

where $\boldsymbol{r}_G$ is the distance from the origin to the center of gravity and $\boldsymbol{S}$ is Skew-Symmetric (asymmetric) matrix, e.g.

$$\boldsymbol{S}(\lambda) = \begin{bmatrix} 0 & -\lambda_3 & \lambda_2 \\ \lambda_3 & 0 & -\lambda_1 \\ -\lambda_2 & \lambda_1 & 0 \end{bmatrix}, \quad \boldsymbol{S}(\boldsymbol{\nu}_2) = \begin{bmatrix} 0 & -r & q \\ r & 0 & -p \\ -q & p & 0 \end{bmatrix}. \tag{2.17}$$

Since the origin of the BF coordinate system is placed in way which makes it coincide with the RRs center of gravity $\boldsymbol{r}_G = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}^\top$. From this it is obvious that $\boldsymbol{S}(\boldsymbol{r}_G) = 0_{3\times 3}$, why the following expression for $\boldsymbol{C}_{RB}(\boldsymbol{\nu})$ can be written as

$$\boldsymbol{C}_{RB}(\boldsymbol{\nu}) = \begin{bmatrix} 0 & -mp & mq & 0 & 0 & 0 \\ mr & 0 & -mp & 0 & 0 & 0 \\ -mq & mp & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -pI_{xx} + qI_{xy} + rI_{xz} & -pI_{yx} + qI_{yy} - rI_{yz} \\ 0 & 0 & 0 & -pI_{zx} - qI_{zy} + rI_{zz} & 0 & -pI_{xx} + qI_{xy} + rI_{xz} \\ 0 & 0 & 0 & pI_{yx} - qI_{yy} + rI_{yz} & pI_{xx} - qI_{xy} - rI_{xz} & 0 \end{bmatrix}. \tag{2.18}$$

## 2.4 Hydrodynamic forces and moments

Since the inertia of the surrounding fluid is affecting the RR the so called added mass effect [3] has to be taken into account. This matrix is denoted as $\boldsymbol{M}_A$ and can be seen in Eq. 2.19.

$$\boldsymbol{M}_A = -\begin{bmatrix} X_{\dot{u}} & X_{\dot{v}} & X_{\dot{w}} & X_{\dot{p}} & X_{\dot{q}} & X_{\dot{r}} \\ Y_{\dot{u}} & Y_{\dot{v}} & Y_{\dot{w}} & Y_{\dot{p}} & Y_{\dot{q}} & Y_{\dot{r}} \\ Z_{\dot{u}} & Z_{\dot{v}} & Z_{\dot{w}} & Z_{\dot{p}} & Z_{\dot{q}} & Z_{\dot{r}} \\ K_{\dot{u}} & K_{\dot{v}} & K_{\dot{w}} & K_{\dot{p}} & K_{\dot{q}} & K_{\dot{r}} \\ M_{\dot{u}} & M_{\dot{v}} & M_{\dot{w}} & M_{\dot{p}} & M_{\dot{q}} & M_{\dot{r}} \\ N_{\dot{u}} & N_{\dot{v}} & N_{\dot{w}} & N_{\dot{p}} & N_{\dot{q}} & N_{\dot{r}} \end{bmatrix}, \tag{2.19}$$

but since the RR is symmetrical about the $x_b z_b$-plane. $\boldsymbol{M_A}$ can be assumed symmetric, therefore $\boldsymbol{M}_{A_{ij}} = \boldsymbol{M}_{A_{ji}}$. As the off-diagonal terms in $\boldsymbol{M}_A$ and $\boldsymbol{M}_{RB}$ are small compared to the diagonal elements, they could be neglected for further simplification, see Eq. 2.20.

$$\boldsymbol{M}_A = -diag(X_{\dot{u}},Y_{\dot{v}},Z_{\dot{w}},K_{\dot{p}},M_{\dot{q}},N_{\dot{r}}), \tag{2.20}$$

where the parameters in $\boldsymbol{M}_A$ are calculated using strip theory, see Section 2.4.1.

As for the inertia matrix also the Coriolis and centripetal matrix is affected by the movement of the fluid it lays in. Therefor a Coriolis and centripetal matrix $\boldsymbol{C}_A(\boldsymbol{\nu})$ for the added mass must be included, see Fig. 2.21.

$$\boldsymbol{C}_A(\boldsymbol{\nu}) = \begin{bmatrix} \boldsymbol{0}_{3\times3} & -\boldsymbol{S}(\boldsymbol{A}_{11}\boldsymbol{\nu}_1 + \boldsymbol{A}_{12}\boldsymbol{\nu}_2) \\ -\boldsymbol{S}(\boldsymbol{A}_{11}\boldsymbol{\nu}_1 + \boldsymbol{A}_{12}\boldsymbol{\nu}_2) & -\boldsymbol{S}(\boldsymbol{A}_{21}\boldsymbol{\nu}_1 + \boldsymbol{A}_{22}\boldsymbol{\nu}_2) \end{bmatrix}, \tag{2.21}$$

where $\boldsymbol{A}_{ij}, i \in [1,2]$ and $j \in [1,2]$ are the coefficients that constitutes the added mass inertia matrix, i.e. Eq. 2.22

$$\boldsymbol{M}_A = \begin{bmatrix} \boldsymbol{A}_{11} & \boldsymbol{A}_{12} \\ \boldsymbol{A}_{21} & \boldsymbol{A}_{22} \end{bmatrix}, \tag{2.22}$$

where $\boldsymbol{M}_A$ also hold the equality from Eq. 2.20. Thus of this it can be concluded that $\boldsymbol{A}_{12} = \boldsymbol{A}_{21} = \boldsymbol{0}_{3\times3}$ and $\boldsymbol{A}_{11}$ and $\boldsymbol{A}_{22}$ form the following expression

$$\boldsymbol{A}_{11} = \begin{bmatrix} X_{\dot{u}} & 0 & 0 \\ 0 & Y_{\dot{v}} & 0 \\ 0 & 0 & Z_{\dot{w}} \end{bmatrix}, \qquad \boldsymbol{A}_{22} = \begin{bmatrix} K_{\dot{p}} & 0 & 0 \\ 0 & M_{\dot{q}} & 0 \\ 0 & 0 & N_{\dot{r}} \end{bmatrix},$$

and with this known the final expression for $\boldsymbol{C}_A(\boldsymbol{\nu})$ can be seen in Eq. 2.23.

$$\boldsymbol{C}_A(\boldsymbol{\nu}) = \begin{bmatrix} 0 & 0 & 0 & 0 & -wZ_{\dot{w}} & vY_{\dot{v}} \\ 0 & 0 & 0 & wZ_{\dot{w}} & 0 & -uX_{\dot{u}} \\ 0 & 0 & 0 & -vY_{\dot{v}} & uX_{\dot{u}} & 0 \\ 0 & -wZ_{\dot{w}} & vY_{\dot{v}} & 0 & -rN_{\dot{r}} & qM_{\dot{q}} \\ wZ_{\dot{w}} & 0 & -uX_{\dot{u}} & rN_{\dot{r}} & 0 & -pK_{\dot{p}} \\ -vY_{\dot{v}} & uX_{\dot{u}} & 0 & -qM_{\dot{q}} & pK_{\dot{p}} & 0 \end{bmatrix}. \tag{2.23}$$

Finally, the expression for $\boldsymbol{C}(\boldsymbol{\nu})$ can be expressed in Eq. 2.24.

$$\begin{aligned} \boldsymbol{C}(\boldsymbol{\nu}) &= \boldsymbol{C}_{RB}(\boldsymbol{\nu}) + \boldsymbol{C}_A(\boldsymbol{\nu}) \\ &= \begin{bmatrix} 0 & -mp & mq & 0 & -wZ_{\dot{w}} & vY_{\dot{v}} \\ mr & 0 & -mp & wZ_{\dot{w}} & 0 & -uX_{\dot{u}} \\ -mq & mp & 0 & -vY_{\dot{v}} & uX_{\dot{u}} & 0 \\ 0 & -wZ_{\dot{w}} & vY_{\dot{v}} & 0 & -pI_{xx} + qI_{xy} + rI_{xz} - rN_{\dot{r}} & -pi_{yx} + qI_{yy} - rI_{yz} + qM_{\dot{q}} \\ wZ_{\dot{w}} & 0 & -uX_{\dot{u}} & -pI_{zx} - qI_{zy} + rI_{zz} + rN_{\dot{r}} & 0 & -pI_{xx} + qI_{xy} + rI_{xz} - pK_{\dot{p}} \\ -vY_{\dot{v}} & uX_{\dot{u}} & 0 & pI_{yx} - qI_{yy} + rI_{yz} - qM_{\dot{q}} & pI_{xx} - qI_{xy} - rI_{xz} + pK_{\dot{p}} & 0. \end{bmatrix}. \end{aligned} \tag{2.24}$$

### 2.4.1 Strip Theory

To derive and estimate the hydrodynamic derivatives the *strip theory* is applied [3]. The procedure is to describe the submerged part of the RR as a two dimensional strip. The two-dimensional coefficients for the added mass, that the strip corresponds to, can then be summarized over the length of the body of the RR in order to get the three dimensional coefficients see Fig. 2.6. The three-dimensional coefficients are obtained by integrating the two-dimensional coefficients with respect to each length in the geometry. The hydrodynamic coefficients are now calculated with the assumption that the cross-section of the submerged part of the RR has the shape of a half-ellipse, see Fig. 2.5. For the RR that also has a part of its geometry above the sea level certain parameters can be approximated accordingly to

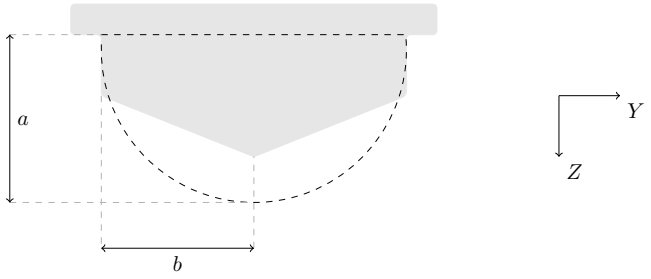



**Figure 2.5:** Illustration on how shape of the hull is approximated. The grey area illustrates the hull of the RR and the dashed area is the simplified half-ellipse area used as an approximation. Here, $a$ denotes the submerged depth of the hull and $2b$ is the total width of the hull at the water line.

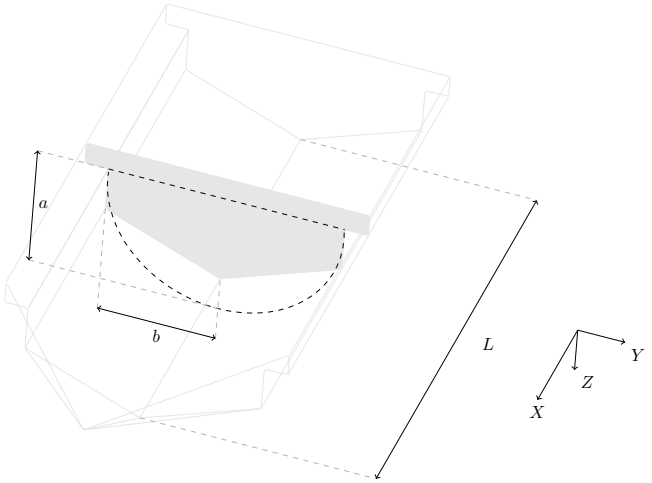



**Figure 2.6:** Illustration on how the hull is divided into strips. The grey are illustrates how the submerged part of the hull of the RR is divided into a number of strips. The grey plane represents one strip. $L$ is the length of the hull.

$$A_{22}^{2D} = \frac{1}{2}\rho\pi a^2, \tag{2.25}$$

$$A_{33}^{2D} = \frac{1}{2}\rho\pi b^2, \tag{2.26}$$

$$A_{44}^{2D} = \frac{1}{16}\pi\rho\left(b^2 - a^2\right)^2. \tag{2.27}$$

According to [3] the three dimensional coefficient for the added mass can be approximated and put together accordingly to Eq. 2.28-2.33

$$A_{11} = -X_{\dot{u}} = 0.05m. \tag{2.28}$$

With this known all the coefficients for the added mass can be calculated as

$$A_{22} = -Y_{\dot{v}} = \int_{-L/2}^{L/2} A_{22}^{2D}(y,z)dx = \frac{1}{2}L\rho a^2, \tag{2.29}$$

$$A_{33} = -Z_{\dot{w}} = \int_{-L/2}^{L/2} A_{33}^{2D}(y,z)dx = \frac{1}{2}L\pi\rho b^2, \tag{2.30}$$

$$A_{44} = -K_{\dot{p}} = \int_{-L/2}^{L/2} A_{44}^{2D}(y,z)dx = \frac{1}{16}L\pi\rho\left(b^2 - a^2\right)^2. \tag{2.31}$$

The parameter $A_{55}$ can be calculated as

$$A_{55} = -M_{\dot{q}} = \int_{-L/2}^{L/2} x^2 A_{33}^{2D}dx + \int_{-a}^{a} z^2 A_{11}^{2D}dz = \frac{a^3 m}{30} + \frac{L^3\rho a^2}{24}. \tag{2.32}$$

Finally the last coefficient $A_{66}$ can be calculated as

$$A_{66} = -N_{\dot{r}} = \int_{-b}^{b} y^2 A_{11}^{2D}dy + \int_{-L/2}^{L/2} x^2 A_{22}^{2D}dx = \frac{b^3 m}{30} + \frac{L^3\rho a^2}{24}. \tag{2.33}$$

With the notations above in Eq.s 2.28-2.33 the added mass matrix $M_A$ can now be rewritten in terms of $A_{11}, A_{22}, A_{33}, A_{44}, A_{55}$ and $A_{66}$

$$\boldsymbol{M}_A = \begin{bmatrix} \frac{m}{20} & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{L\rho a^2}{2} & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{L\pi\rho b^2}{2} & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{16}\left(b^2 - a^2\right)^2 L\pi\rho & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{ma^3}{30} + \frac{L^3\rho a^2}{24} & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{mb^3}{30} + \frac{L^3\rho a^2}{24} \end{bmatrix}. \tag{2.34}$$

### 2.4.2 Gravitational forces and moments

When calculating the gravitational moments and hydrostatic forces of the RR the restoring forces will depend on the metacentric height $\overline{GM}_i$ of RR, where $i \in \{T,L\}$ for transverse and longitudinal metacentric height and is the distance from the metacenteric $M_i$

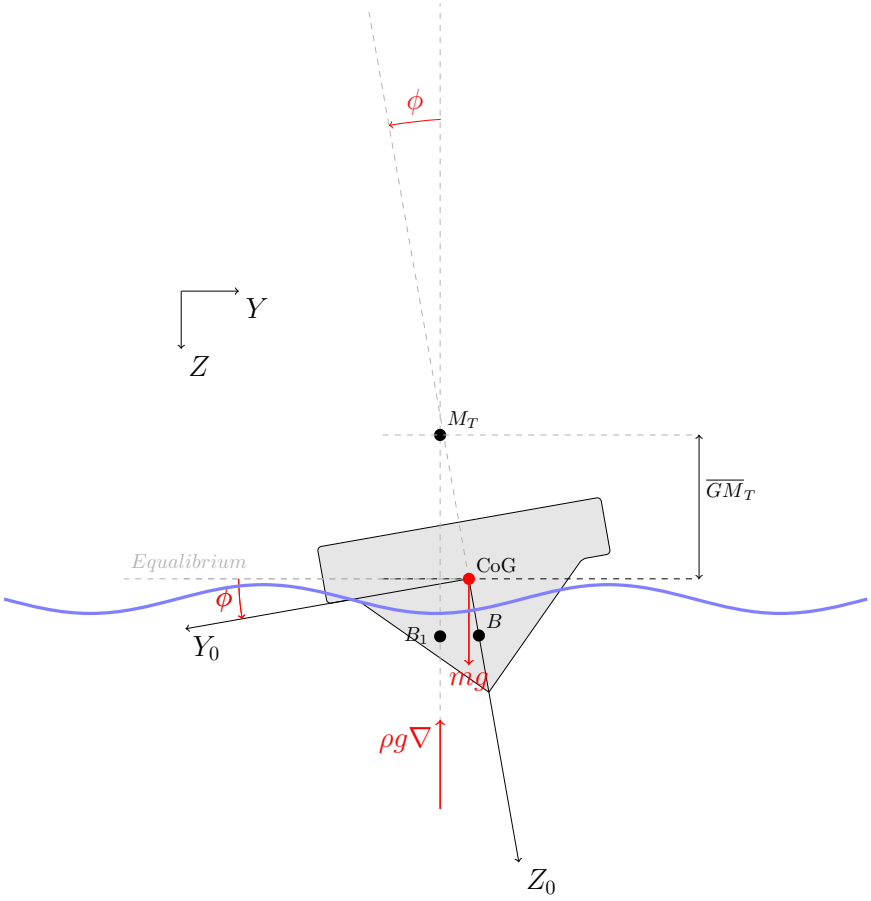



**Figure 2.7:** Illustration of the transverse metacentric stability. The grey area illustrates the transverse metacentric stability. $\overline{GM}_T$ is the transverse metacentric height. When the RR is at rest $mg = \rho g \nabla$. The lateral metacentric stability can be expressed in a similar way, simply by replacing $M_T$ to $M_L$ and $\phi$ with $\theta$.

to the $CoG$, the location of the center of gravity and the center of buoyancy ($B$) [3], see Fig. 2.7.

Thus to this, the restoring forces will only affect the heave, pitch and roll modes [3]. The complete parameter list is shown in Tab. 2.4.

**Table 2.4:** Parameter list of gravitational forces and moments

| | |
|---|---|
| $\rho$ | : water density $(kg/m^3)$ |
| $z_G(\boldsymbol{\nu})$ | : z-coordinate center of gravity $(m)$ |
| $z_B(\boldsymbol{\nu})$ | : z-coordinate center of buoyancy $(m)$ |
| $\nabla$ | : displaced volume of water $(m^3)$ |
| $A_{wp}(\boldsymbol{\eta})$ | : water plane area $(m^2)$ |
| $\overline{GM}_T$ | : transverse metacentric height $(m)$ |
| $\overline{GM}_L$ | : longitudinal metacentric height $(m)$ |

When the RR is at rest i.e. no external forces and moments are applied to it, the buoyancy and weight are said to be in balance and can be written as Eq. 2.35.

$$mg = \rho g \nabla. \tag{2.35}$$

Let the heave displacement be $w = 0$, which will express the equilibrium position with respect to the nominal displaced water volume $\nabla$. Thus, the hydrostatic force will results in the difference between the gravitational and buoyancy forces [5]m see Eq. 2.36

$$Z_{restoring} = mg - \rho g[\nabla + \delta\nabla(z) = -\rho g \delta\nabla(z)], \tag{2.36}$$

where the variation of heave position results in the change of displace water $\delta\nabla(z)$. The change in displaced water can be expressed by Eq. 2.37.

$$\delta\nabla(Z) = \int_0^z A_{wp}(\zeta)\,\mathrm{d}\zeta. \tag{2.37}$$

Here, the water plane area, $A_{wp}$, is a function of the heave position. For simplification $A_wp(\zeta) \in A_{wp}(0)$ is always constant for small disturbances in $z$. With this assumption the restoring force $Z$ will be linear in $z$ and can be expressed as Eq. 2.38.

$$Z_{restoring} \in -\rho g A_{wp}(0) z. \tag{2.38}$$

Thus there can be situations where the RR is pushed downwards by external forces, e.q. a rider steps onto it, hence $z \geq 0$. Because of this external force the buoyancy will increase and will grow larger than the gravitational force since the submerged volume

part $\nabla$ of the hull will increase by $\delta\nabla$ to $\nabla + \delta\nabla$. This assumption by [5] is physically equivalent to a spring with stiffness and it holds that $Z_z = \rho g A_{wp}(0)$ with position $z$

$$Z_{restoring} \in -Z_z z. \tag{2.39}$$

The restoring force can be expressed in the BF frame, $\delta \boldsymbol{f}_r^b$[5] and can be seen in Eq. 2.40 and the restoring moment can be seen in Eq. 2.41.

$$\delta \boldsymbol{f}_r^b = -\rho g \nabla \begin{bmatrix} -sin(\theta) \\ cos(\theta)sin(\phi) \\ cos(\theta)cos(\phi) \end{bmatrix} \int_0^z A_{wp}(\zeta)\, \mathrm{d}\zeta. \tag{2.40}$$

$$\boldsymbol{m}_r^b = \begin{bmatrix} \overline{GM}_T sin(\phi)cos(\theta)cos\phi \\ \overline{GM}_L sin(\theta)cos(\theta)cos\phi \\ -(\overline{GM}_L cos(\theta) + \overline{GM}_T sin(\phi)sin\theta \end{bmatrix}. \tag{2.41}$$

Finally, the two Eqs. 2.40 and 2.41 can med written as the total restoring forces and moments, see Eq. 2.42.

$$\boldsymbol{g}(\boldsymbol{\eta}) = -\begin{bmatrix} \delta \boldsymbol{f}_r^b \\ \boldsymbol{m}_r^b \end{bmatrix}. \tag{2.42}$$

For further simplification it can be convenient to use a linear approximation and assuming that $\int_0^z A_{wp}(\zeta) \approx A_{wp}(0)z$ and $\phi, \theta$ and $z$ are small, with other words $\sin(\cdot) \approx (\cdot)$ and $\cos(\cdot) \approx (1), (\cdot) \in \{\theta,\phi\}$. The linear approximation can then be written

$$\boldsymbol{g}(\boldsymbol{\eta}) \approx \boldsymbol{G}\boldsymbol{\eta}. \tag{2.43}$$

Here, $\boldsymbol{G} = diag\{0,0,\rho g A_{wp}(0),\rho g \nabla \overline{GM}_T,\rho g \nabla \overline{GM}_L,0\}$, which is based on the assumption of $yz$-plan symmetry. With this mention, together with Eq. 2.42 the final simplified matrix of restoring forces and moments can be summarized in Eq. 2.44.

$$\boldsymbol{g}(\boldsymbol{\eta}) = \begin{bmatrix} -\rho g A_{wp}(0) z\theta \\ \rho g A_{wp}(0) z\psi \\ \rho g A_{wp}(0) z \\ \rho g \nabla \overline{GM_T} \psi \\ \rho g \nabla \overline{GM_L} \theta \\ \rho g \nabla (-\overline{GM_L} + \overline{GM_T} \psi \theta \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \rho g A_{wp}(0) z \\ \rho g \nabla \overline{GM_T} \psi \\ \rho g \nabla \overline{GM_L} \theta \\ 0 \end{bmatrix}. \tag{2.44}$$

## 2.5 State space model

To summarize what has been done in this chapter, an example of how a mathematical model of the RR in 6 DoF can be derived. In this last sub chapter of the modeling section, the final mathematical expression of the RR will be presented. Eq. 2.10 can now be rewritten in terms of the forces and moments acting on the RR. To simplify the model even more, the 6 DoF has been reduced to a 3 DoF model with heave, roll and pitch neglected, see Tab. 2.5.

**Table 2.5:** Notations of the RR in 3 DoF

| **3 DoF** | | **Forces and moments** | **Linear and angular vel.** | **Position and Euler angles** |
|---|---|---|---|---|
| 1. Motions in the x-direction | (surge) | X | u | x |
| 2. Motions in the y-direction | (sway) | Y | v | y |
| 4. Rotation around the z-axis | (yaw) | N | r | $\psi$ |

With the new 3 DoF the state space model can be expressed as Eq. 2.45 and 2.46.

$$\dot{\boldsymbol{\eta}} = \boldsymbol{J}(\boldsymbol{\nu})\boldsymbol{\nu}. \tag{2.45}$$

$$(\boldsymbol{M}_A + \boldsymbol{M}_{RB})\dot{\boldsymbol{\nu}} + \boldsymbol{C}_{RB}(\boldsymbol{\nu}) + \boldsymbol{C}_A(\boldsymbol{\nu}) + \boldsymbol{D}(\boldsymbol{\nu})\boldsymbol{\nu} + \boldsymbol{G}\boldsymbol{\eta} = \boldsymbol{\tau}. \tag{2.46}$$

With the state space variables:

$$\boldsymbol{\eta} = \begin{bmatrix} x \\ y \\ 0 \\ 0 \\ 0 \\ \psi \end{bmatrix} \text{ and } \boldsymbol{\nu} = \begin{bmatrix} u \\ v \\ 0 \\ 0 \\ 0 \\ r \end{bmatrix}. \tag{2.47}$$

# 3

# Simulation

In the following chapter the simulation procedure is explained. The simulations are carried out in 3 DoF and not with the complete 6 DoF model derived in section 2. This due to the fact that such complexity was not needed for the simulation results to be satisfactory. An overview of the main implementation can be seen in Fig. 3.1. Initially the *guidance system* block, implementing the Line of Sight (LOS) procedure for determining how the RR is supposed to follow the generated WPs is mentioned. Consecutively this section will be followed by a description of how the reference models for both the speed and the heading are implemented. It is also mentioned how the speed and heading angle error are controlled using two separate PID controller. Consecutively it is also described how the PID controllers are properly tuned for this purpose. Finally a connection between a model of the water jet thruster and the 3 DoF transformation from the BF coordinate system to the EF coordinate system is described. The simulation environment was developed in and performed with the MATLAB/Simulink software. The *Marine Systems Simulator Toolbox*[6] for Simulink was used since it includes a lot of convenient functions when simulating a sea environment.

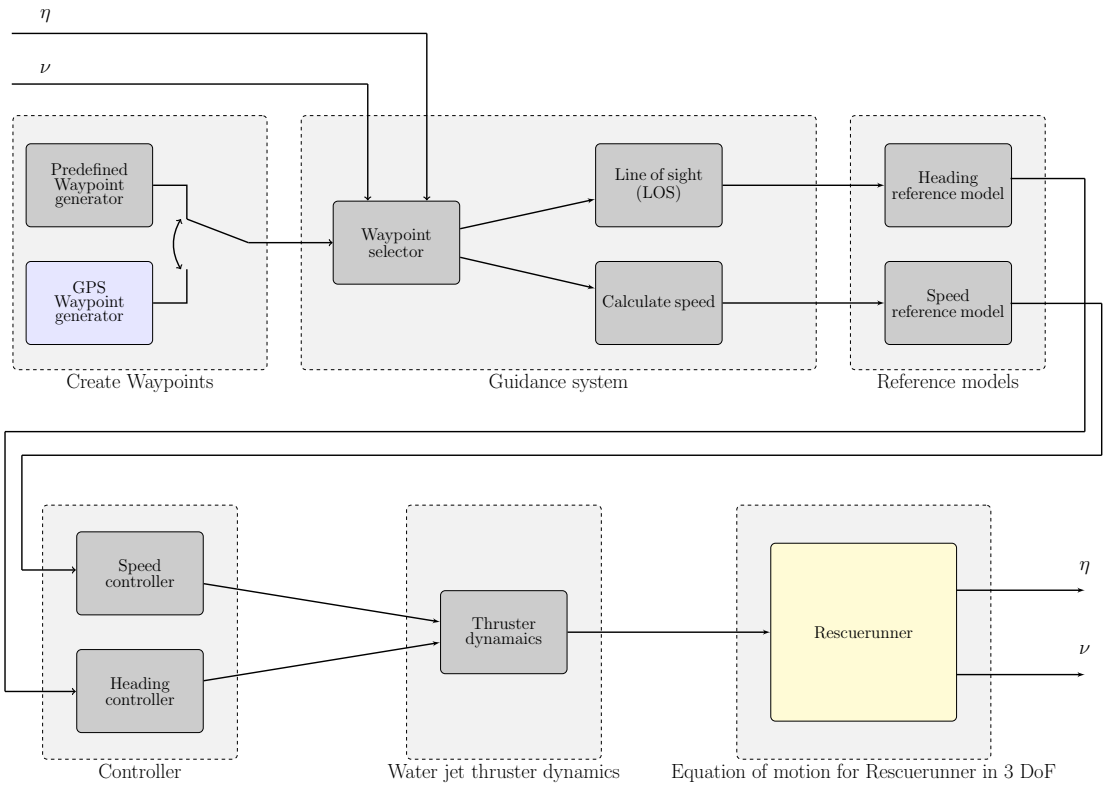



**Figure 3.1:** Illustration of the different steps of the software implementation and in what order they were carried out. The initial inputs to the systems are the to vectors $\boldsymbol{\eta}$ and $\boldsymbol{\nu}$, describing the motions and rotations of the RR with respect to the earth-fixed and the boat-fixed coordinate system respectively, together with the predefined WP generator. For the real system, the WPs will be generated from the GPS on the EB and sent to the *guidance system* block, *reference model*, *controller*, *water jet thruster dynamics* and finally the *equation of motion for the RR in 3 DoF* block. Output from the system will be the updated values of the vectors $\boldsymbol{\eta}$ and $\boldsymbol{\nu}$.

## 3.1 Generating waypoints

In this section it will be described how the WPs are generated, both for the simulation environment and for the real implementation, see Fig. 3.2.

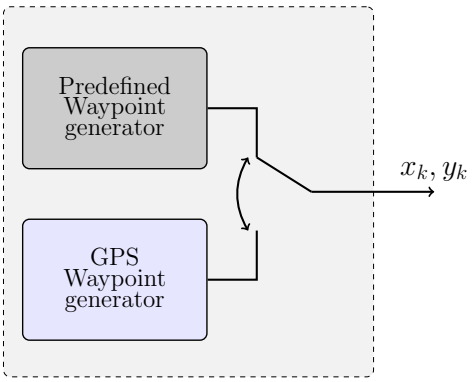



**Figure 3.2:** Schematic overview of the *Waypoint block* that is a part the software implementation presented in Fig. 3.1. In the block there are two sub-blocks, the *Predefined Waypoint generator* and the *GPS Waypoint generator* blocks. The first block is used during the simulation, to simulate WPs from the EB. Further, when the application is used in real, the second block illustrates that the GPS position of the EB will instead be taken from a GPS unit onboard.

### 3.1.1 Predefined waypoint generator

In the simulation environment the EB's WPs are predefined for the RR to follow. This means that the complete path the RR shall follow is determined before the simulation starts. A Graphical User Interface (GUI) is created in MATLAB, in which the user simply can set out WPs for the simulation. The WPs created from the GUI are stored in vector, which is fetched from the MATLAB workspace once the simulation is started. It is possible to create a path consisting of a maximum of 100 WP. This number is not limited to 100, i.e. it can be change but its recommended for faster calculations to use smaller routes.

### 3.1.2 GPS waypoint generator

A predefined WP generator can not be used in the real implementation since the RR shall be fed with a reference position at a certain sampling rate, depending on how the

EB is moving. The implementation in the real system will be described in chapter 5.

## 3.2 Guidance system

The main blocks of the *guidance system* can be seen in Fig. 3.3.

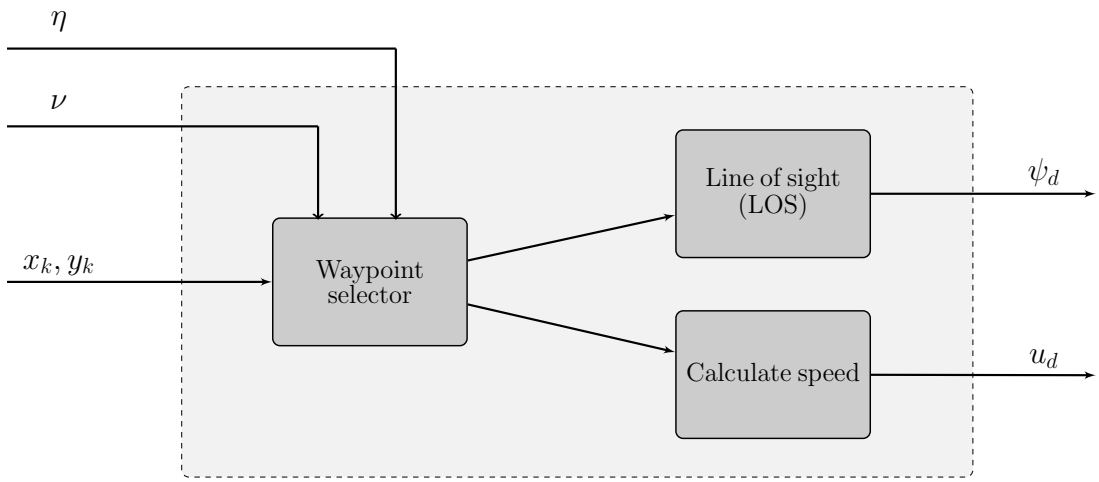



**Figure 3.3:** Schematic overview of the *guidance system* that is a part the software implementation presented in Fig. 3.1. This block determines which WP the RR shall be heading towards. When the WP and the position of the RR is known, the desired yaw angle is calculated together with the proper speed, which depends on whether the RR is on course or not.

### 3.2.1 Waypoint selection

Because the EB is continuously dropping coordinates, depending on its current position at a certain time, the RR needs to be able to determine when it shall start to look for a new WP. Let the current position of the RR be denoted as $x_i$,$y_i$ and the coordinates of the current WP as $x_{i+1}$,$y_{i+1}$. Knowing this information, i.e where the RR and the next WP it the distance to target $s$ can be calculated by Pythagorean theorem, see Eq. eq:pythan and Fig. 3.5.

$$s = \sqrt{(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2}. \tag{3.1}$$

This means that the distance $s$ can be used as a threshold value for determining when the current WP should be switched to the next one, see Eq. 3.2

$$s < r_a, \tag{3.2}$$

which means that the WP is switched when the RR is within a certain distance, $r_a$, from the WP. A figure demonstrating the path through the WPs can be seen in Fig. 3.4. The value of $r_a$ can be set arbitrarily and can be interpreted as the so called Radius of Acceptance (RoA) surrounding the WP.

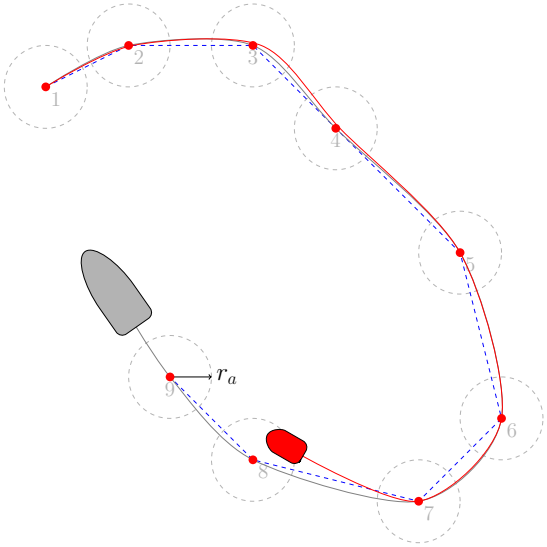



**Figure 3.4:** Illustration of the path following technique. The EB (grey in picture) drops WPs along its path. The objective for the RR (red in picture) is to navigate to the current WP before switching to the next one

Different tolerances on the RoA will be discussed more thoroughly in chapter 6. Nonetheless it can be said that a big value of $r_a$ will make it easier for the RR to follow the path of the EB, but will also contribute to bigger deviations from the original path. A smaller $r_a$, will make the path smoother since the RR would not be able to to drive as fast but will on the other hand imply a more limited and rigorous way of determining the speed of the RR.

It is suggested to implement a strategy for handling situations where a WP is missed. If a time, $t$, is calculated based on the distance between two consecutive WPs and the speed of the RR, Eq. 3.3 can be written as

$$t = \frac{\sqrt{\Delta x^2 + \Delta y^2}}{|u + v|}. \tag{3.3}$$

The value of the variable $t$ is the critical mean time the RR has to reach the next WP. This mean time can be multiplied with a constant, $k$, depending on how much time that shall be given between the WPs. This gives an additional condition for determining when the RR shall aim at a new target. The ciritcal thresholds can be summarize in Eq. 3.4

$$(x_k - x_{i+1})^2 + (y_k - y_{i+1})^2 < r^2 \;\; || \;\; t > k\frac{\sqrt{\Delta x^2 + \Delta y^2}}{|u + v|}. \tag{3.4}$$

In Simulink a clock is used for keeping track of the time $t$, which is reset once the condition is fulfilled. This implementation is very useful since WPs that are laying on so called "unstrategic" places, which is difficult for the RR to reach, will simply be neglected and replace by a more unstrategic WP.

### 3.2.2 Line of Sight

In order to make it possible for the RR to follow the EB a guidance algorithm has to be implemented. The idea is that the EB drops $x$ and $y$ coordinates at a certain sampling rate, which the RR then can follow. The WPs have a certain RoA, making the boat aim at a new WP when it is adequately close to the current one. Similarly to what is done in [7] the LOS guidance controller is designed in a way that the angle between the current position of the RR and the current WP is calculated. Say that the RRs current position is denoted $x_i$,$y_i$ and the position of the WP it currently is aiming at is denoted $x_k$,$y_k$. Then the LOS-angle from the WP can be calculated using the $atan2$ function, i.e Eq. 3.5

$$\psi = atan2\left(\frac{y_k - y_i}{x_k - x_i}\right) = atan2\left(\frac{\Delta y}{\Delta x}\right), \tag{3.5}$$

where $atan2$ is the four-quadrant inverse of tangent. The geometry is illustrated in Fig. 3.5. Eq. 3.5 is solved online for every step, which means that for every sample the position of the RR is updated and a new LOS-angle is calculated, therefore Eq. 3.5 can express the desired heading angle, $\psi_d$, see Eq: 3.6

$$\psi_d = atan2\left(\frac{y_k - y_{i+1}}{x_k - x_{i+1}}\right), \tag{3.6}$$

where the value $i$ is increased every sample, e.g. for the initial position the value will be $x_1$,$y_1$ and then consecutively increasing with a value of 1 throughout the simulation.

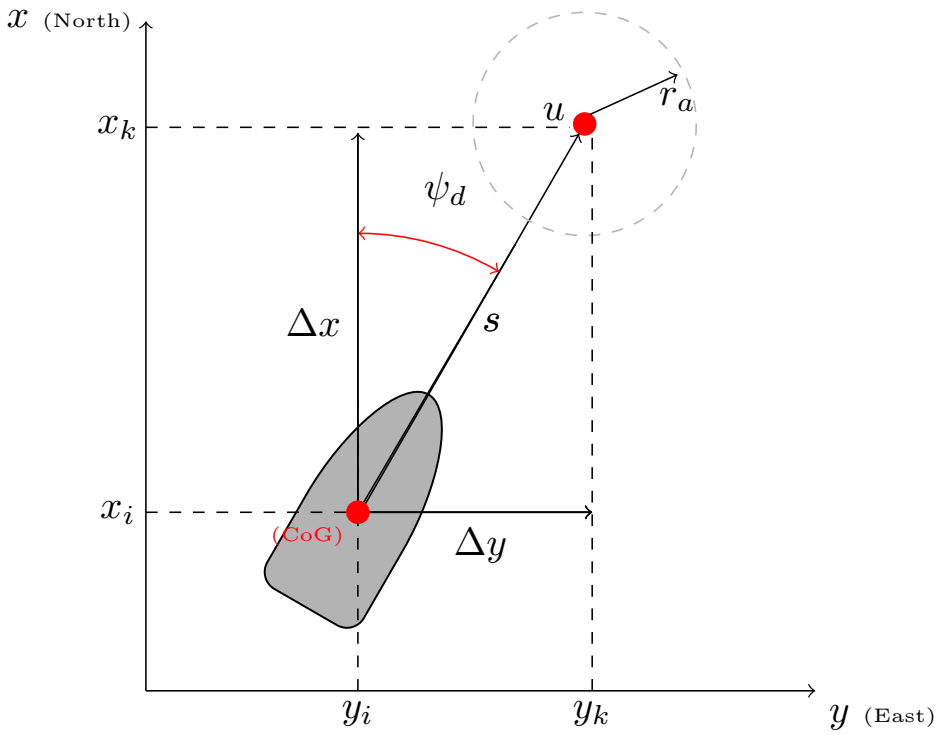



**Figure 3.5:** Illustration of how the desired heading angle, $\psi_d$, is calculated. The desired angle $\psi_d$ to the next WP is determined by taking the four quadrant inverse tangent $atan2$ of $\Delta y$ divided by $\Delta x$. The distance $s$ from the current position of the RR to the next target is calculated simply by the Pythagorean theorem. The RoA is denoted $r_a$.

### 3.2.3 Calculating a continuous Line of Sight angle

Because of the fact that *atan*2 function is discontinuous between the intersection $(-\pi,\pi)$ a mapping has to be performed to make sure that the desired yaw angle, $\psi_d$, do not behave strange when crossing this particular point. If this is not accounted for the RR its desired heading angle suddenly will jump from $-\pi$ to $\pi$ if this intersection is crossed. Hence, the RR will make a 360° turn. This is solved by using a procedure similar to [8] where the discontinuous angle is mapped into a continuous angle that is fed to the *reference model*. Hence the problem is solved by developing a mapping for the LOS-angle from $(-\pi,\pi)$ to $(-\infty,\infty)$. The pseudo code for the implementation can be found in Appendix A but a brief explanation of the procedure is found below.

By dividing the unit circle into its four regular quadrants the current state can be kept track on by introducing a variable called `state`. This variable can be used for saving the current value of the desired heading angle in a new variable denoted `psi_current`. A variable for keeping track of the previous heading angle, i.e. the previous value of the *atan*2-function is also needed and this variable can be denoted `psi_previous`. The challenge here is to keep the change between the previous and the current angle as small as possible. The last variable needed is a variable that is keeping track of the accumulated angular information, i.e. the angular change for all the previous time steps. In the beginning of a simulation all three variables are set to zero and the challenge is to compute the `accumulation` variable in a way such that the angular change between the current and previous value of the angle become as small ass possible. The last thing that happens in the code is that the variable `accumulation`, i.e. the accumulated angular error, is increased and the value of the variable `psi_previous` is assigned the value of the variable `psi_current` and the procedure is done over and over again throughout the simulation.

### 3.2.4 Speed calculation

A speed calculation is also performed in the *guidance system* block as a way to increase safety, since the RR is prevented from driving too fast depending on how close it is to the WP. In the function where the WPs are created also a recommended speed between two WPs are calculated. The speed is chosen as a function of the angle between the current and the consecutive WP, $x_k$,$y_k$ and $x_{k+1}$,$y_{k+1}$. This angle can be calculated as

the angle between the two vectors, see Eq. 3.7.

$$\alpha_k = atan2(\|\boldsymbol{z}_k \times \boldsymbol{z}_{k+1}\|, \boldsymbol{z}_k \cdot \boldsymbol{z}_{k+1}), \tag{3.7}$$

where $\boldsymbol{z}_k = [x_k, y_k]^\top$ and $\boldsymbol{z}_{k+1} = [x_{k+1} - x_k, y_{k+1} - y_k]^\top$. It would not be necessary to reduce the speed if the RR is moving along a straight path. However, if it has to perform a sharp turn, it sounds reasonable that the speed is set to a lower value. This can be implemented as in [9], where the preferable (angle dependent) velocity is calculated using a Gaussian normal distribution function, see Eq. 3.8

$$v(\alpha_k) = ce^{-\frac{\alpha_k^2}{\sigma^2}}. \tag{3.8}$$

Here the parameter $\alpha_k$ determines the position of the center of the peak and the parameter $\sigma$ determines the width of the "bell" that arises in a normal distribution function. The coefficient, $c$, bounds the value between an upper and a lower limit. Since the velocity given from the exponential function needs to be bounded it can be favorable to describe a relation between the maximum and minimum velocity in order to keep the traveling speed at reasonable values. This is done by introducing the following equation as the coefficient, $c$, in front of Eq. 3.8 such as

$$c = v_{min} + \Big(v_{max} - v_{min}\Big). \tag{3.9}$$

The exponential function has the following properties

$$\lim_{b\to\infty} e^{-b} = 0 \tag{3.10}$$

$$\lim_{b\to 0} e^{-b} = 1, \tag{3.11}$$

which means that when the exponential function yields a large value (close to 1) the RR will travel with a speed that is approximately equal to $v_{max}$. On the other hand, if the exponential function yields a small value (close to 0) the RR will travel with a speed that is approximately equal to $v_{min}$ [9]. Hence the expression for the velocity can be calculated as,

$$v(\alpha_k) = v_{min} + \Big(v_{max} - v_{min}\Big)e^{-\frac{\alpha_k^2}{\sigma^2}} \tag{3.12}$$

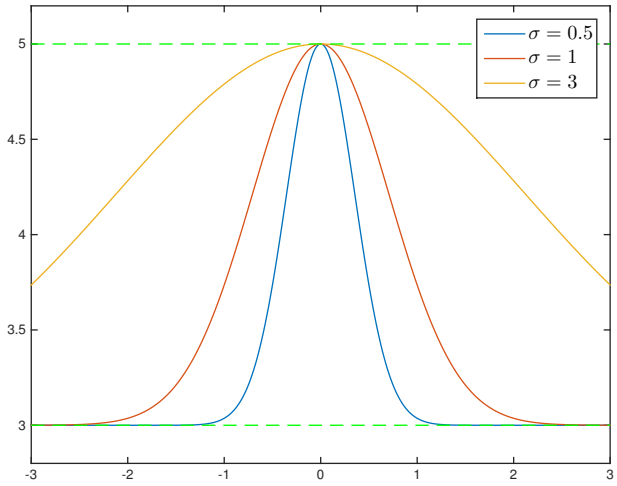



**Figure 3.6:** Illustration of the Gaussian function used for setting the speed limit. The shape of the Gaussian function for three different values of $\sigma$ is illustrated. The function is bounded between the velocities $v_{min} = 3\text{m/s}$ and $v_{max} = 5\text{m/s}$.

A plot of the Gaussian function with velocity bounds implemented is shown in Fig. 3.6. in the picture the angle, $\alpha_k$, is plotted from $-\pi$ to $\pi$. The velocities are chosen as $v_{min} = 3\text{m/s}$ and $v_{max} = 5\text{m/s}$ and sigma are varied from between three different values, as can be seen in the figure. The calculated value of the velocity is, together with the coordinates of the WPs, saved in a matrix and fetched from the MATLAB workspace once a simulation is executed.

## 3.3 Reference model

It is preferable to use a *reference model* for the desired heading and the desired velocity to smooth out the control signals, hence avoiding a fitfully behavior. For the *heading reference model* this is done be sending the desired angle, i.e. $\psi_d$ through a third-order low-pass filter [5], see Fig. 3.7.

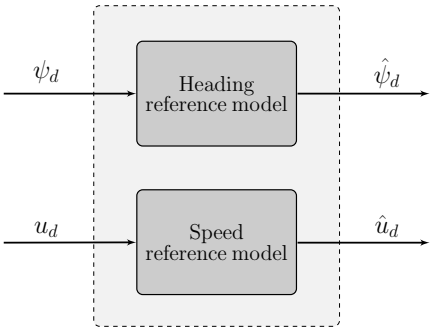



**Figure 3.7:** Schematic overview of the *Reference model* block that is a part of Fig. 3.1. A third-order low-pass filter obtains the reference models for heading and speed for each of the two signals.

This is a common procedure in applications of trajectory tracking control when the heading or velocity is changed. This is also implemented in this thesis where a similar system is modeled. When deriving a *reference model* for such an application, a model of the third-order is preferable for satisfactory results. In [5][9] it is suggested that a first-order low-pass filter is multiplied with a mass-spring-damper system. The transfer function for a low-pass filter with the cutoff frequency $w_n$, see Eq. 3.13

$$H(s) = \frac{\omega_n}{s + \omega_n}. \tag{3.13}$$

The transfer function in Eq. 3.14can be used to explain the spring-mass-damper system

$$G(s) = \frac{w_n^2}{s^2 + 2\zeta\omega_n s + w_n^2}. \tag{3.14}$$

A third-order *reference model* is now derived by multiplying the two transfer functions, $H(s)$ and $G(s)$, in Eq. 3.15.

$$\frac{\psi_d}{\hat{\psi}_d} = \frac{\omega_n^3}{s^3 + s^2(2\zeta\omega_n + \omega_n) + s(2\zeta\omega_n^2 + \omega_n^2) + \omega_n^3}. \tag{3.15}$$

This result in a transfer function from the desired angle $\psi_d$ to the filtered desired angle $\hat{\psi}_d$. From Eq. 3.15 it shall be noted that $\omega_n$ represents the natural frequency and $\zeta$ denotes the damping. For such a system it shall be observed that the final value theorem yields that the desired angle, $\psi_d$, approaches the filtered desired angle, $\hat{\psi}_d$, as the time, $t$, approaches infinity. This can be seen in Eq. 3.16

$$\lim_{t\to\infty} \psi_d = \hat{\psi}_d. \tag{3.16}$$

Filtering the signal is of importance, since it decreases the error between the desired and the actual heading. It is important that the eigenvalues of the desired states are chosen in a right way. That means that the bandwidth of the closed-loop system shall never be exceeded by the cutoff frequency of the filtered desired angle. Such behavior will have the outcome that the RR will not be able to track the desired state. For implementation of this model in the simulation environment it is preferable to rewrite Eq. 3.15 on a canonical controllable form[10], see Eq. 3.17

$$\begin{aligned} a_1 &= 2\zeta\omega_n + \omega_n, \\ a_2 &= 2\zeta\omega_n^2 + \omega_n^2, \\ a_3 &= w_n^3. \end{aligned} \tag{3.17}$$

Eq. 3.15 is rewritten with inverse Laplace transformed and coefficients from Eq. 3.17 are used, see Eq. 3.18.

$$\dddot{\psi}_d + a_1\ddot{\psi}_d + a_2\dot{\psi}_d + a_3\psi_d = a_3\hat{\psi}_d. \tag{3.18}$$

The state space model can now be derived according to Eq. 3.19.

$$\begin{aligned} q_1 &= \psi_d & \dot{q_1} &= \dot{\psi}_d \\ q_2 &= \dot{\psi}_d & \dot{q_2} &= \ddot{\psi}_d \\ q_3 &= \ddot{\psi}_d & \dot{q_3} &= \dddot{\psi}_d. \end{aligned} \tag{3.19}$$

Added to this, according to Eq. 3.18 the expression for $\dot{q_3}$ can be written as Eq. 3.20.

$$\dot{q_3} = a_3\hat{\dot{\psi}}_d - a_1\ddot{\psi}_d - a_2\dot{\psi}_d - a_3\psi_d. \tag{3.20}$$

With this said, the states can be chosen as $\boldsymbol{\psi_d}(s) = [\psi_d.\dot{\psi}_d, \ddot{\psi}_d,] = [\psi_1, \psi_2, \psi_3]$ and the state space model can be written on controllable canonical, see Eq. 3.21

$$\begin{bmatrix} \dot{\psi}_1 \\ \dot{\psi}_2 \\ \dot{\psi}_3 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -a_3 & -a_2 & -a_1 \end{bmatrix} \begin{bmatrix} \psi_d \\ \dot{\psi}_d \\ \ddot{\psi}_d \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ a_3 \end{bmatrix} \hat{\boldsymbol{\psi}}_{\boldsymbol{d}} \tag{3.21}$$

,

$$y = \begin{bmatrix} \omega_n^3 & 0 & 0 \end{bmatrix} \boldsymbol{\psi_d} \tag{3.22}$$

This *reference model* is implemented for the speed and the heading signal but the parameters are set a bit different in the two models, although the procedure is carried out in a similar manner. In Fig. 3.8 the difference between the desired heading angle before ($\psi_d$) and after ($\hat{\psi}_d$) the *reference model* are shown. It becomes clear that the *reference model* manages to efficiently smooth out the curve, hence making it possible to avoid a fitfully behavior.

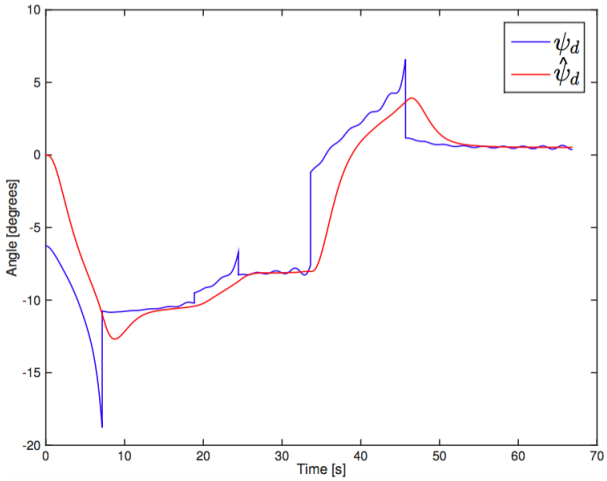



**Figure 3.8:** Comparison between the desired heading before the reference model, $\psi_d$, and after the reference model, $\hat{\psi}_d$. The figure clearly illustrates how the filtering smooths the desired heading after the reference model out.

The filtered desired angle, $\hat{\psi}_d$, is later on sent to the controller. Fig. 3.9 illustrates the filtered desired angle, $\hat{\psi}_d$, and the output from the controller, i.e. $\psi_r$ which is the steering control signal sent to the RR. Note that the reference angle, i.e. the control signal sent to the RR, $\psi_r$ is oscillating around the curve for the filtered desired heading $\hat{\psi}_d$ with a maximum value of $\pm 5$ degrees. It can be discussed whether this is a good behavior or not but clearly the RR is staying on course and the mean value of the oscillations are approximately equal to the value of the desired heading. A good idea is to implement a constraint on the steering signal since the control signal from the heading is very active, strongly oscillating with $\pm 5$ degrees around the input value.

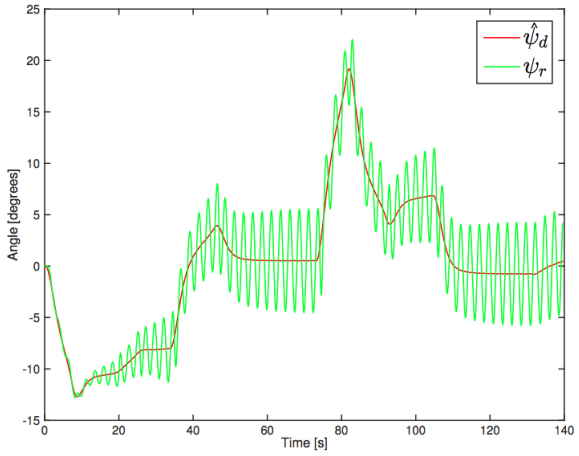



**Figure 3.9:** Comparison between the desired heading, $\psi_d$, and the reference heading, $\psi_r$ of the RR. The oscillations of the reference heading, $\psi_r$, sent to the RR by the controller indicates an active control signal.

## 3.4 Controller

Since the purpose is to make the RR able to follow an EB the only parameters that needs to be controlled is the surge and yaw angle. Minimizing the pitch and heave moments will not be taken into consideration, see Fig. 3.10. This chapter will present basic PID tuning approach. With that said this chapter will be present a guideline how the PID controllers should be tuned.

For this application two controllers will be constructed based on the general expression for a simple PID controller, see Eq. 3.23.

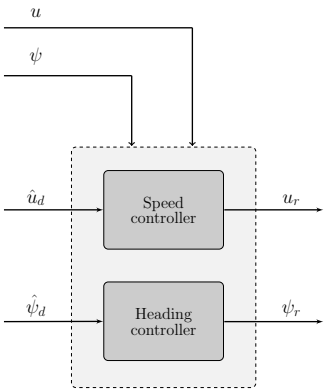



**Figure 3.10:** Schematic overview of the *Controller block* that is a part of Fig. 3.1. In this block, the speed $u_r$ and the reference heading angle $\psi_r$ is regulated by two PID controllers.

$$u(t) = K_p e(t) + K_i \int_0^t e(t)dt + K_d \frac{de(t)}{dt}, \tag{3.23}$$

where $u(t)$ is the input signal, $K_p, K_i$ and $K_d$ are the proportional, integral and derivative tuning gain parameters and $e(t)$ is the resulting error. The main objective is therefore to minimize the error angle relative the target by forcing the RR to steer in the right direction by

$$e_\psi(t) = \lim_{t\to\infty}(\psi - \psi_d) = 0. \tag{3.24}$$

A similar way goes for the speed controller. As mentioned in section 3.2.4 the velocity changes depending on whether the RR are on course or not. If the yaw error is close to zero the model will allow the RR to go faster. The speed error is is therefore also desired to be minimized as

$$e_v(t) = \lim_{t\to\infty}(v - v_d) = 0. \tag{3.25}$$

When applying speed and heading controllers to the RR one approach is to use the so called first-order Nomoto model[11], see Eq. 3.26.

$$r = \frac{K}{1+Ts}\delta. \tag{3.26}$$

The Nomoto model explains the transformation from the water jet thruster angle $\delta$ to the actual yaw angle $r$ of the RR. The thruster angle is measured in the BF coordinate system

while the yaw angle is measured in the EF coordinate system. The two parameters $K$ and $T$ from the Nomoto model are often used to calculate the parameters of PID-controller. With this said, the Eq. 3.26 can be rewritten with the assumption that the yaw velocity is equal to the yaw rate ($\dot{\psi} = r$, see Eq. 3.27

$$\psi = \frac{\psi}{s(1+Ts)}\delta. \tag{3.27}$$

Further, it can be said that mass of the RR is related to Nomoto's first-order model, see 3.28

$$m = \frac{T}{K}. \tag{3.28}$$

It is also necessary to calculate the natural frequency $\omega_n$, which is depending on the relative damping ratio $\zeta$ and the closed-loop-bandwidth $\omega_b$ of the regulated system, see Eq. 3.29.

$$\omega_n = \frac{\omega_b}{\sqrt{1-2\zeta^2+\sqrt{4\zeta^4-4\zeta^2+2}}}. \tag{3.29}$$

Normally the relative damping ratio $\zeta \in [0\ 1]$ depending on how the system is design to tolerate the amount of overshooting[10]. When $\zeta \in [0\ 1]$ will result in a damped oscillation, which is preferable in this case. $\omega_b$ should be chosen between the bandwidth of the Nomoto model and the thruster angle bandwidth[9].

Finally, the three gain parameters can be calculated, see Eqs. 3.30-3.30. It can be noted that the integral gain $K_i$ is set as $\frac{K_p}{10}$. This can be explained by a relation of a large proportional gain compared to a smaller integral gain will provide a slower decaying step disturbance.

$$\text{The proportional gain: } K_p = m\omega_n^2 \in K_p > 0. \tag{3.30}$$

$$\text{The integral gain: } K_i = \frac{1}{10}\omega_n K_p \in K_i > 0. \tag{3.31}$$

$$\text{The derivative gain: } K_d = 2m\zeta\omega_n - \frac{1}{K} \in K_d > 0. \tag{3.32}$$

As the system is nonlinear it will most likely behave dissimilar in different situations. This mostly depends if the RR have a low speed compared to when the RR planes. Thus of this phenomena the application requires the control parameters to be adaptive, a so called *gain scheduling*[10]. This can easy be done by setting up a certain thresholds where the PID controllers uses different parameters depending in which kind of state its currently is in into.

## 3.5 Water jet thruster dynamics

In similar work where a mathematical model for a boat is derived it concerns either a boat driven with a classic propeller [7][8][9] or a sailing yacht [4][12][13]. Since the RR is driven by a water jet thruster instead of the principles mentioned above, a new block handling a jet thruster must be developed. The water jet works such that it pushes the RR forward by shooting water in a certain direction, contributing to a force that pushes the RR in the desired direction. Hence the water jet works as both a propeller and a rudder simultaneously.

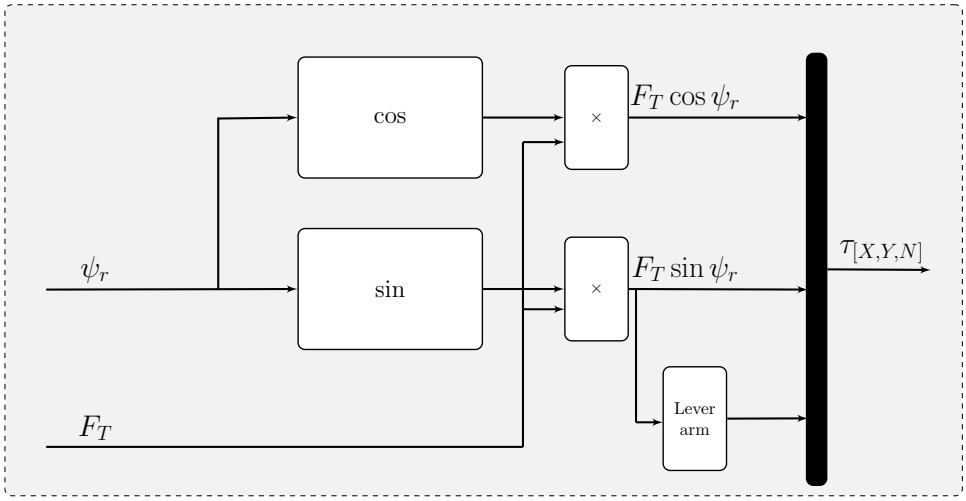



**Figure 3.11:** A schematic overview of the *Water jet thruster dynamics* model that is a part of Fig. 3.1.

The output from the speed controller, i.e. the throttle, $F_T$, and the output from the heading controller, i.e. the reference angle, $\psi_r$, is therefore combined in a way such that the driving force $F_T$ can be divided into an $x$ and $y$ component, see Eq. 3.33.

$$F_y = F_T \cos \psi_r,$$
$$F_x = F_T \sin \psi_r, \tag{3.33}$$

which is then fed to the 3 DoF-model of the RR alongside with a moment $N$ around the $z$-axis, which is proportional to the water jet angle and makes it possible for the RR to turn. An overview of the procedure can be seen in Fig. 3.11. It shall also be noted that the force in the $z$-direction and the moments around the , $x$- and $y$-axis, denoted $K$ and $M$ are neglected.

## 3.6 Equation of motion in 3 DoF

As mentioned in the introduction of chapter 3 the simulations, in which the RR follows a predefined path, are carried out in a 3 DoF. Although the model allows a 6 DoF simulation, this was not considered necessary for this project, hence the 3 DoF-model was considered sufficient for yielding satisfactory results. This means that the forces in the $x-$ and $y-$ direction (surge and sway) and the moment around the $z-$axis, the yaw, are used. Hence the force in the $z-$direction (heave) and the moments around the $x-$ and $y-$axis (roll and pitch) are neglected.

As shown in Fig. 3.12, the forces $\boldsymbol{\tau}$ are calculated from the *water jet thruster dynamics* in section 3.5. This is further fed to the 3 DoF-model, where the calculations of the state space variables $\boldsymbol{\eta}$ and $\boldsymbol{\nu}$ are computed and fed back throughout the whole simulation.

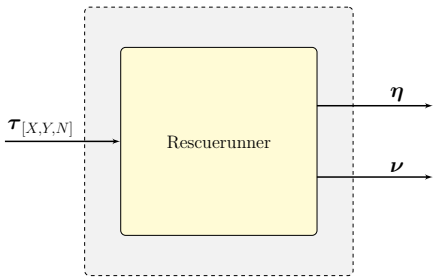



**Figure 3.12:** Schematic overview of the *Equation of motion block* that is a part of Fig. 3.1.

For the 3 DoF-model a simulation block from the *Marine Systems Simulator* toolbox [6] was used. This block includes speed-dependent fluid memory effects, cross-flow drag

and surge resistance[14].

This block is convenient to use since only the matrices that concern the mass inertia, $\boldsymbol{M}_A$, the added mass, $\boldsymbol{M}_{RB}$, the inverse mass matrix, $(\boldsymbol{M}_A + \boldsymbol{M}_{RB})^{-1}$ and and the buoyancy matrix, $\boldsymbol{g}$ have to be determined before the simulation. The matrices are calculated and described in the chapter 2. The Coriolis matrices in the state space form, i.e. $\boldsymbol{C}_A$ and $\boldsymbol{C}_{RB}$ are calculated from the inputs to the model automatically.

# 4

# Sensor fusion

When the simulations were carried out, the heading, i.e. the yaw angle, of the RR in the EF coordinate system was always known. In order to achieve this angle in reality the RR has to use an unit telling which heading the vessle holds. This can be done by using a GPS unit and combine it with an *Interial Measurement Unit* (IMU), see Fig. 4.1. An IMU is a combination of two different sensors, an accelerometer and a gyroscope. Sometimes a magnetometer is included as well. An IMU is a useful tool for measuring the velocity, gravitational forces and heading of a vessel[15]. For the implementation done in this thesis the follow components are used: the *ADXL345* accelerometer, the *ITG3200* gyroscope and the *HMC5843* magnetometer, which together represent 9 DoF.

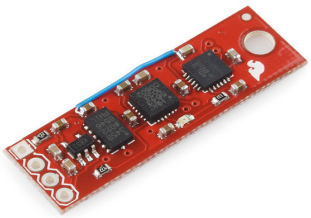


**Figure 4.1:** The IMU with accelerometer *ADXL345*, gyroscope *ITG3200* and magnetometer *HCM5843* [16]

The IMU is complemented by a GPS device. Combining these sensor yields a stable measurement of the heading angle of the RR. As the GPS device only works proper when moving, it is not appropriate to use at low velocities. The IMU, on the other hand, is much more accurate at low velocities. Hence, the heading angle is fetched from the GPS device or the IMU depending on the velocity of the RR, i.e.

$$velocity = \begin{cases} IMU & \text{if } velocity < threshold\text{m/s} \\ GPS & \text{if } velocity >= threshold\text{m/s} \end{cases}$$

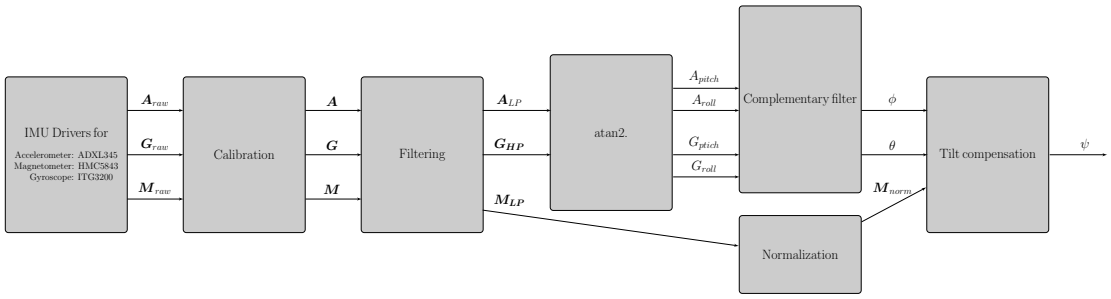



**Figure 4.2:** Schematic overview of the IMU sensor fusion procedure. The consecutive steps in the procedure of combining the accelerometer, the gyroscope and the magnetometer presented in the figure yields a tilt compensated yaw angle, $\psi$.

## 4.1 Arduino I$^2$C communication

To read the raw data from the 9 DoF IMU stick, it is coupled to an Arduino Mega 2560 board[17]. This application uses the Inter-Integrated Circuit (I$^2$C) bus, which enables multi-master, multi-slave, single-ended setups. The $I^2C$ is usually used for attaching lower-speed peripherals to processors on computer motherboards and embedded systems, making it useful for this application[18]. The pin configuration can be seen in Tab. 4.1.

**Table 4.1:** Pin configuration between Arduino Mega 2560 and 9 DoF IMU

| **9 DoF IMU** | **Arduino Mega 2560** |
|---|---|
| VCC | 3.3V |
| GND | GND |
| SCL | SCL |
| SDA | SDA |

Each pin on the IMU stick is connected to a pin on the Arduino board[18]. When the VCC and GND is connected this application just need two more wires, the Serial Clock Line (SCL) and the Serial Data Line (SDA). The SCL wire is a clock line, synchronizing all data that is transferred over the $I^2C$ bus and the data is transferred over the SDA wire[18].

In order to collect data from the IMU stick, multiply register addresses have to be read and written from. This is done by using the $I^2C$ connection bus on the IMU stick. Register addresses for the three sensors combined on the IMU can be found in the data sheet for each sensor respectively[19][20][21]. Below a few examples in pseudo code are shown on how the reading of the sensors was performed in this particular implementation. The authors of this thesis implemented the pseudo code in the C-language but of course it can be changed to other programming languages as well. As an example the gyroscope's (ITG3200[20]) registers can be reached by the following pseudo code.

**Algorithm 1** Algorithm for initializing the ITG3200 Gyroscope

```
1:  procedure READ RAW DATA FROM ITG3200 GYROSCOPE
2:      define GYRO_ADDRESS
3:      define SMPLRT_DIV
4:      define GYRO_DLP_FS
5:      define GYRO_FS
6:      define FORMAT_GYRO_DATA
7:      define GYRO_X_LSB
8:
9:      start i2c connection, set sample full scale measurement mode and sample rate
10:     Begin transmission from GYRO_ADDRESS
11:     Write on address GYRO_DLP_FS
12:     Write on address GYRO_FS
13:     End connection from GYRO_ADDRESS
14:
15:     Begin transmission from GYRO_ADDRESS
16:     Write on address SMPLRT_DIV
17:     Write on address 0x1F
18:     End connection from GYRO_ADDRESS
19:
20: end procedure
```

This code described in Alg. 1 is an initialization of the sensor. Hence, it only has to be executed before the main program has started. This code tells the program how to read the data that is coming from the sensor. Certain register addresses on the *ITG3200* gyroscope sensor are defined in the program. Consecutively the $I^2C$-connection is initialized with a chosen sample rate as well as a measurement mode[20]. This is important in order to understand the output of the sensor and how much data that is sampled from the sensor each second.

To get interpretable values from the *ITG3200*, the least and most significant bits for the readings on the $X-$, $Y-$ and $Z-$axis must be combined. The procedure of reading the sensor data and to combine the readings to interpretable values is described in Alg. 2.

**Algorithm 2** Algorithm for reading ITG3200 IMU device

```
1:  procedure READ RAW DATA FROM ITG3200 GYROSCOPE
2:
3:      Begin transmission from GYRO_ADDRESS;
4:      Write on GYRO_XLSB;
5:      Request 6 bytes from (GYRO_ADDRESS);
6:
7:      while (connection available) {
8:      GyroValues [j++] = Read values from GYRO_ADDRESS);
9:      }
10:
11:     end connection from GYRO_ADDRESS
12:
13:     GyroX = (X_LSB << 8) +X_MSB; // convert x reading to int 16
14:     GyroY = (Y_LSB << 8) +Y_MSB; // convert y reading to int 16
15:     GyroZ = (Z_LSB << 8) +Z_MSB; // convert z reading to int 16
16:
17: end procedure
```

On rows 13,14 and 15 the values of the least significant bit are bit shifted and added to the most significant bit of the reading of each axis in order to get proper values. The procedure how to combine the least at most significant bits differs for different sensors on the IMU stick. In this example the readings relate to the *ITG3200* gyroscope but similar procedures hold for the *ADXL345* accelerometer and the *HMC5843* magnetometer. The full C-code implementation is found in Appendix D

## 4.2 Inertial measurement unit

A tilt compensated digital compass requires an accelerometer, a gyroscope and a magnetometer to work properly. Since the sensors have there pros and cons a fusion algorithm has to be implemented for the readings to be reliable. The sensor fusion procedure is shown in Fig. 4.2, where the pitch and roll angles are derived by combining the signals from the accelerometer and the gyroscope, passing them through a complementary filter. This filtering gives a stable signal on the pitch angle, $\varphi$, and the roll angle, $\theta$. The

tilt compensation is then performed using filtered angles combined with the readings from the magnetometer[22]. The sensors used in this project, with their corresponding notations can be found in Table 4.2.

**Table 4.2:** Sensor notations

| **Sensor typ** | **Sensor name** | **Vector - 3 axes** | **Parameters** | | |
|---|---|---|---|---|---|
| Accelerometer | ADXL3456 | $\boldsymbol{A}$ | $A_x$ | $A_y$ | $A_z$ |
| Gyroscope | ITG3200 | $\boldsymbol{G}$ | $G_x$ | $G_y$ | $G_z$ |
| Magnetometer | HCM5843 | $\boldsymbol{M}$ | $M_x$ | $M_y$ | $M_z$ |

## 4.3 Calibration

Before using the sensor measurements, each sensor must be calibrated for reading accurate values. The calibration is done individually for each sensor.

### 4.3.1 Accelerometer

Calibrating the accelerometer is fairly easy. First the sensor values need to be transformed from the raw data to values in $g$ $[m/s^2]$ [19], see Eq. 4.1.

$$\boldsymbol{A}_g = \gamma \boldsymbol{A}_{raw} \tag{4.1}$$

where $\gamma = 0.0039063$ is the scaling factor used for the conversion. When the offset is calculated, the sensor should lie flat in the xy-plane, where the x-axis pointing forward, y-axis pointing left and z-axis pointing downwards. Eq. 4.2 calculates the offset for each axis.

$$\boldsymbol{A} = \boldsymbol{A}_g - \boldsymbol{A}_{off}. \tag{4.2}$$

This can also be seen in Fig. 4.3, where $A_{x,off}$ is sub tracked from $A_x$,$g$. The signal is pretty noisy, hence it can result in a unwanted behavior. Because of this the signal from the accelerometer needs to be low-pass filtered, see Eq. 4.3.

$$\boldsymbol{A}_{LP} = \boldsymbol{A}_{LP} * \alpha + (1 - \alpha)\boldsymbol{A}_{LP,pre}. \tag{4.3}$$

Further explanation of the low-pass filtering can be found in section 4.4. The values measured with the accelerometer can now be used to calculate the roll and pitch, see Eq. 4.4.

$$A_{pitch} = atan2\left(\frac{A_{x,LP}}{\sqrt{A_{y,LP}^2 + A_{z,LP}^2}}\right), \tag{4.4}$$
$$A_{roll} = atan2\left(\frac{A_{y,LP}}{\sqrt{A_{x,LP}^2 + A_{z,LP}^2}}\right).$$

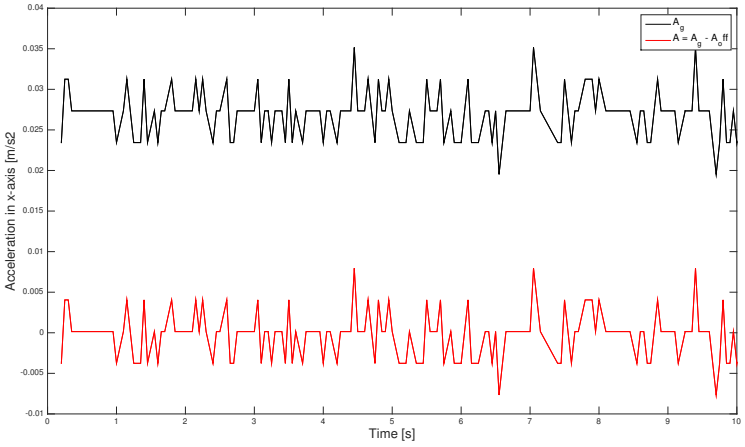



**Figure 4.3:** Comparison of the raw data from the accelerometer before (black) and after (red) the calibration. The offset is sub tracked from $\boldsymbol{A}_g$

.

### 4.3.2 Gyroscope

The raw data from the gyroscope *ITG3200* measures $\pm 2000 deg/sec$[20]. To be able to read useful and accurate values from the gyroscope the raw data is transformed to radians by Eq. 4.5. The gyroscopes resolution is 14.7 Least significant bit (LSB)'s per second), which means the maximum value will be $2000 * 14.7 \approx 29000$ which also is the maximum value possible with a signed 2-byte number.

$$\boldsymbol{G} = \boldsymbol{G}_{raw} \frac{1}{14.7} * \frac{\pi}{180}. \tag{4.5}$$

This signal is then integrated to obtain the angular position $\boldsymbol{G}_{Apos}$, see Eq. 4.6.

$$\boldsymbol{G}_{Apos} = \int \boldsymbol{G}(t)dt. \tag{4.6}$$

The gyro has one big disadvantage, called drift[22]. It means that the gyro tend to drift away from the correct value with time, this phenomenon is shown in Fig. 4.4. The solution to this problem lies in low-pass filtering the gyroscope signal together with the accelerometer values. This is further explained in section 4.4.

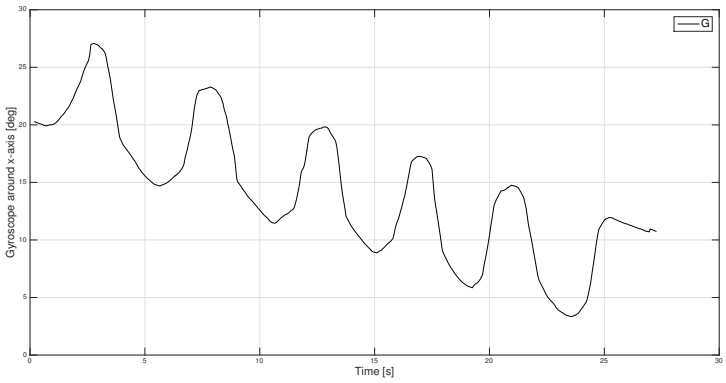



**Figure 4.4:** Illustration of the drift phenomenon, over time, when the gyroscope is rotated around the $z$-axis $\pm 5°$

### 4.3.3 Magnetometer

The raw data from the magnetometer *HCM5843* measures the magnetic fields around each axis in milli Gauss [$mGa$][21]. The sensor input field range is set to $\pm 1$ $Ga$ with gain 1300 $counts/mGa$ and the output range is $-2048$ $to$ 2047.
The offset on the magnetometer is divided into two sub offsets, hard-iron and soft-iron offset. They both need to be accounted for in order for the sensor to be able to provide reliable values. Hard-iron offsets correspond to external magnetic fields that affect the magnetometer, e.g. an additional constant magnetic field generated by a cell phone[23]. This is accounted for by correcting the center of measurements for all axis for $\boldsymbol{M}$ around 0, see Eq. 4.7.

$$M_{x,HIO} = \frac{1}{2}\Big(max(M_x) - min(M_x)\Big), \tag{4.7}$$
$$M_{y,HIO} = \frac{1}{2}\Big(max(M_y) - min(M_y)\Big),$$
$$M_{z,HIO} = \frac{1}{2}\Big(max(M_z) - min(M_z)\Big),$$

where the index $HIO$ stands for Hard-iron offset. This forms the vector $\boldsymbol{M}_{HIO} = [M_{x,HIO}\ M_{y,HIO}\ M_{z,HIO}]$. Fig. 4.5 demonstrates the influence of hard-iron offsets since the center of the ellipse is not located in the point (0,0,0).

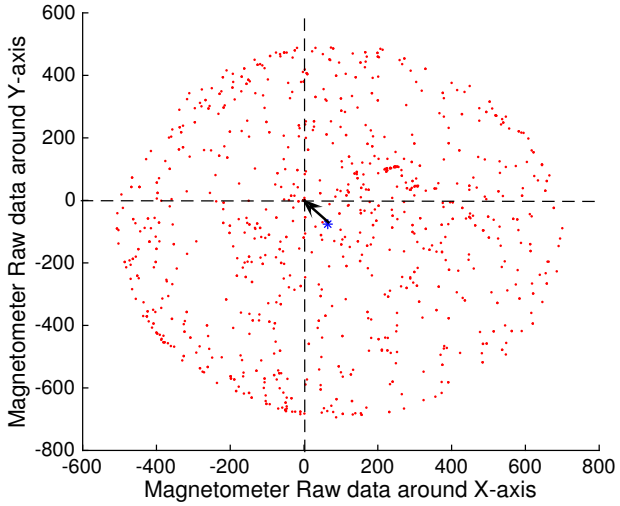



**Figure 4.5:** Illustration of the raw data from the magnetometer around the $y$- and $x$-axis before the calibration. The hard-iron errors are the reason the ellipse is not centered in $x = 0, y = 0$ and $z = 0$ and the soft-iron errors are the reason the data forms an ellipse instead of a sphere.

The soft-iron offset corresponds to the direction change and magnitude of the earth's magnetic fields when other ferromagnetic objects are in the vicinity of the sensor[23]. Materials such as kovar and steel are usually impart of such errors. Since the soft-iron errors are the errors that makes the plot of the magnetic field look elliptical (as can be seen from 4.5 the circle looks more like an egg), those errors are usually sorted out by using rotations matrices[24]. However, in this thesis, a different approach presented by Camel Software is used[25]. The elliptical shape is removed with a scaling factor instead

of using rotation matrices and the procedure is described below.

The average values for the hard-iron offsets, calculated in Eq. 4.7, are used to create a minimum and maximum vector for each axis by taking the minimum and maximum values and subtracting the average value. For the maximum vectors the procedure is shown in Eq. 4.8.

$$\begin{aligned} M_{x,SIOmax} &= max(M_x) - \frac{1}{2}\Big(max(M_x) - min(M_x)\Big), \\ M_{y,SIOmax} &= max(M_y) - \frac{1}{2}\Big(max(M_y) - min(M_y)\Big), \\ M_{z,SIOmax} &= max(M_z) - \frac{1}{2}\Big(max(M_z) - min(M_z)\Big), \end{aligned} \tag{4.8}$$

and for the minimum values respectively as shown in Eq. 4.9

$$\begin{aligned} M_{x,SIOmin} &= min(M_x) - \frac{1}{2}\Big(max(M_x) - min(M_x)\Big), \\ M_{y,SIOmin} &= min(M_y) - \frac{1}{2}\Big(max(M_y) - min(M_y)\Big), \\ M_{z,SIOmin} &= min(M_z) - \frac{1}{2}\Big(max(M_z) - min(M_z)\Big). \end{aligned} \tag{4.9}$$

The next step is to find the distance from the center, therefore the negative values are inverted and the average distance from the center on each axis is calculated. The calculations for the $x$-component is show below but the same procedure holds for the $y$- and $z$-components as well, see Eq. 4.10

$$\bar{x} = \frac{1}{2}\Big(M_{x,SIOmax} + M_{x,SIOmin}\Big). \tag{4.10}$$

Now the components are averaged between the three axes according Eq. 4.11.

$$\bar{M} = \frac{1}{3}\Big(\bar{x} + \bar{y} + \bar{z}\Big) \tag{4.11}$$

From Eq. 4.11 the scaling factor on each axis, $\tilde{x}, \tilde{y}$ and $\tilde{z}$ are now calculated and combined in the vector $\boldsymbol{M}_{SIO}$, see Eq. 4.12.

$$\boldsymbol{M}_{SIO} = \begin{bmatrix} \tilde{x} = \bar{M}/\bar{x} \\ \tilde{y} = \bar{M}/\bar{y} \\ \tilde{z} = \bar{M}/\bar{z} \end{bmatrix}, \tag{4.12}$$

and is multiplied with the magnetometer readings before the hard-iron offset is subtracted as done in Eq. 4.13.

$$\boldsymbol{M} = \boldsymbol{M}_{raw} \times \boldsymbol{M}_{SIO} - \boldsymbol{M}_{HIO}. \tag{4.13}$$

The results from the calibrated magnetometer is shown in Fig. 4.6 and it can be seen that the hard- and soft-iron offsets have been compensated for. Thus the center of the spheres now is located in (0,0,0) and the elliptical shape of the sphere is as good as gone.

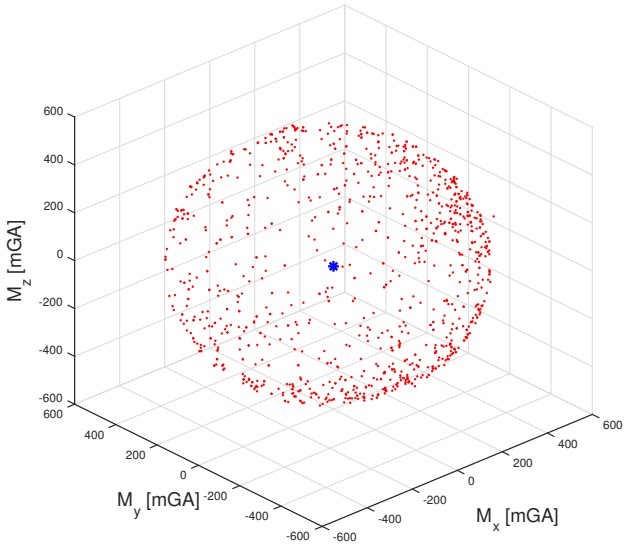



**Figure 4.6:** Illustration of the shape of the data given from the magnetometer after calibration.

Before using the data from the magnetometer the signal is normalized, see Eq. 4.14.

$$\boldsymbol{M}_{norm} = \boldsymbol{M} \frac{1}{\sqrt{M_x^2 + M_y^2 + M_z^2}}. \tag{4.14}$$

The magnetometer can now measure an accurate yaw angle if the sensor is placed horizontal in the $xy-$plane described in section 4.3.1 with no roll and pitch. Thus this application is placed on the RR the roll and pitch can not be neglected. This requires measurements from the gyro and the accelerometer to be combined with the magnetometer readings in order to make the yaw angle, $\phi$, tilt compensated.

## 4.4 Filtering

In the following section the procedure of filtering the signals are more thoroughly described. Starting with the low-pass filtering of the accelerometer signals for removing the noise. Continuing with the complementary filtering of the gyroscope and accelerometer signals for obtaining stable pitch and roll angles without drift. Ending this section by combining the filtered signals together with the normalized magnetometer values in order to calculate a tilt compensated yaw angle, $\psi$.

### 4.4.1 Low-pass filtering

Even after the sensors are calibrated the data can still be noisy. In order to get rid of the noise, the signals are filtered as a step to get a smoother signal. For the *ADXL345* accelerometer and *HMC5843* magnetometer a low-pass filter is used for this purpose, see Eq. 4.15

$$A_{cali} = A_{cali} \times \alpha + (A_{prev\ cali} \times (1 - \alpha)), \tag{4.15}$$

where $\alpha \in [0,1]$ decides how much of the signal that should be filtered[22]. If $\alpha$ has a high value ($\approx 1$) then the filter just relay on the current measure and the filter is said to be nor or less inactive. If $\alpha$ is close to 0, then instead only rely on the previous measure and the behavior will be very slow. An arbitrary value for this application is chosen to $\alpha = 0.5$. The result of the low-pass filtering for the accelerometer in the $x$-direction can be seen in Fig. 4.7.

It is clear that the filtered signal, the thick red line is smoother than the non-filtered signal. This example demonstrates the $x$-axis from the accelerometer but the same filtering is done for all three axes of the accelerometer. A low-pass filtering is also made on the magnetometer readings since its raw data is also noisy.

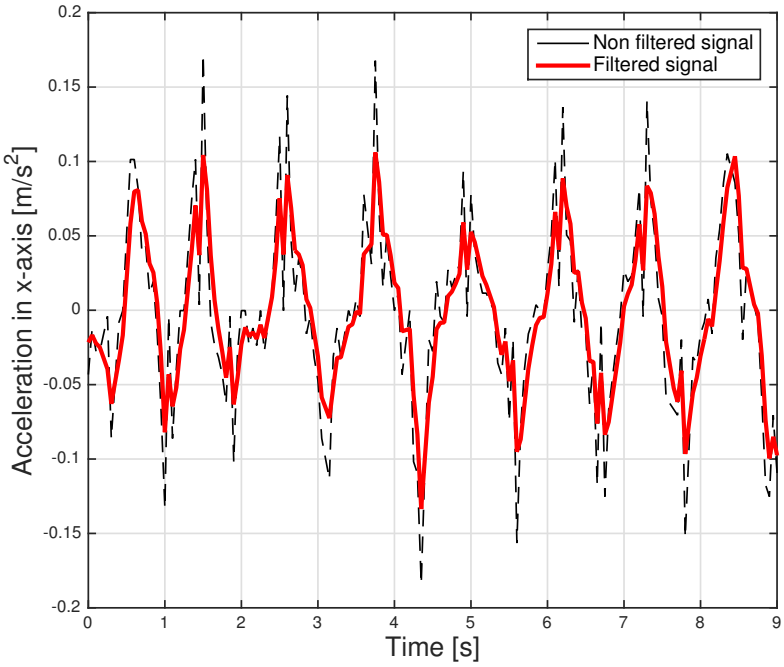



**Figure 4.7:** Comparison of the accelerometer data on the $x$-axis before and after the low-pass filtering. The blue dashed signal is the signal before the filtering and the thick red line is the smoother signal after the filtering.

### 4.4.2 Complementary filtering

To prevent the drift phenomenon from the gyroscope a complementary filter needs to be added. This filter combines the roll and pitch values from the accelerometer and the gyroscope and creates a more stable, non-drifting behavior. The complimentary filter have many similarities with the low-pass filter, see Eqs. 4.16 and 4.17, where the roll $\phi$ and pitch $\theta$ are calculated from the accelerometer and the gyroscope.

$$\phi = (1 - \beta)A_{roll} + \beta G_{x,Apos}, \tag{4.16}$$

$$\theta = (1 - \zeta)A_{pitch} + \zeta G_{y,Apos}, \tag{4.17}$$

where $\beta$ and $\zeta \in [0,1]$. In several applications a Kalman filter is used for this purpose but the authors of this thesis were satisfied with the readings from the complementary filtered signals and considered them sufficient. A Kalman Filter could be more powerful,

but more computationally expensive, hence for an application using an Arduino board a complementary filter is more preferable[26].

## 4.5 Tilt compensation

Finally, the readings from the accelerometer, the gyroscope and the magnetometer can be combined to derive the tilt compensated yaw angle[22], see equation 4.18 and 4.19 form the Eq. 4.20

$$TCx = M_{x,norm}cos(\theta) + M_{y,norm}sin(\phi)sin(\theta) + M_{z,norm}cos(\phi)sin(\theta), \tag{4.18}$$

$$TCy = M_{y,norm}cos(\phi) + M_{z,norm}sin(\phi), \tag{4.19}$$

$$\psi = atan2\left(\frac{TCx}{TCy}\right). \tag{4.20}$$

# 5

# Communication

This chapter presents the basic theory behind the communication between the RR and the EB and how it is implemented. The hardware is built using Arduino Mega [17] microcontrollers, with additional shields attached for enabling necessary applications. The communication is built both on a wire setup over Ethernet and wireless communication over Wi-Fi, using User Datagram Protocol (UDP). The navigation of the RR is controlled in a Graphical User Interface (GUI) program.

## 5.1 Hardware

The main hardware that was used in this application is shown in Tab. 5.1. Both the RR and the EB are equipped with an Arduino Mega 2560 board. Additional to this, multiple shields were connected to enabling the boards to collect GPS data, enable handlebar steering, switching between manual/automatic steering and throttle. A full description of the hardware implementation can be seen in the twin project report [27], which was carried out as a B.Sc. thesis at Chalmers University of Technology during the first half of 2015.

**Table 5.1:** Hardware implementation where the table specifies which components are used at each vessel.

| | RR | EB | Purpose |
|---|---|---|---|
| Arduino Mega 2560 | YES | YES | Motherboard |
| Arduino Wi-Fi shield | YES | NO | Send data over Wi-Fi |
| Arduino Motor shield | YES | NO | Enables motor steering.<br>Enable Manual/Auto mode.<br>Steer the handlebar |
| Arduino Ethernet shield | NO | YES | Send data over Ethernet |
| Arduino GPS shield | YES | YES | Collects GPS data |
| Songle Relay board<br>5V 8-Channel | YES | NO | Throttle: Manual/Auto<br>Handlebar Left/ Right. |
| Sparkfun IMU sensor | YES | NO | Digital compass |
| Pulse sensor (IR) | YES | NO | Measure the handlebars<br>motor revolutions |
| DLink WirelessN 150<br>Home Router DIR-600 | NO | YES | Setting up the network |

## 5.2 Arduino software implementation

As there are two separate Arduinos, one in the RR and one in the EB, they have been programmed different. Explanations of the programs are provided in Appendix B.

## 5.3 User Datagram Protocol communication

UDP makes it possible to send information from one unit to another unit over a network. In this application the UDP will send strings of text containing information about the GPS position, speed and heading angle from both the RR and the EB. With this information known it is possible to derive a proper heading angle and throttle for the RR. A schematic overview of how the application is built can been seen in Fig. 5.1.

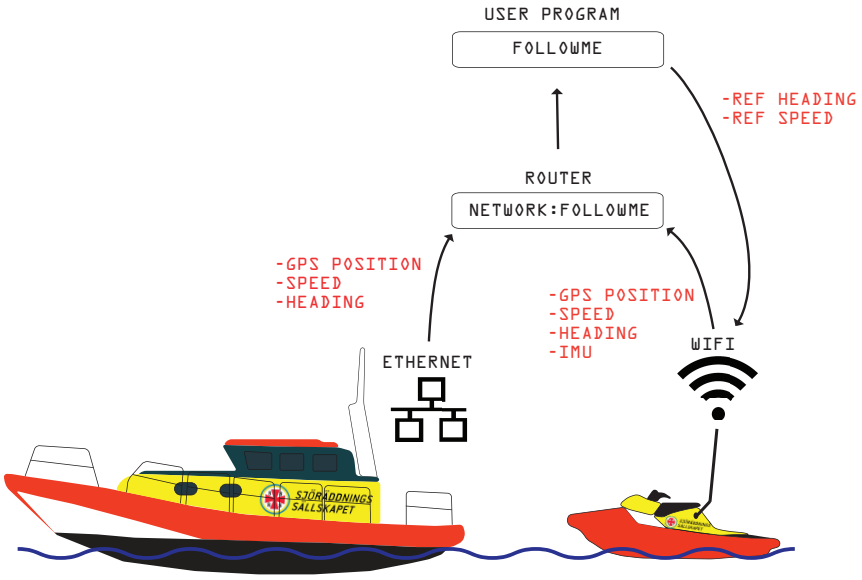



**Figure 5.1:** Schematic overview of the communication between the RR and the EB.

The RR is equipped with a wireless device, sending information from its GPS and the IMU. This is information is merge into a string `RR_getData`, see string below,

```
RR_getData = Longitude/Latitude/Speed/HeadingAngle/HeadingAngleIMU
```

This string is sent to GUI via a router. The EB on the other hand will not use the wireless communication as the RR does, instead the data is sent through Ethernet to the router and the further to the GUI. Thus the router is also placed on the EB. This string is called `EB_getData`, see string below,

```
EB_getData = Longitude/Latitude/Speed/HeadingAngle
```

Note, that the EB is not equipped with an IMU as the RR were. The GUI will now continuously receive strings (packages) with this information from both the RR and the EB.

## 5.4 Graphical User Interface

As mentioned in section 5.3 a program, where the user can chose between two separate settings (modes) additional to the regular *Manual* mode, has been developed, see Fig. 5.2.

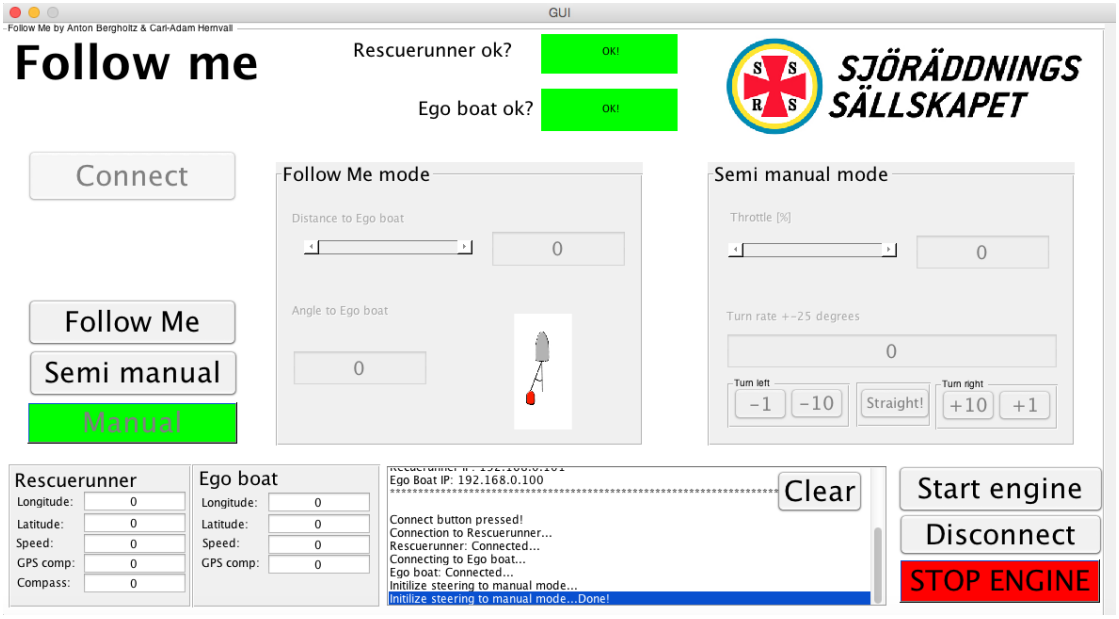



**Figure 5.2:** Overview of the GUI that is used for running the different applications programmed into the RR on a computer.

Before the RR can run, the network needs to be up and ready. By pushing the button *Connect*, see Fig. 5.2 the program checks if the RR's and EB's communication hardware are online. The program sends a message to both vessels asking if they are connected. If they are available they will answer with an acknowledgement telling they are okey.

If both vessels were successfully connected the program continues. As long as the user is connected, the user has the ability to stop the engine at any time by pressing the *STOP ENGINE* button, see Fig. 5.3. This simply shuts down the engine and sends an acknowledgement to the GUI telling the engine is off. The engine can be started again by pressing the *Start engine* button, running the starting engine for four seconds. When the engine is on, an acknowledgement is sent, telling the engine is on. If the user want to disconnect completely from the RR and the EB the user simply press *Disconnect*. This

will put the RR in *Manual* mode before the program shuts down the connection between the units.

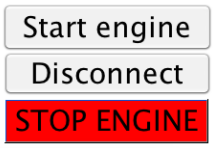



**Figure 5.3:** Illustration of buttons in the GUI that is a part of Fig. 5.2.

However, now the system is connected and in *Manual* mode. This manual button is green, see Fig. 5.4.


**Figure 5.4:** Illustration of application setup, where the modes can be switch by pressing the buttons. This figure is a part of Fig. 5.2.

There are two additional modes beyond the *Manual* mode, e.i. *Follow me* and *Semi manual*. The first one is the application where the RR should follow the path of the EB, copying its heading and speed. The user can chose which distance and angle to the EB the RR should have. This is controlled by two controllers (see chapter Controller - 3.4), which regulates the heading angle and throttle. The RR is now said to be in *Follow me* mode and will follow the EB by itself until another command has been given by the user. Note, the *Follow me* button is green.

If the user wants to stop the *Follow me* application it is simply done by pressing the *Manual* button. This will stop the application and put the RR into *Manual* mode. Note that the program is still running, i.e. the program is connected. From here the user can select the other modes, e.g. *Semi manual*. This application has the purpose of maneu-

vering the RR by commands given from the user directly, i.e. this mode will not follow the EB directly. Instead the user can chose the heading angle and speed, maneuvering the RR in a specific path.

When using *Semi manual* and *Follow me* mode the navigation parameters `throttle` and `steering` are sent approximately three times every seconds. This means if the RR is able to receive the commands, the navigation parameters will be updated three times every second. When the RR receives a command it does not resend an acknowledgement message back to the GUI telling the commands where received. Instead the RR only updates the navigation parameters with a new `throttle` and a new `steering`.

The idea is that when combining all three modes, *Manual*, *Follow Me* and *Semi manual* the RR can first be set in *Semi manual* to navigate out from harbor. As soon as the boat is on open water the user switches over to *Follow me* mode and the RR follows the EB until the scene of accident has been reached. Again, the user switches back to *Semi manual* mode and navigates the RR next to the EB. After this, when the RR is boarded to the EB the rider change to *Manual* mode and start the rescue mission. When the mission is done and it is time to return, the user simply press *Follow me* button again and the RR will follow the EB back to the harbor.

Further explanation of how the Graphical User Interface (GUI), i.e. the user program works and the functions it is invoking can be found in Appendix C.

# 6

# Results & Analysis

In the follow chapter the results from this thesis will be presented. That pertains both the results from the simulated system and the tests from the implementation of the control system in the actual RR. Initially the different tests from the simulated system will be described and properly commented and thereafter the same will be done for the actual implementation.

## 6.1 Simulation results

For the simulated environment the tests has been performed using the MATLAB/Simulink software. The tests was performed to check the ability of the RR how well it could follow a path of predefined WPs (as described in section 3.2). The tests pertains to both different parameters set on speed limits, but also on the RoA that the RR has to reach, before its target is shifted to a new WP. The main focus when performing the tests, were to investigate how the model was affected by changing the control algorithms, i.e.:

- Different limits set on the maximum speed, $v_{max}$ according to a normally distributed curve (as described in section 3.2.4).
- Different RoAs on the WPs the RR has to follow.

The speed and the RoA are key parameters that affects the behavior of the system to a large extent. Imagine the case where the RoA is set to two different values. A high

value denoted, $r_h$, and a low value denoted, $r_l$. This means, when the higher value is used, the RR will be able to pass through almost all WPs without any trouble and the speed can be set to a higher value. Thus the RR starts to aim at the next WP once the RoA is reached. However, the path taken by the RR will not be as precise as if the RoA is set to a lower threshold value, i.e. $r_l$.

In this section the results of choosing a number of different values on those parameters will be thoroughly evaluated and commented. The results will be collected from two different paths, one circular-shaped and the other one is zigzag-shaped. On each path, six tests will be performed, see Tab. 6.1. The minimum allowed speed is constant and set to $v_{min} = 2.1$m/s. This limit is set due to the fact when the RR is idle, the speed is around 2.1 m/s.

**Table 6.1:** Overview of the tests that will be performed with the simulation model of the system where the RR follows consecutive WPs, released by the ego-boat.

| **Test** | **Path** | $\boldsymbol{v_{max}[m/s]}$ | $\boldsymbol{r_a[m]}$ |
|---|---|---|---|
| 1 | Circular | 5 | 5 |
| 2 | Circular | 10 | 5 |
| 3 | Circular | 5 | 8 |
| 4 | Circular | 10 | 8 |
| 5 | Circular | 5 | 15 |
| 6 | Circular | 10 | 15 |
| 7 | Zigzag | 5 | 5 |
| 8 | Zigzag | 10 | 5 |
| 9 | Zigzag | 5 | 8 |
| 10 | Zigzag | 10 | 8 |
| 11 | Zigzag | 5 | 15 |
| 12 | Zigzag | 10 | 15 |

### 6.1.1 Simulation test 1: $v_{max} = 5m/s$, $r_a = 5m$

Starting with the circular path, the first test will be done with the following values on the key parameters:

- $v_{max} = 5$m/s,
- $r_a = 5$m.

The result from the simulation can be seen in Fig. 6.1.

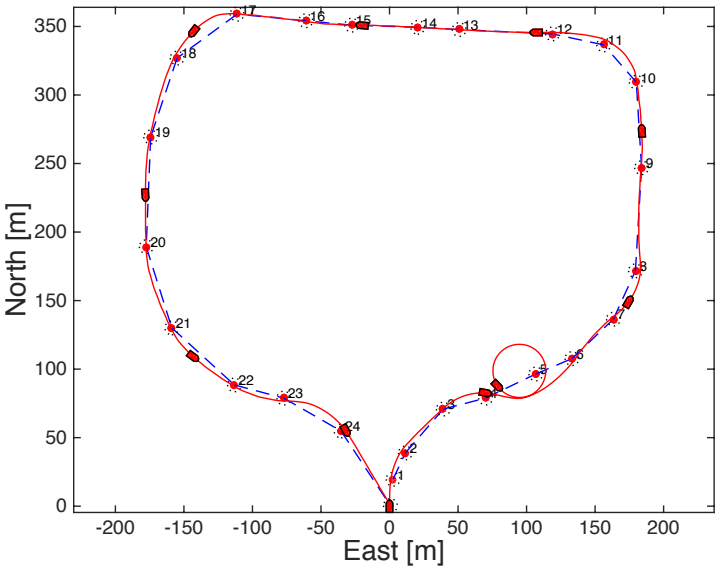



**Figure 6.1:** Illustration of how the RR follows WPs released by the EB, with key parameters set to $v_{max} = 5$ m/s and $r_a = 5$ m.

It can be seen from the Fig. 6.1 that RR manages to follow the path in a good way as long as the path is straight. When the path turns quickly it misses a WP. The controller does not manage to correct the steering angle enough for the RR to reach the RoA of the next . This is what happend at WP 5 in Fig. 6.1. In Fig. 6.2 the speed profile for the simulation is shown.

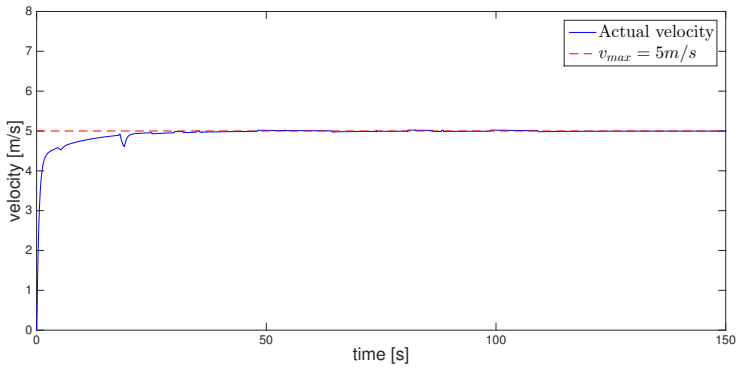



**Figure 6.2:** Illustration of the speed profile for the RR for the path shown in Fig. 6.1.

It can be seen that the actual speed of the RR approaches the maximum speed allowed, $v_{max}$. The reason the RR does not immediately accelerates is dependent on the PID-controller is setting the speed. Until the RR has driven approximately 30m the steady state error is not completely removed by the integral action. The dips in the curve symbolizes what happens when the RR makes a turn, therefore consequently the PID-controller decreases the velocity. Since the path is not very rigorous, the controller allows the RR to drive at the maximum allowed velocity, i.e., $v_{max}$ for almost the whole simulation. It shall be noted that the RoA, $r_a$, is set to a low value, which is a hard constraint on the system.

### 6.1.2 Simulation test 2: $v_{max} = 10m/s$, $r_a = 5m$

For the second test the key parameters are set to:

- $v_{max} = 10$m/s,
- $r_a = 5$m.

The value of $r_a$ remains unchanged, while the value of $v_{max}$ is increased by 5m/s. The result of the RR following the path with increased speed is shown in Fig. 6.3.

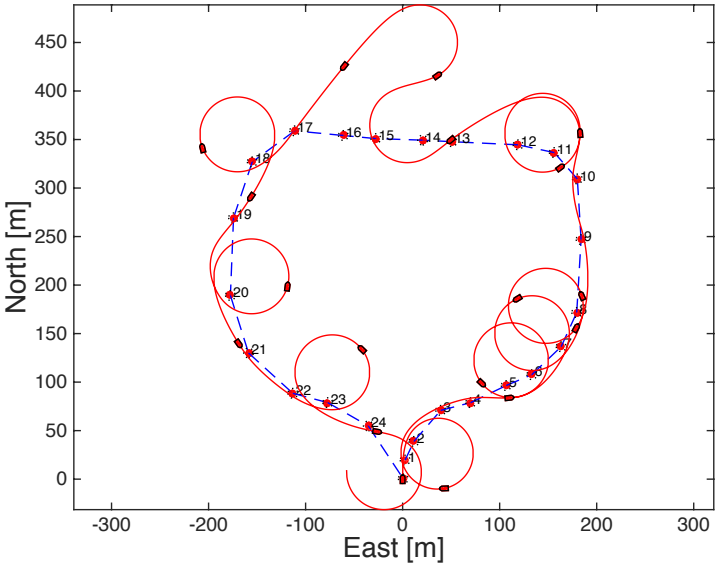
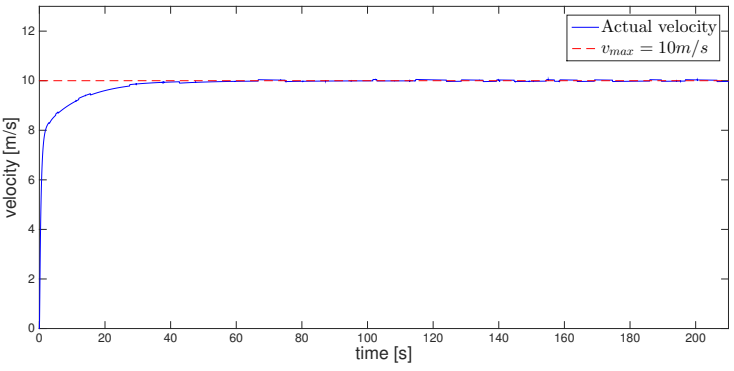



**Figure 6.3:** Illustration of how the RR follows WPs released by the EB for the key parameters set to $v_{max} = 10$ m/s and $r_a = 5$ m.

Fig. 6.3 illustrates that the RR does not manage to follow the desired path in a satisfactory way with these settings. Several WPs are missed. This is not understood by the controller, which interprets the misses as if the RR is driving on a straight line. Hence, the speed is set almost constant as $v_{max} = 10$ m/s. The behavior is illustrated in Fig. 6.4.




.

**Figure 6.4:** Illustration of the speed profile for the RR for the path shown in figure 6.3.

Note, a distinct connection between the values set on the maximum speed $v_{max}$ and the RoA $r_a$ can be observed.

### 6.1.3 Simulation test 3: $v_{max} = 5m/s$, $r_a = 8m$

For the third test the RoA $r_a$ will be changed to the larger value 8 m, i.e. the key parameters are set to:

- $v_{max} = 5$ m/s,
- $r_a = 8$ m-

In Fig. 6.5 it can be seen that the RR manages to follow the desired path in a good way when the maximum speed is set to $v_{max} = 5$ m/s.

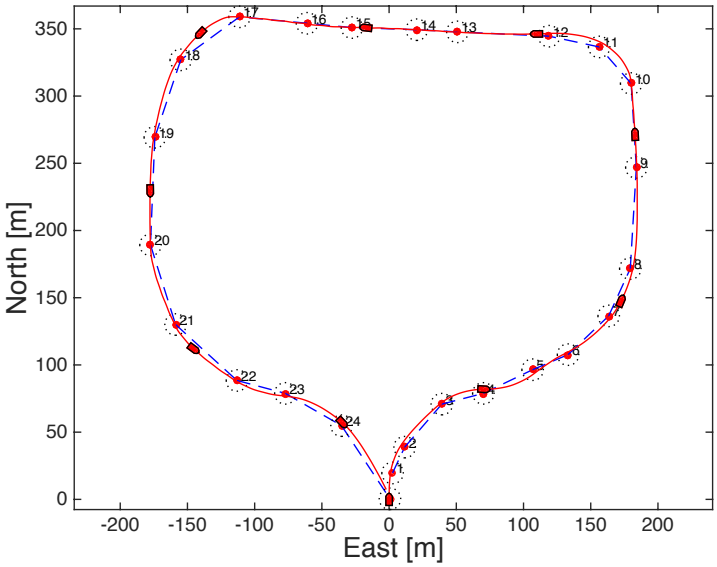



**Figure 6.5:** Illustration of how the RR follows WPs released by the EB for the key parameters set to $v_{max} = 5$ m/s and $r_a = 8$ m.

However, from the speed profile shown in Fig. 6.6 it can be seen that when the RoA, $r_a$, is set to a larger value the RR starts aiming at the next WP at an earlier moment in time.
Hence, the speed profile get more bumps, which is a result of this since the controller is lowering the speed in order to manage to reach the new WP.

### 6.1.4 Simulation test 4: $v_{max} = 10m/s$, $r_a = 8m$

For the fourth test on the circular path the values of the key parameters are set to:

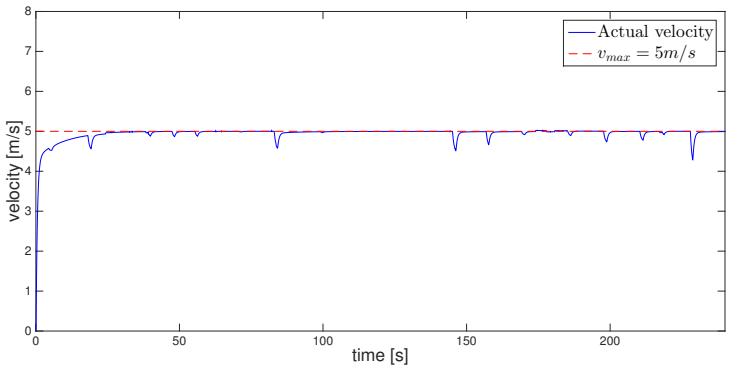
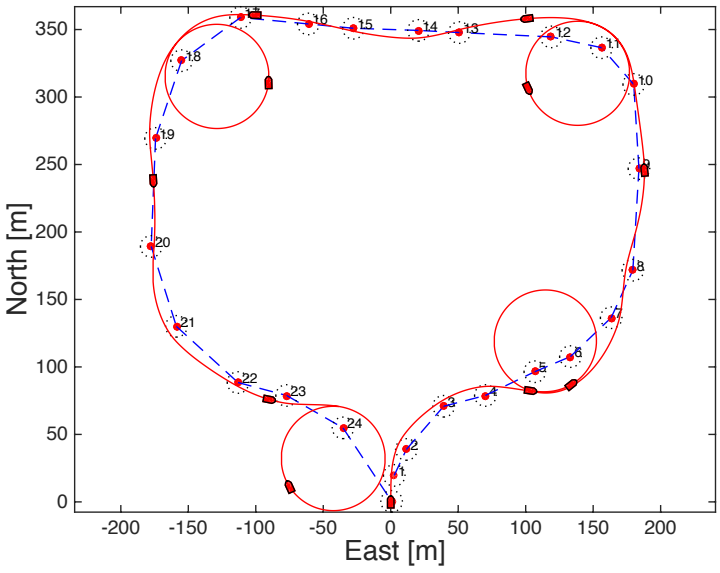



**Figure 6.6:** Illustration of the speed profile for the RR for the path shown in Fig. 6.5.

- $v_{max} = 10$m/s,
- $r_a = 8$m.

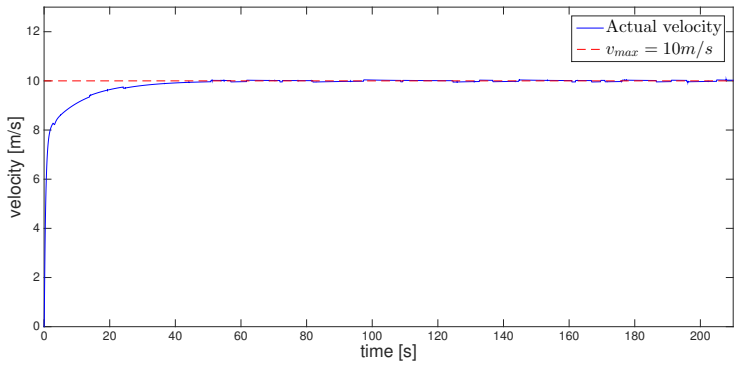
The result can be observed in Fig. 6.7. The figure illustrates that the RR manages to follow the path in a robust way than when the value of $r_a$ is increased alongside the value of $v_{max}$. It misses four of the WPs but manages to stabilize its path after the fourth WP is missed.




.

**Figure 6.7:** Illustration of how the RR follows WPs released by the EB for the key parameters set to $v_{max} = 10$ m/s and $r_a = 8$ m.

Basically, the controller allows the RR to drive at $v_{max}$ throughout the whole simulation, as can be seen from Fig. 6.8




.

**Figure 6.8:** Illustration of the speed profile for the RR for the path shown in Fig. 6.7.

### 6.1.5 Simulation test 5: $v_{max} = 5$ m/s, $r_a = 15$ m

For the last two tests on the circular path the value key parameters are set to:

- $v_{max} = 5$ m/s,
- $r_a = 15$ m.

The results are shown in Fig. 6.9 and Fig. 6.10 and it can be seen that the RR manages to follow to desired path almost perfectly.

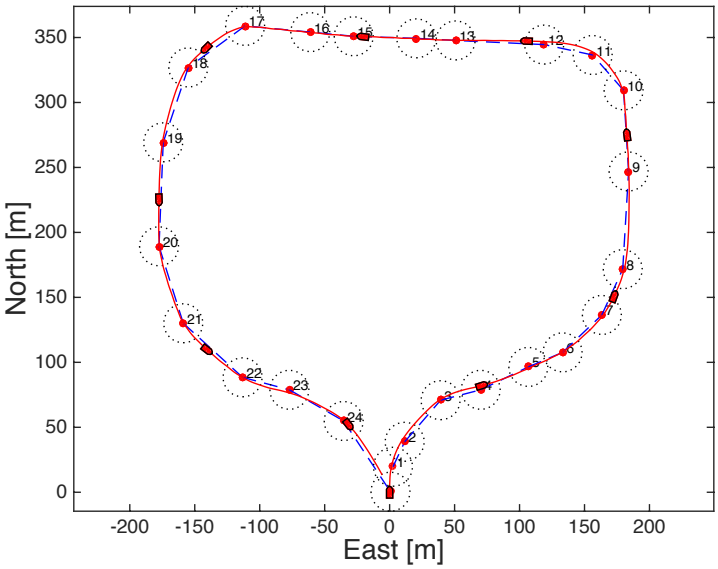



**Figure 6.9:** Illustration of how the RR follows WPs released by the EB for the key parameters set to $v_{max} = 5$ m/s and $r_a = 15$ m.

This is a combination of the maximum allowed velocity set to a lower value and the RoA set to a comparatively large value. As mentioned in Test 3, a larger threshold value on RoA causes more bumps in the speed profile. Simply because the RR need to turn more often to aim at the consecutive WP. Comparing the speed profiles for the maximum speed $v_{max}$ to 5 m/s, it can be observed that the bumps are bigger when the RoA is set to 15 m, see Fig. 6.10 compared with the value set to 8 m/s as shown in Fig. 6.6.

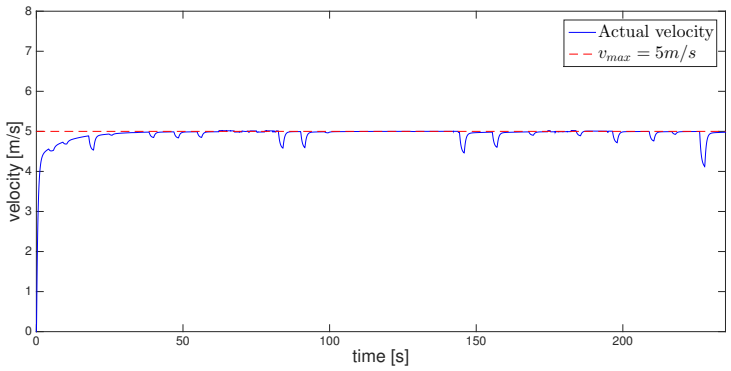



**Figure 6.10:** Illustration of the speed profile for the RR for the path shown in Fig. 6.9.

This is a consequence of what happens when the value of $r_a$ increases. The RR enters the RoA around a specific WP more often than if this parameter is given a lower threshold value.

### 6.1.6 Simulation test 6: $v_{max} = 10$ m/s, $r_a = 15$ m

For the last test on the circular path the values of the key parameters $v_{max}$ and $r_a$ are set to

- $v_{max} = 10$ m/s,
- $r_a = 15$ m.

The obtained result is visualized in 6.11, which shows that this is the first test when the RR manages to follow the circular path without missing a WP for the larger value set on the maximum speed velocity, $v_{max}$.

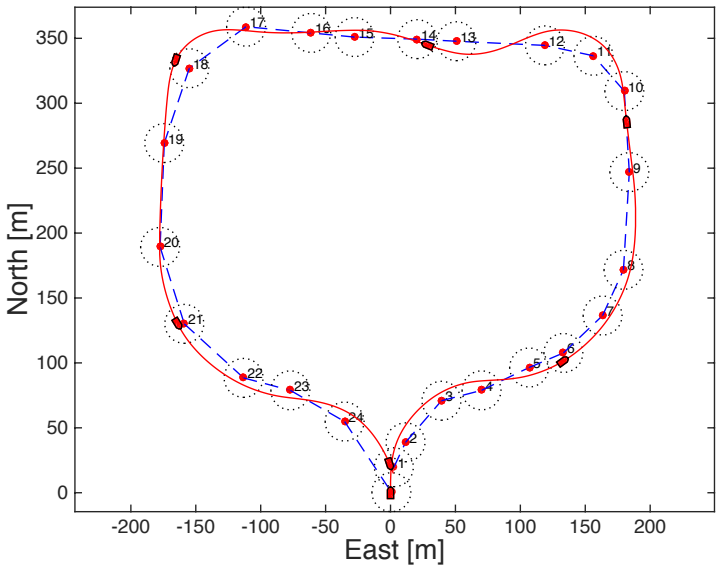



**Figure 6.11:** Illustration of how the RR follows WPs released by the EB for the key parameters set to $v_{max} = 10$ m/s and $r_a = 15$ m.

It can also be observe from the speed profile in Fig. 6.12 that in this test it takes more time for the steady-state error to be removed for the maximum speed than in previous tests.

.

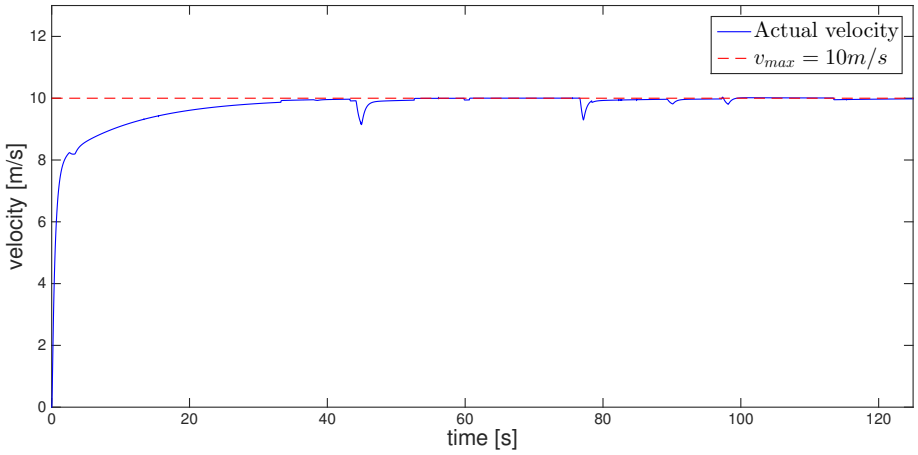



**Figure 6.12:** Illustration of the speed profile for the RR for the path shown in Fig. 6.11.

It also shows that the controller is working properly, since the bumps in the velocity curve are present in this test. It visualizes how the controller is adapting the motion of the RR in a way that makes it follow the desired path in a satisfactory way.

### 6.1.7 Simulation test 7: $v_{max} = 5$ m/s, $r_a = 5$ m

The last six tests form the simulations are performed on a zigzag-shaped path. The key parameters that are varied are the same is the first 6 tests, i.e. the maximum allowed speed $v_{max}$ and the RoA $r_a$. For the first test on the zigzag-shaped path the values on the key parameters are set to:

- $v_{max} = 5$ m/s,
- $r_a = 5$ m.

From Fig. 6.13, it can be observed that the RR manages to follow the path.
However, observing the speed profile shown in Fig. 6.14 the bumps in this curve are much bigger than for the tests performed on the circular path. This shows that in order to be able to pull of the sharp turns, the velocity has to be decreased even more than on the circular path.

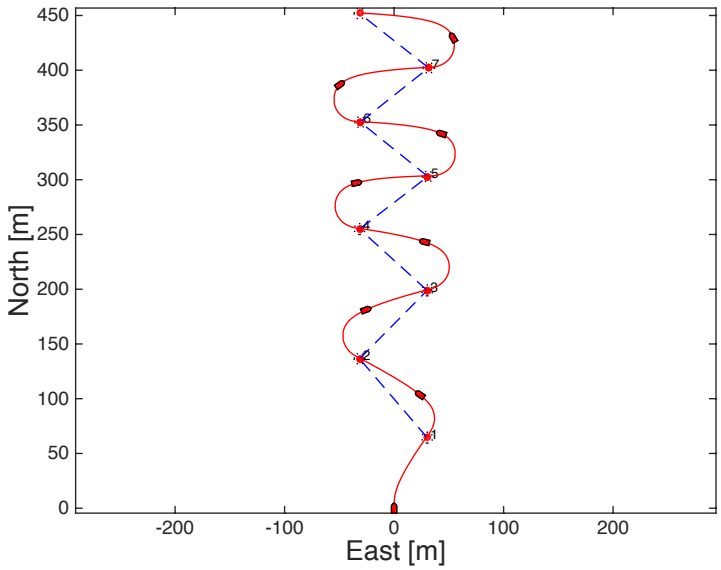



**Figure 6.13:** Illustration of how the RR follows WPs released by the EB for the key parameters set to $v_{max} = 5$ m/s and $r_a = 5$ m.

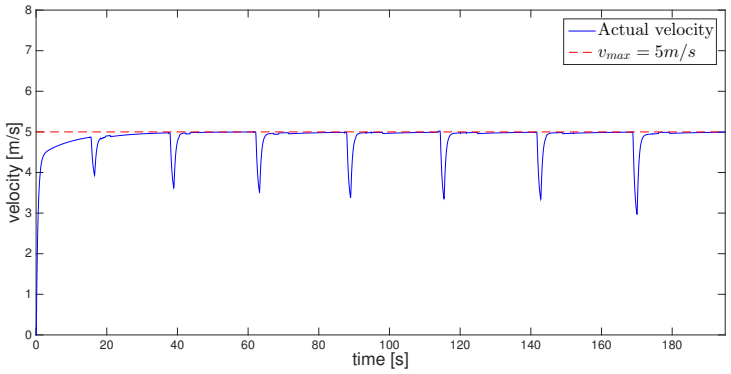



**Figure 6.14:** Illustration of the speed profile for the RR for the path shown in Fig. 6.13.

### 6.1.8 Simulation test 8: $v_{max} = 10$ m/s, $r_a = 5$ m

For the second test on the zigzag-shaped path the value of the key parameters are set to the values:

- $v_{max} = 10$ m/s,
- $r_a = 5$ m.

Clearly from Fig. 6.15 it can be seen that the combination of the maximum velocity set to 10m/s and the comparatively small value set on the RoA, the path becomes to sharp for the controller to handle and at the last two WPs, the RR does not even manage to follow a path close to the desired path.

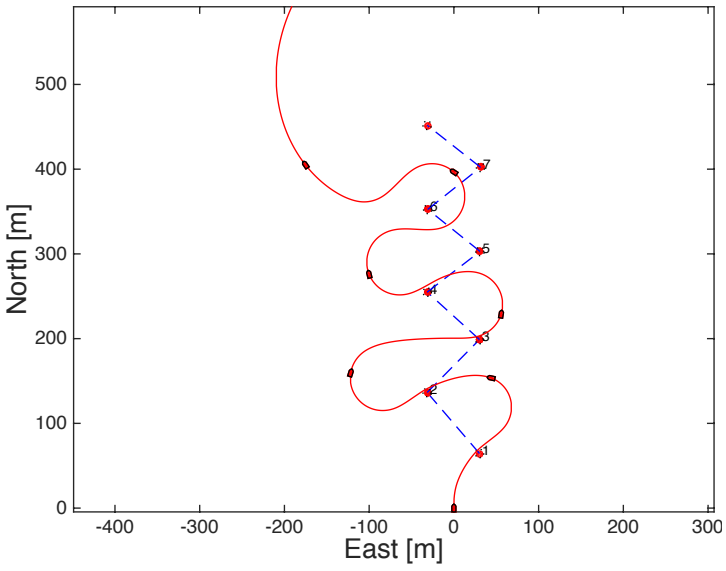



**Figure 6.15:** Illustration of how the RR follows WPs released by the EB for the key parameters set to $v_{max} = 10$ m/s and $r_a = 15$ m.

From the speed profile in Fig. 6.16, small deflections can be seen for the first WPs. When the controller does not manage to follow the WPs due to the clingy value set on $r_a$, the RR accelerates to the maximum velocity, constantly trying to find the next WP.

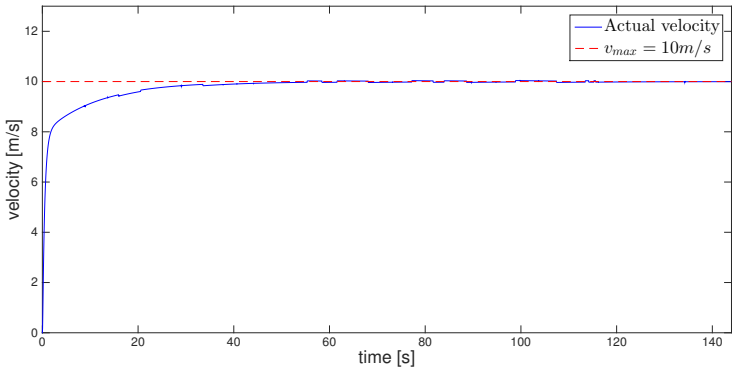



**Figure 6.16:** Illustration of the speed profile for the RR for the path shown in Fig. 6.15.

### 6.1.9 Simulation test 9: $v_{max} = 5$ m/s, $r_a = 8$ m

As for the tests on the circular path the tests on the zigzag-shaped path continues with the same setups. The key parameters are now set to :

- $v_{max} = 5$ m/s,
- $r_a = 8$ m.

In Fig. 6.17 it is shown that the RR still manages to follow the zigzag-shaped path in satisfactory way, even with the increased RoA.

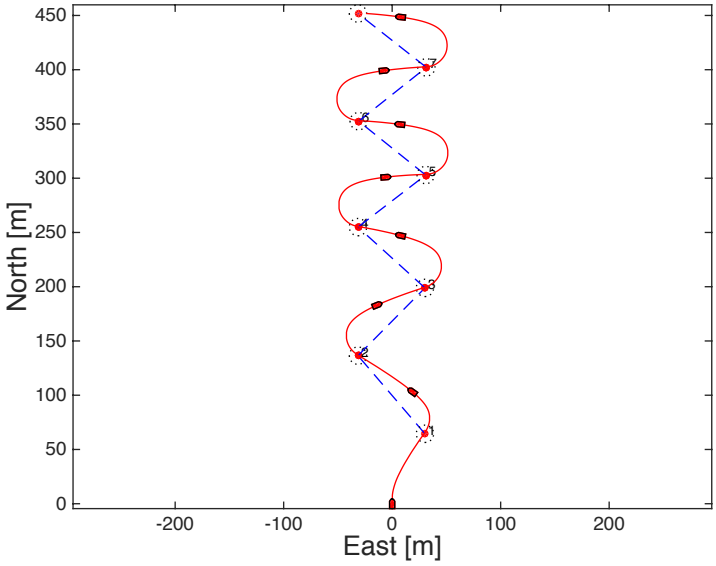



**Figure 6.17:** Illustration of how the RR follows WPs released by the EB for the key parameters set to $v_{max} = 5$ m/s and $r_a = 8$ m.

Similarly to tests performed on the circular path, i.e. as shown from Fig. 6.6 and Fig. 6.10 the bumps where the speed is decreased are deeper with the increased value on the RoA, $r_a$. Once again, the reason for this is the fact that the RR starts aiming at a new WP at an earlier moment in time. This causes the controller to change the heading and speed more rapidly.

### 6.1.10 Simulation test 10: $v_{max} = 10$ m/s, $r_a = 8$ m

For the key parameters on the zigzag-shaped path chosen as:

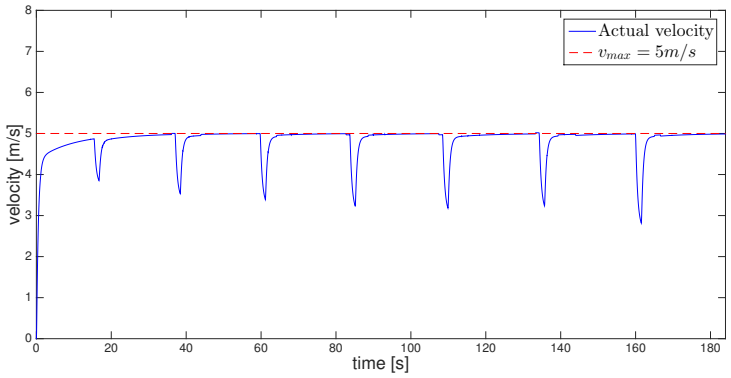



**Figure 6.18:** Illustration of the speed profile for the RR for the path shown in Fig. 6.17.

- $v_{max} = 10$ m/s,
- $r_a = 8$ m.

The results are shown in Fig. 6.19.

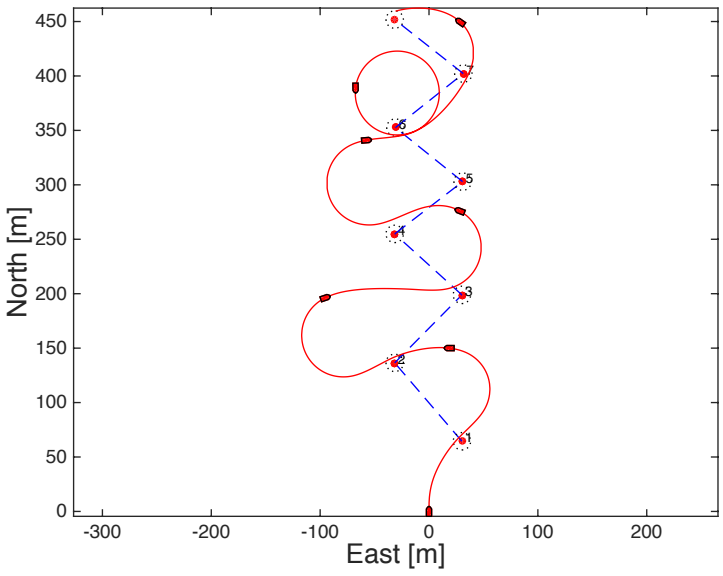
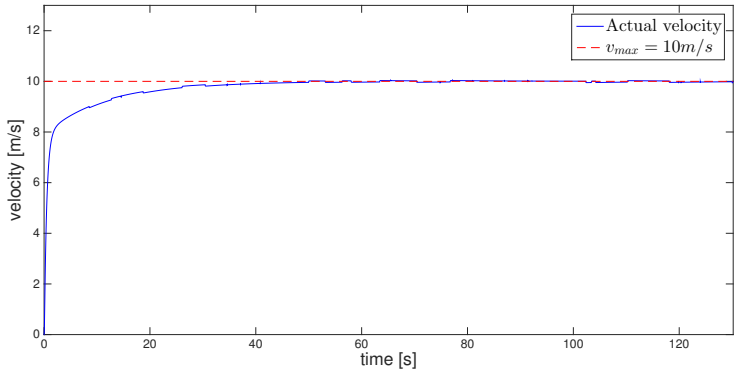



**Figure 6.19:** Illustration of how the RR follows WPs released by the EB for the key parameters set to $v_{max} = 10$ m/s and $r_a = 8$ m.

Even though the maximum speed allowed is set to a higher value, the RR manages to reach the first two WP. It then lose track of the desired path but manages to re-establish it at the seventh WP. Finally it reaches the eighth WP. From Fig. 6.20 small deflections

are seen on the speed profile for the earlier points, but then when the controller does not longer manage to re-establish the path it tries to compensate for this behavior. Hence, accelerating the RR up to the maximum speed.




.

**Figure 6.20:** Illustration of the speed profile for the RR for the path shown in Fig. 6.19.

### 6.1.11 Simulation test 11: $v_{max} = 5$ m/s, $r_a = 15$ m

For the second last test, the zigzag-shaped path the key parameters are set to:

- $v_{max} = 5$ m/s,
- $r_a = 15$ m.

From Fig. 6.21 it can be observed that the RR manages to follow the desired path in a satisfactory way. As predicted in Fig. 6.22 shows that bumps in the speed profile are even larger, because the RR reaches the desired WPs at earlier and earlier moments in time.

### 6.1.12 Simulation test 12: $v_{max} = 10$ m/s, $r_a = 15$ m

For the last test on the zigzag-shaped path the value of the key parameters are set to:

- $v_{max} = 10$ m/s,
- $r_a = 15$ m/s.

From Fig. 6.23 it can be observed that the RR manages to follow the desired path for the first three WPs and then it looses the path, which it manages to re-establish for the sixth, seventh and eighth WP.

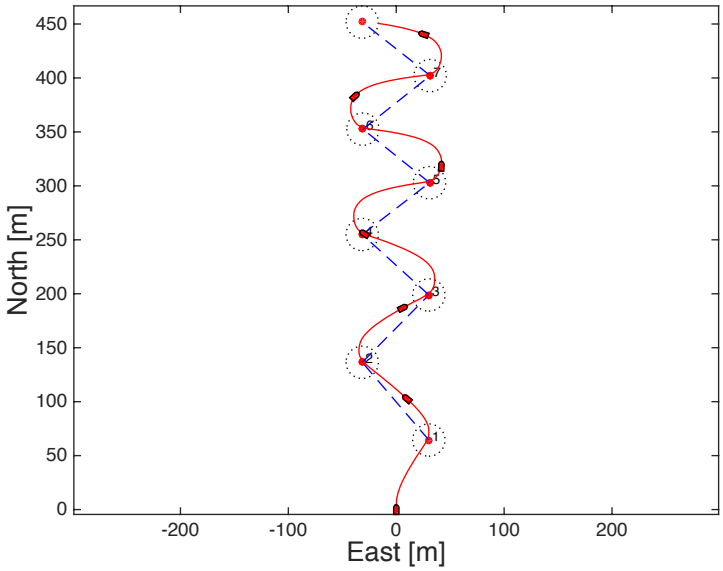



**Figure 6.21:** Illustration of how the RR follows WPs released by the EB for the key parameters set to $v_{max} = 5m/s$ and $r_a = 15$ m.

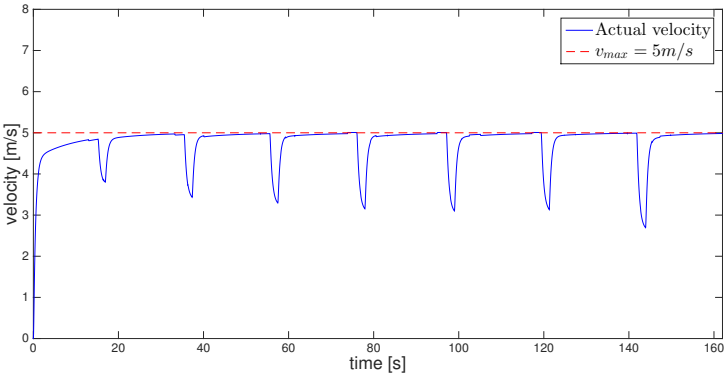



**Figure 6.22:** Illustration of the speed profile for the RR for the path shown in Fig. 6.21.

As predicted, Fig. 6.24 the speed is almost constantly set to the maximum speed allowed, i.e. $v_{max}$. The steady-state error is removed by the controller at a later moment in time since the first three WPs are passed in a satisfactory way.

Clearly the RR have issues making sharp turns, when driving at the higher speeds. This is a result of the controller simply being to slow for this application. The results will be discussed more in section 7.1.

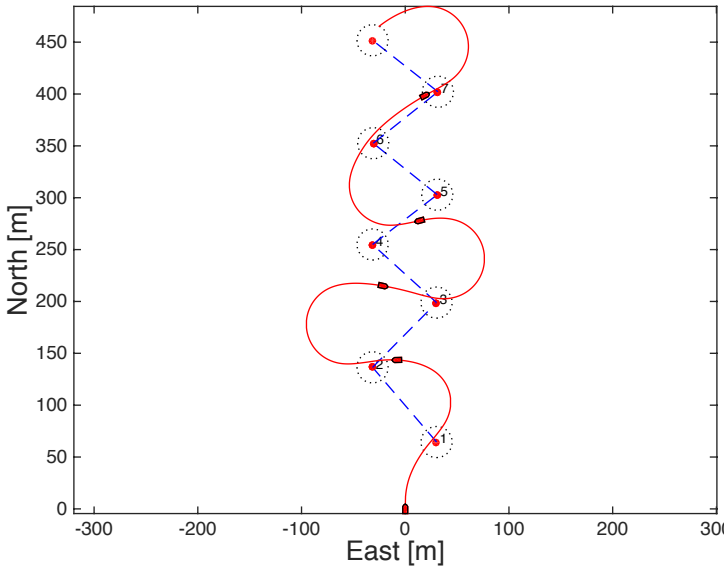



**Figure 6.23:** Illustration of how the RR follows WPs released by the EB for the key parameters set to $v_{max} = 10$ m/s and $r_a = 15$ m.

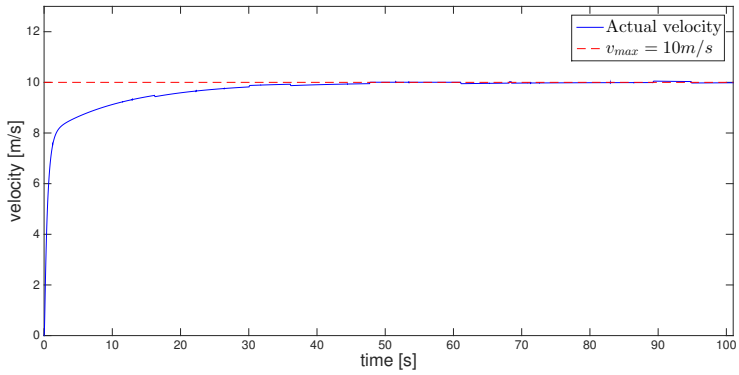



**Figure 6.24:** Illustration of the speed profile for the RR for the path shown in Fig. 6.23.

## 6.2 Empirical results

This part of the chapter presents the results that were collected during actual tests performed on the RR. All tests were performed in shallow and calm conditions in Långedrag, Sweden.

### 6.2.1 Empirical test 1 - Steering accuracy

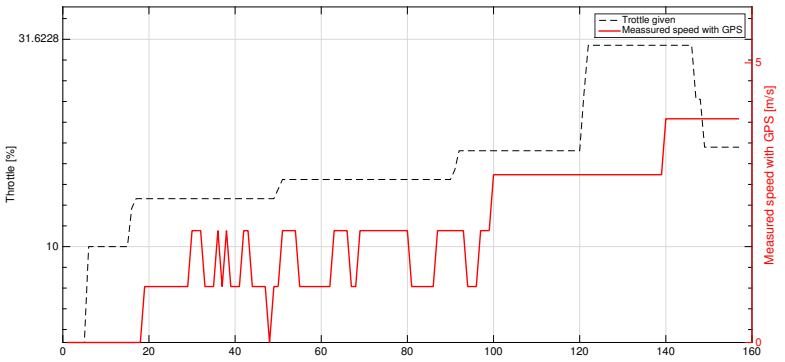
When testing the accuracy of the steering motor the result were mixed. There were some indistinct behavior of the steering, most likely because the highest accuracy the IR-sensor was able to measure was 0.66 degrees out on the handlebar. Larger accurancy was not availble thus the hardware installed on th RR. This made a uncertainty how the steering bar was steering in some cases.

### 6.2.2 Semi-manual results

The *Semi-manual* results were obtained when the RR was remote controlled from the GUI.

#### 6.2.2.1 Emperical test 2 - Throttle

This test shows how the throttle given in percent affects the real speed measured by the GPS on-board the RR, see Fig. 6.25.




.

**Figure 6.25:** Comparison between desired and actual throttle of the RR. This plot contains two plots with there own axes. The dashed black line is the throttle given in percents and the red line is the real speed measured by the GPS on-board the RR.

The GPS speed is expressed with an `int` variable, hence the edgy plot. At sample 44 it can be seen that there is a decrease in the velocity even though the RR have an increasing throttle. This is because the RR took 180° turn at this moment. Therefore the velocity was decreased.

#### 6.2.2.2 Empirical test 3 - Communication range

The test was carried out in order to test the connection ability o the Wi-Fi network. The purpose was to determine how far from the router the RR could go before the connection was lost. This is of great importance since a lost connections means the controller cannot navigate RR. The test was performed by driving in a rectangular path of varying dimension. The path is illustrated in Fig. 6.26.

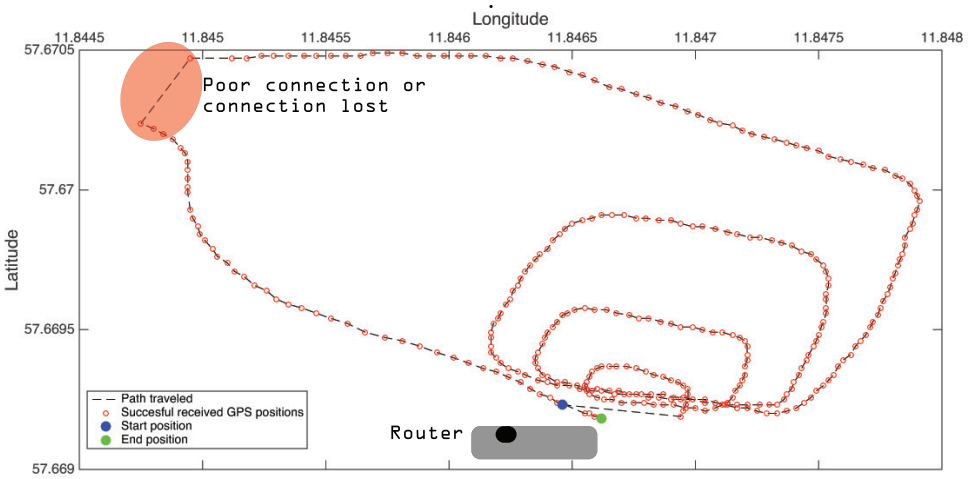



**Figure 6.26:** Illustration of how the connection of the Wi-Fi network is tested at large distances. The red dot represents the areas where the RR lost packages sent from the GUI.

At close distances, i.e. below 120 m, the connection was good and not a single package was lost. When the distance increased, i.e. above 120 m, the connection became unstable and the connection was sometimes lost. The area where the connection was lost is marked red, right corner in Fig. 6.25. This distance is crucial for the RR. At larger distances the information of how the RR should navigate is limited. However, the test lasted for approximately three minutes and of totally 426 packages sent, 411 were received successfully. That corresponds to 3.5 % package loss and comes from the red area in Fig. 6.25. The average response time during the test was 6.41 ms.

### 6.2.3 Path following results

When testing the *Follow me* application a WP containing the GPS position of Asperö was set. Asperö is an island in the vicinity of Långedrag, where the tests were carried

out. Hence, the objective for the *Follow Me* application was to navigate the RR to towards Asperö autonomously. Asperö[1] is located south-west in relation to the test site. Two test were performed, the results were a bit different but the controller showed satisfactory behavior in both tests.

#### 6.2.3.1 Empirical test 4 - *Follow Me* application

The results from the first test of the *Follow me* application is shown in Fig. 6.27 and 6.28.

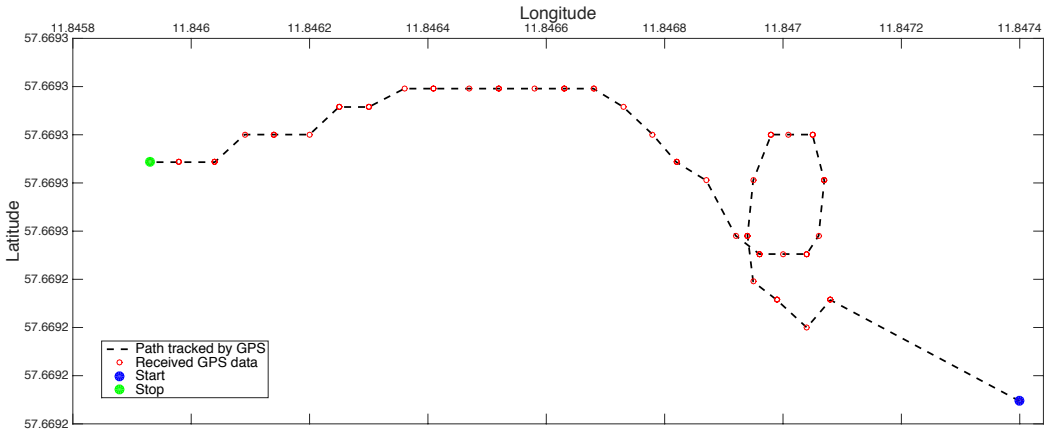



**Figure 6.27:** Illustration of Test 1 where the RR drives autonomously towards Asperö. The RR, controlled by the PID-controller, tries to drive towards Asperö. In the beginning there is some trouble establishing the direction but this is sorted out after some time and the RR starts driving towards Asperö.

It can be seen from the figures that the control system has a bit hard to determine in what direction the RR shall be driving. It drives in a circle but then the controller manages to stabilize the path and set the heading towards Asperö. It can also be noted that the throttle was allowed to only 10% of its maximum when the RR was driving in a circle. The throttle was than increased to 30% of its maximum and the RR responded better to the control signals.

In Fig. 6.29, two plots are shown. The top one shows how the PID-controller sets the steering angle for the RR, i.e. the angle that is fed to the Arduino which sets the angle of the RRs handlebar. The bottom plot shows how the LOS-angle between the

[1]The location of Asperö is $N57°64'8707688734994$ and $E11.79'9488067626953$

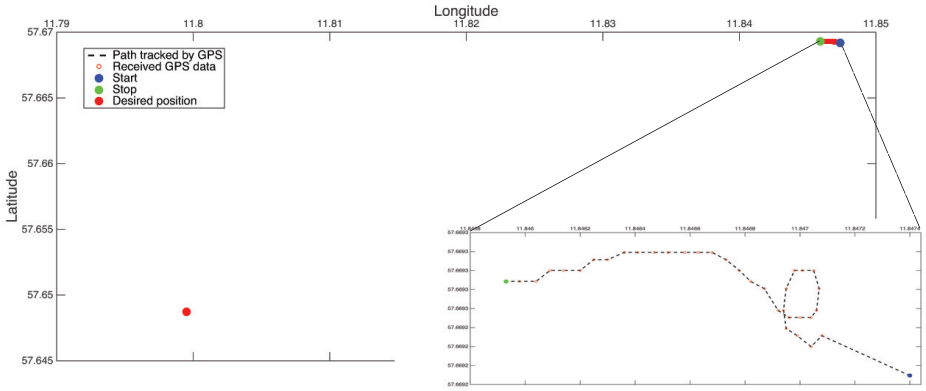



**Figure 6.28:** Illustration of Test 1 zoomed out where the RR drives autonomously towards Asperö.

RR and the GPS point on Asperö is changing. The reason that the angle is changing with a very small rate is due to the fact that Asperö is located far (approximately 3 km) from the current position of the RR, hence the rate of change becomes very small.

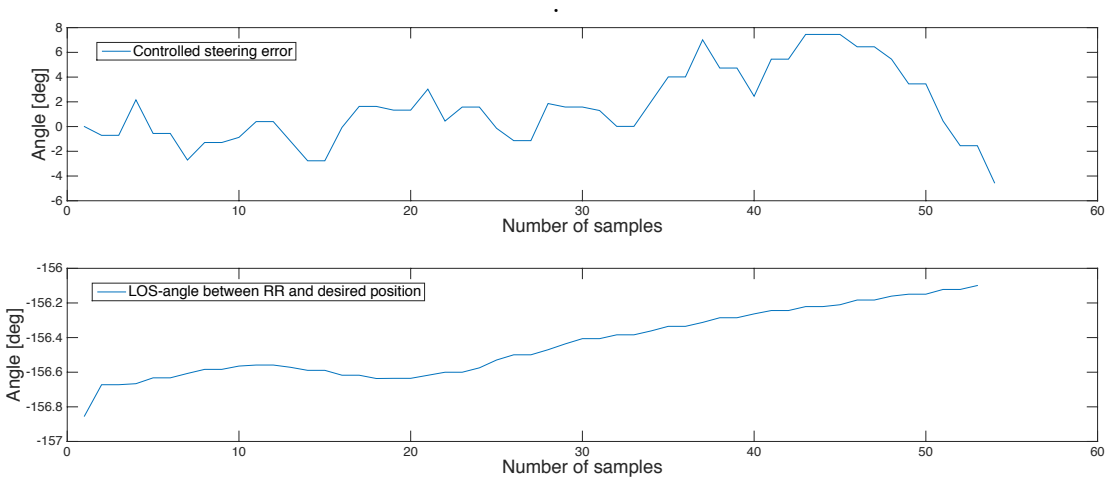



**Figure 6.29:** Illustration of the controlled steering angle and LOS-angle for test 1 where the RR drives autonomous towards Asperö. The PID-controlled steering angle (top) and how the LOS-angle between the RR and the GPS position on Asperö is changing (bottom) for the first test where the RR is driving autonomously in the direction of Asperö.

#### 6.2.3.2 Empirical test 5 - *Follow Me* application

For the second test the results are illustrated in Fig.6.30 and 6.31. Once again it can be observed that the RR has a hard time establishing its path when the throttle was given a low, i.e. around 10%. When the throttle was increased to 30% of its maximum the RR manages to establish a path but it drives a northwest. However, when observing the top plot of Fig.6.32, i.e. the controlled steering angle that is sent from the PID-controller to the RR is $-20$ degrees, which means that the handlebars make a deflection to the left. Hence, if the test were to continue longer the RR would have turned left, towards the GPS-position placed on Asperö.

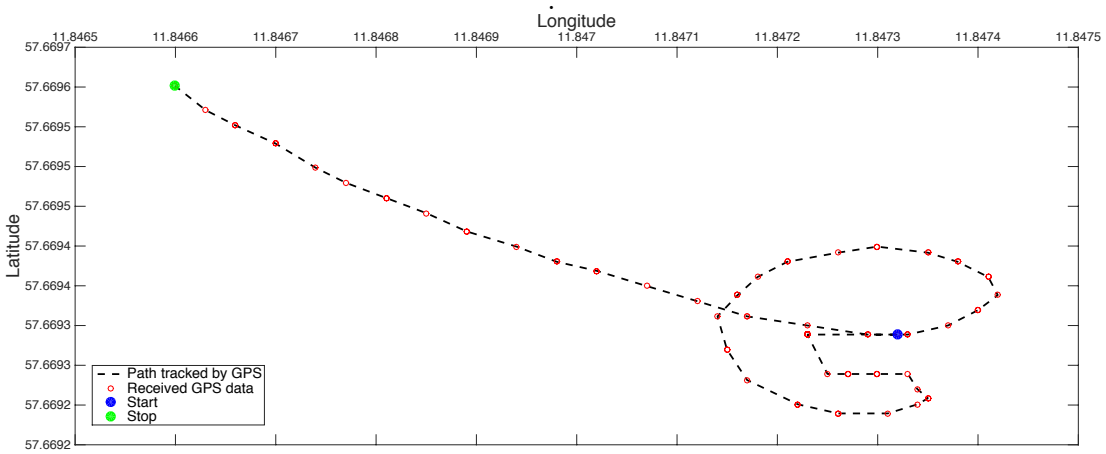



**Figure 6.30:** Illustration of Test 2 where the RR drives autonomously towards Asperö.The RR controlled by the PID-controller, tries to drive towards Asperö. In the beginning there is some trouble establishing the direction but this is sorted out after some time and the RR starts driving north-west.

For this test it can also be noted that the LOS-angle changes with a very small rate, the reason for this is the same as in section 6.2.3.1.

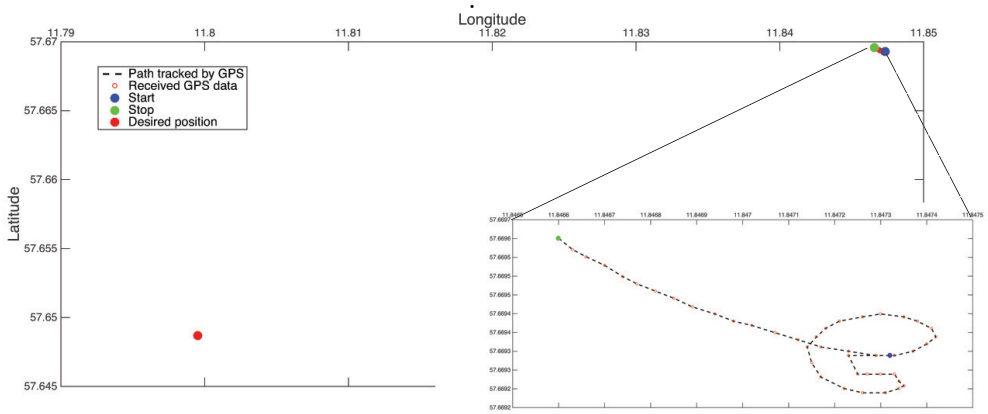



**Figure 6.31:** Illustration of Test 2 zoomed out where the RR drives autonomously towards Asperö.

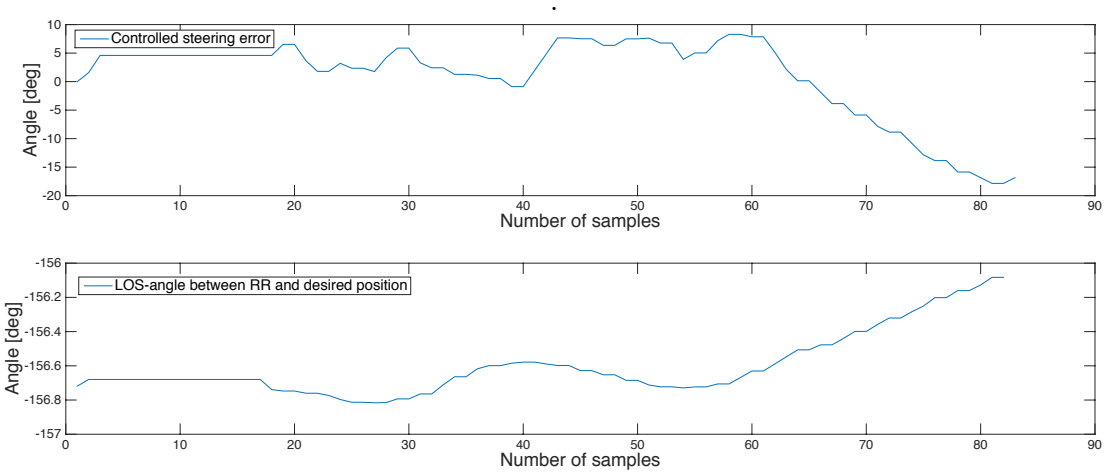



**Figure 6.32:** Illustration of the controlled steering angle and LOS-angle for test 2 where the RR drives autonomous towards Asperö. The PID-controlled steering angle (top) and how the LOS-angle between the RR and the GPS position on Asperö is changing (bottom) for the second test where the RR is driving autonomously in the direction of Asperö.

### 6.2.4 Empirical test 6 - Heading angle

The performance of the IMU, i.e. the tilt compensated compass the yields the yaw angle, was also tested. The test was performed by driving in a rectangular path of varying dimension. The result from the test is illustrated in Fig. 6.33. It can therefore be stated that the yaw angle given from the IMU corresponds well with the yaw angle given from the GPS. Simultaneously as the test was performed the results were also compared with the readings of the analog compass attached on the RR. It shall be mentioned that this compass gave a reading that lied somewhere between the GPS and the IMU. Which one of the two devices to use and reasons for noise in the IMU signal, will be discussed in section 7.2.2.

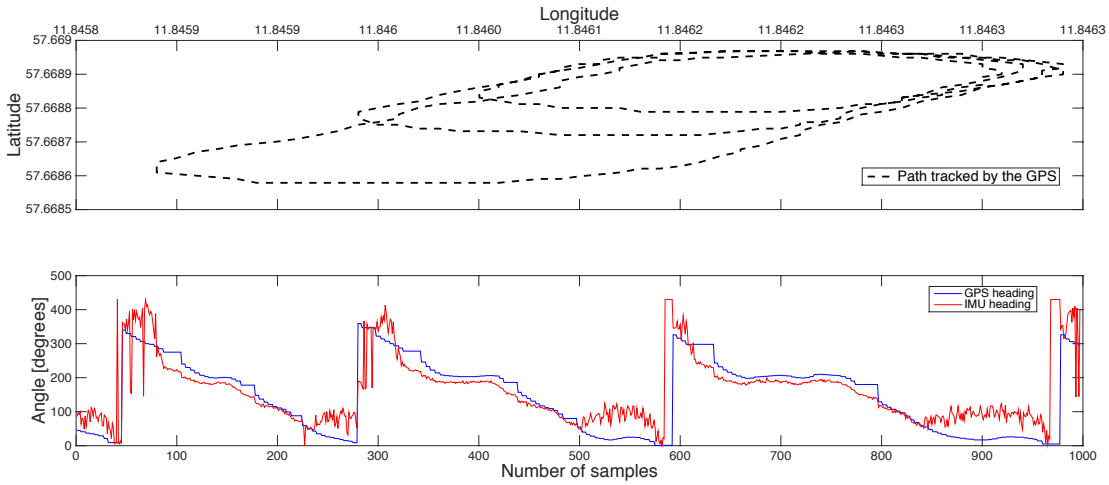



**Figure 6.33:** Illustration of the angular difference between the GPS and the IMU. The angular difference between the GPS and the IMU was tested when driving in a rectangular path of varying dimension as shown in the top figure. The bottom figure shows the difference between the angular deflection given from both the GPS and the IMU.

The performance of the GPS and the IMU was also tested for the RR lying still. See Fig. 6.34

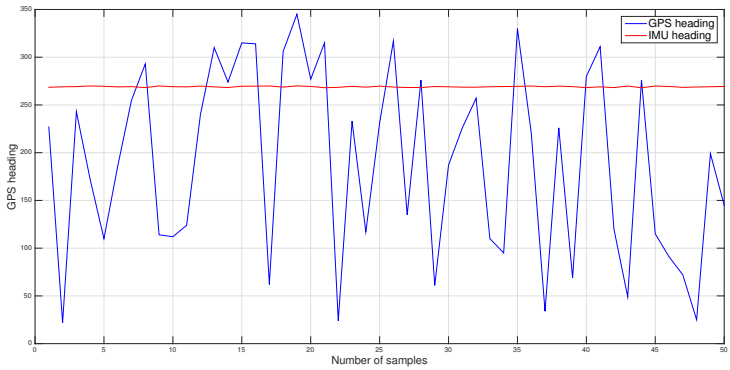



**Figure 6.34:** Illustration of the angular difference between the GPS and the IMU when the RR was lying still.

# 7

# Discussion and concluding remarks

In the following chapter the results obtained in this master's thesis, will be discussed and conclusions will be drawn. In section 1.2 the following objectives were stated:

1. Deriving a mathematical model for the RR.
2. Developing a simulation environment for the RR, in which it is able to autonomously follow a path, consisting of waypoints (WP) fed to its control system.
3. Development of a communication system which enables wireless data transfer between the RR and EB.
4. Assess the results gathered from the simulation and use them to redesign the RR for automatic navigation.

The conclusions pertains to the results of obtained from the testing the actual implementation in a water environment, but also the results received from the simulation model. The simulation model has had a great impact on how the control system in the actual RR was implemented.
The conclusion chapter starts by commenting on the results derived from the simulation model. The authors of this master's thesis feel that the objectives have been fulfilled in a satisfying way. What has been done within the framework of this project can be

improved for even better results. Considering what has been achieved within the time frame, under which this project was carried out, the results are definitely more than acceptable.

## 7.1 Simulation model

A great part of the work that has focused on developing a simulation model of the RR following a predefined path. Starting with deriving the rigid body dynamic equations of the behavior of the RR when in a fluid, ending with deriving a functional model of the RR able to follow a predefined path. The way in which the model was developed, has greatly affected the way the implementation in the RR was carried out.

When considering the results obtained from simulations, presented in chapter 6, it can be seen that the simulated model is good at following the predefined path when the RoA and the maximum allowed speed, had values corresponding good with each other. The shape of the path is also a parameter that affects the simulated behavior, hence a good combination of the three parameters yields a satisfying result. Taking the circular path, e.g. Fig. 6.1 the RR manages to follow the path when the speed was chosen to $5m/s$ and small values were set on the RoA. When the speed is high it becomes harder for the RR to follow the predefined path. It manages to do so only for the RoA is set to a larger value, i.e. $15m$, as shown in Fig. 6.11. The results presented in section 6.1 shows that the system is robust for lower velocities and manages to follow a predefined path even if the RoA is assigned a clingy value. If the simulated system shall have a chance to follow a path, the RoA must be set more generously when the speed is increased.

For the zigzag-shaped path it can be seen that for quick adjustments, regarding the heading direction of the RR, the controller has very hard to adjust the speed and heading for higher speeds. Even if the RoA is assigned with a more generous value, see Fig. 6.23. It shall be mentioned that the zigzag-shaped path is designed in a complex way, which puts hard constraints on the controller. Therefore it should be hard for the RR to follow it. For lower values set on the speed, the model manages to follow the predefined path in a satisfactory way. The reasons for this behavior could be many. One possible explanation is that the RR was simulated in 3 DoF. Adding the other 3 DoF would yield a different result on the behavior. At least for higher speeds. The reason it was kept to

3 DoF in this thesis was due to the fact that it was considered enough for the results to be satisfactory. Another interesting aspect, certainly affecting the behavior, is the pick of controllers. The authors of this thesis chose to work with PID-controllers, for setting both the speed and the heading. A different pick of controllers might have given a different behavior.

A final remark that also affects the behavior to a large extent is deviation, wind and waves, which are all neglected in this model. The impact these three components have will not be a clincher when the weather is calm, which has been assumed throughout this thesis work. For stormy weather though, these three components will have a great impact on the behavior of the RR.

## 7.2 Real implementation

For the implementation in the actual RR the tests were carried out in two different ways. The ability to control the RR was performed both with a remote control, i.e. sending steering and throttle signals directly to it from the self-developed GUI, but also by making it follow a predefined path, fed to a steering and a throttle controller.

### 7.2.1 Steering accuracy

The PID-controller had a hard time trying to steer the RR in a correct way: Probably, since the accuracy of the IR-sensor measuring the position of the handlebars was low. An attempt to increase the accuracy was done. Instead of measuring deflections two times per revolution with the IR sensor, four deflections were measured. This would yield a higher accuracy in theory but the result ended up in a bad loop, where the IR-sensor continuously interrupted the main sequence in the Arduino program. This prevented the main program loop to run properly and resulted in a program failure. To gain better accuracy, parallel executions would have been preferable. This could be achieved by adding additionally Arduino Mega 2560's solely for counting the revolutions from the steering motor.

### 7.2.2 Sensor fusion

The sensor fusion procedure was used in order to get a tilt compensated yaw angle, i.e. to determine in what direction the RR was oriented. In section 6.2.4 a comparison was made between the yaw angle returned from the GPS and the yaw angle returned from the IMU. The angular values from both units were almost similar, as can be seen in Fig. 6.33. It shall be mentioned though, that the GPS returns a reliable yaw angle only when the RR is moving. This is due to the fact that the GPS estimates the yaw angle based on how it passes through two consecutive points on its path. If it is standing still, these points lie at the same place, hence a reliable yaw angle cannot be estimated. In order to get a reliable estimation of the yaw angle at lower velocities, the IMU has to be used. Though the yaw angle returned by the GPS is more stable than the yaw angle returned by the IMU a suggestion from the authors of this thesis is to use the data from the IMU at lower velocities, and the data from the GPS at higher velocities. It shall also be noted that a reason for the noise in the signal might be due to the sensor calibration. The sensors on the IMU stick was calibrated when the RR was standing on land with the motor turned off. The noise in the signal might be a result of the vibrations of the motor in the RR being turned on. Hence, the box containing the IMU starts to sway. When the motor is turned on it generates magnetic fields that also affect the IMU.

### 7.2.3 Communication

The Wi-Fi communication worked good and only 3.5% of the packages were lost at a distance of approximately 120 m from the router. At distances $< 120$ m it was easy to navigate the RR with proper behavior. Sine the steering angle and throttle were sent three times every second it was not crucial if a command was missed. Navigation problems arose with an increased package loss rate. When this rate was to high the RR was impossible to control properly.

As *Semi manual* and *Follow me* parameters are sent without an acknowledgement, the user has no idea whether the RR received a package or not. The application could be modified, forcing the Arduino program in the RR to send an acknowledgement every time the navigation parameters were received. This would increase the robustness in the system. The reason this has not been taken into account in this thesis work is due to problems with communication speed between the RR and the GUI when sending several

data strings using UDP. Excluding the handshake between the two programs had no big influence on the system when the reception was good.

A critical situation was when a package was sent with no restriction of returning a acknowledgement back to the GUI, e.g. when setting the throttle and steering relays from automatic to manual and vice versa. If such a command was not received, the RR was not able to navigate as intended. This created an uncertainty between the GUI and the behavior of the RR.

### 7.2.4 Remote control

For the application, where the RR is driven with remote control over the Wi-Fi the results where very satisfactory. The RR responded well on the steering signals that was fed to it from the GUI. For this application it became obvious that it was easier to maneuver the RR when it was allowed to drive at higher percentages of the maximum throttle.

### 7.2.5 Path following

The RR could successfully be directed to its wished position. The path following technique was implemented in a similar way as done in the simulations. A predefined GPS position was placed in a vector, which was fed to the implemented controllers in the actual RR. In analogy with the simulation a tolerance was put on each WP. The objective of the RR was to switch WP when it was within the RoA of the current one. The two controllers calculated the errors of the steering angle and of the throttle and successfully fed to the Arduino in the RR, making it able to establish its wished position.

As described in section 6.2 the controller had trouble guiding the RR in the right direction when the throttle was limited to lower values. When the throttle was allowed the reach approximately 30% of the maximum throttle the control system managed to steer the RR in the correct direction. A thing, which was not considered in this control system, that has a great impact on the control behavior is the rotation velocity of the RR. The controller manages to guide the RR in the correct direction. Although it cannot compensate for the overshoot on the tail of the RR, which is a result of its inertia together with drift and other environmental disturbances as a result of traveling in water.

However, the path following implementation can be perfected, since the RR sometimes have a hard time following several WPs set out as a path. Sometimes the WP switching does not work properly and this is something that has to be investigated in future work. This behavior might be due to the pick of controller for the application and maybe a different implementation would have been more suitable. However, this was outside the scope of this project.

## 7.3 Future work

In this section the authors of this thesis will propose a couple of things that are worth investigation thoroughly in continuous work. A list of areas to investigate is attached below:

- Implement environmental disturbances on the simulation model such as deviation, turbulence from waves and wind.
- Implement a 6 DoF model of the RR for a more realistic result at higher velocities.
- Investigate if different controllers are required for velocity and heading. Faster, more adaptive controllers, are suggested when the RR reaches higher velocities and for its path following to become more robust.
- Further development of the path following implementation. Make the application more robust.
- Take more advantage of the powerful IMU sensor installed on the RR, e.g. use the gyroscope and accelerometer to increase the steering performance preventing the RR to slide sideways.

# Bibliography


[1] Sjöräddningssällskapet. (2015, January) We save lives at sea with no government funding. [Online]. Available: http://www.sjoraddning.se/ine-english/

[2] F. Falkman. (2015, January) Follow-me, little rescue boat! - how might we make the rescuerunner follow a bigger rescue boat automatically on a safe distance? [Online]. Available: http://fredrikfalkman.tumblr.com/

[3] T. I. Fossen, *Guidance and Control of Ocean Vehicles*. Buffins Lane, Chichester, West Sussex, England: John Wiley & Sons Ltd., 1994.

[4] D. Krammer, "Modeling and control of autonomous sailing boats," Master's thesis, Eidgenössiche Technische Hochschule, Zürich, Switzerland, 2014.

[5] T. I. Fossen, *Handbook of Marine Craft Hydrodynamics and Motion Control*. Buffins Lane, Chichester, West Sussex, England: John Wiley & Sons Ltd., 2011.

[6] T. I. Fossen and T. Perez, "MSS - Marine Systems Simulator," March 2015. [Online]. Available: http://www.marinecontrol.org

[7] T. I. Fossen, M. Breivik, and R. Skjetne, "Line-of-sight path following of underactuated marine craft," Center of Ship and Ocean Structures (CESOS), Norwegian University of Science and Technology (NTNU), Trondheim, Norway, Tech. Rep., 2003.

[8] M. Breivik, "Nonlinear maneuvering control of underactuated ships," Master's thesis, Norwegian University of Science and Technology (NTNU), Trondheim, Norway, 2003.

[9] T. Marius Jensen, "Waypoint-following guidance based on feasibility algorithms," Master's thesis, Norwegian University of Science and Technology (NTNU), Trondheim, Norway, 2011.

[10] B. Lennartson, *Relgerteknikens grunder*, 4th ed. Lund: Studentlitteratur AB, 2002.

[11] K. Nomoto, T. Taguchi, K. Honda, and S. Hirano, "On the steering qualities of ships," International Shipbuilding Progress, vol 4, Tech. Rep., 1957.

[12] L. Xiao and J. Jouffroy, "Modeling and nonlinear heading control for sailing yachts," in *OCEANS 2011,Waikoloa, HI*, Sep.19-22, 2011, pp. 1–6.

[13] F. Furrer, "Developing a simulation model of a catamaran using the concept of hydrofoils," Master's thesis, Eidgenössiche Technische Hochschule, Zürich, Switzerland, 2010.

[14] T. I. Fossen and T. Perez. (2015, March) Kinematic models for seakeeping and maneuvering of marine vessels. modeling, identification and control. [Online]. Available: http://www.marinecontrol.org

[15] G. Geiger, J. Bartholomeyczik, U. Breng, W. Gutmann, M. Hafen, E. Handrich, M. Huber, A. Jäckle, U. Kempfer, H. Kopmann, J. Kunz, P. Leinfelder, R. Ohmberger, U. Probst, M. Ruf, G. Spahlinger, A. Rasch, J. Straub-Kalthoff, M. Stroda, K. Stumpf, C. Weber, M. Zimmerman, and S. Zimmerman, "MEMS IMU for AHRS Applications," in *Position, Location and Navigation Symposium, '08 IEEE/ION, Monterey, CA*, May 5–8, 2008, pp. 225–231.

[16] Sparkfun. (2015, April) 9 degrees of freedom - sensor stick. PICTURE. [Online]. Available: https://www.sparkfun.com/products/10724

[17] Arduino Software. (2015, March) Arduino Mega, Specifications. [Online]. Available: http://arduino.cc/en/Main/arduinoBoardMega

[18] F. Leens, "An Introduction to SPI and $I^2C$ Protocols," *IEEE Instrumentation and Measurement Magazine*, vol. 12, no. 1, pp. 8–13, February 2009.

[19] *ADXL345 Digital Accelerometer Specification*, Analog Devices Inc., 2009-2013.

[20] *ITG3200 Gyroscope Product Specification, Revision 1.4*, InvenSense Inc., 2010.

[21] *3-Axis Digital Compass IC HMC5843*, Honeywell Inc., 2009.

[22] F. Abyarjoo, A. Barreto, J. Cofino, and F. Ortega, "Implementing a Sensor Fusion Algorithm for 3D Orientation Detection with Inertial/Magnetic Sensors," in *Innovations and Advances in Computing, Informatics, Systems Sciences, Networking and Engineering*. Cham: Springer International Publishing, 2015, pp. 305–310.

[23] *Electronic Compass Design Guide - Using The HMC5843 Digital Compass IC*, Honeywell Inc., 2009.

[24] H. Pang, M. Pan, C. Wan, J. Chen, X. Zhu, and F. Luo, "Integrated compensation of magnetometer array magnetic distortion field and improvement of magnetic object localization," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 52, no. 9, pp. 5670–5676, 2014.

[25] Camel Software. (2015, May) 3-axis magnetometer calibration – a simple technique for hard & soft errors. [Online]. Available: http://www.camelsoftware.com/firetail/blog/uavs/3-axis-magnetometer-calibration-a-simple-technique-for-hard-soft-errors/

[26] W. T. Higgins, "A Comparison of Complementary and Kalman Filtering," *IEEE Transactions of Aerospace and Electronic Systems*, vol. 11, no. 3, pp. 321–325, 1975.

[27] A. Andersson, V. Bäckman, O. Pantzare, A. Sarvik, and O. Granqvist, "Eskortering av autonom vattenskoter," 2015, unpublished.

# A

# Simulation

Pseudo code for generating a continuous mapping of the LOS-angle, i.e. from $(-\pi,\pi)$ to $(-\infty,\infty)$.

**Algorithm** pseudo code describing the mapping of the *atan*2-function

```
1:  procedure ATAN2-MAPPING(previous_state,current_state,accumulation)
2:      if previous_state = 1 then
3:          if current_state = 1 then
4:              accumulate = psi_current − psi_previous.
5:          else if current_state = 2 then
6:              accumulate = psi_current − psi_previous.
7:          else if current_state = 3 then
8:              if abs(psi_current + psi_previous <= π then
9:                  accumulate = psi_current − psi_previous.
10:             else
11:                 accumulate = psi_current − psi_previous + 2π.
12:             end if
13:         else
14:             accumulate = psi_current − psi_previous.
15:         end if
16:         current_state = 1
17:         current_state = 2
```

**Algorithm** pseudo code describing the mapping of the *atan*2-function (continued)

```
18:     else if previous_state = 2 then
19:         if current_state = 1 then
20:             accumulate = psi_current − psi_previous.
21:         else if current_state = 2 then
22:             accumulate = psi_current − psi_previous.
23:         else if current_state = 3 then
24:             accumulate = psi_current − psi_previous.
25:         else
26:             if abs(psi_current + psi_previous <= π then
27:                 accumulate = psi_current − psi_previous.
28:             else
29:                 accumulate = psi_current − psi_previous + 2π.
30:             end if
31:         end if
32:     else if previous_state = 3 then
33:         if current_state = 1 then
34:             if abs(psi_current + psi_previous <= π then
35:                 accumulate = psi_current − psi_previous.
36:             else
37:                 accumulate = psi_current − psi_previous + 2π.
38:             end if
39:         else if current_state = 2 then
40:             accumulate = psi_current − psi_previous.
41:         else if current_state = 3 then
42:             accumulate = psi_current − psi_previous.
43:         else
44:             accumulate = psi_current − psi_previous − 2π.
45:         end if
46:         current_state = 3
```

**Algorithm** pseudo code describing the mapping of the *atan*2-function (continued)

```
47:     else
48:         if current_state = 1 then
49:             accumulate = psi_current − psi_previous.
50:         else if current_state = 2 then
51:             if abs(psi_current + psi_previous <= π then
52:                 accumulate = psi_current − psi_previous.
53:             else
54:                 accumulate = psi_current − psi_previous + 2π.
55:             end if
56:         else if current_state = 3 then
57:             accumulate = psi_current − psi_previous + 2π.
58:         else
59:             accumulate = psi_current − psi_previous.
60:         end if
61:         accumulation = accumulation + accumulate.
62:         psi_previous = psi_current.
63:     end if
64:     current_state = 4
65: end procedure
```

# B

# Arduino

## B.1 Rescuerunner

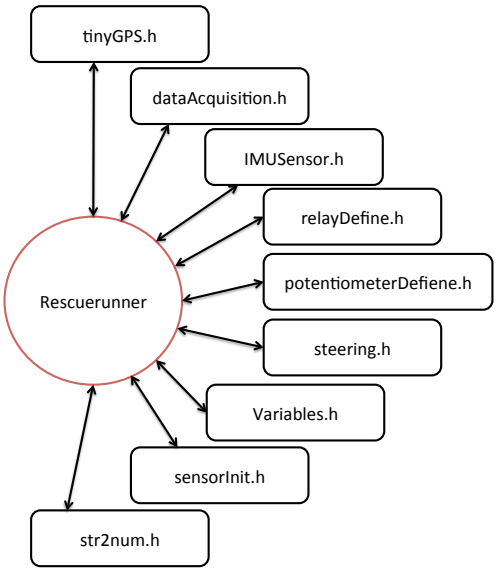



**Figure B.1:** Overview of the Arduino programs in the RR.

- `Rescuerunner` - This is the main function of the RR's Arduino program. This program connects to the wireless network *Follow Me* with static IP-adress 192.168.0.101 with port number 2390. The program then listens if any UDP package where sent to the port. When a package is received there are several if statement depending which message that was received.

  1. `if (message.startsWith("RRconnect"))` - If string was `RRconnect` return an acknowledge to the GUI through UDP that the RR was connected.
  2. `if (message.startsWith("RRstart"))` - If string was `RRstart` invoke function `startRelay()` to start engine.
  3. `if (message.startsWith("RRstop"))` - If string was `RRstop` invoke function `stopRelay()` to stop engine.
  4. `if (message.startsWith("RR_getData"))` - If string was `RR_getData` invoke function `sendYawGPSdata()`, return the GPS and yaw data to the GUI.
  5. `if (message.startsWith("RRinit))` - If string was `RRinit` invoke function `initCounter()` to initilize steering.
  6. `if(strncmp(packetBuffer, "setNav",6) == 0)` - If string starts with `setNav` the following text in that string will contain information about steering and throttle. The string will look like

     ```
     setNav025/050
     ```

     where `setNav` is the identifying string, `010` is the steering angle and `050` is the throttle in percents (50%)
  7. `if (message.startsWith("RRinit))` - If string was `RRinit` invoke function `initCounter()` to initialize steering.
  8. `if (message.startsWith("RR_Man2Aut"))` - If string was `RR_Man2Aut` invoke function `steering_Man2Aut()`
  9. `if (message.startsWith("RR_Aut2Man"))` - If string was `RR_Aut2Man` invoke function `steering_Aut2Man()`
  10. `if (message.startsWith("RRrelayMan"))` - If string was `RRrelayMan` set minimum throttle and invoke function `set2ManRelay();`

11. `if (message.startsWith("RRrelayAut"))` - If string was `RRrelayAut` set minimum throttle and invoke function `set2AutRelay()`

- `tinyGPS.h` - This library contains code for reading the GPS unit. Returning GPS position in longitude and latitude with 10 digits each, e.g. longitude can have the following value `57.68856048`. It is also calculating the true speed in meter/second and heading angle in degrees.

- `dataAcquisition.h` - This library just contains one function

  1. `void senYawGPSdata(void)` - this function builds up the string base in the GPS data and sends the GPS data from `tinyGPS.h` to the GUI through UDP protocol.

- `IMUsensor.h` - This library reads the IMU sensor and calibrate and filtering the values returning a tilt compensating compass with values 0-359 degrees, see Chapter 4.

- `relayDefine.h` - This library sets the Songle relay board's depending if the system want to be in manual mode or automatic mode. This library contains five functions

  1. `void initRealy(void)` - This function initialize the pins on the Arduino and puts all relays into manual mode at start up.
  2. `void startRelay(int realyPos)` - This function enables the start engine to run.
  3. `void stopRelay(int realyPos)` - This function disable the main engine and turn it off.
  4. `void set2AutRelay(void)` - This function disables handlebar throttle and enables digital throttle from the program.
  5. `void set2ManRelay(void)` - This function enables handlebar throttle and disable digital throttle from the program.

- `potentiometerDefine.h` - This library four throttle functions.

  1. `void initPot(void)` - Initialize pin and set the throttle to minimum.
  2. `void digitalPotWrite(int command, int value)` - This function sets the throttle by changing the potentiometer.

3. `double percToOhm(int percent)` - This function returns throttle from percent to ohm by [27]

   ```
   0.036*percent + 0.9
   ```

   where 0.9 is the idle resistance for the motor to be able to start.

4. `int ohmToByte(void)` - This function returns the throttle from ohm to byte by [27]

   ```
   (ohm-0.0957) / 0.0308
   ```

- `sensorDefine.h` - This library handles the steering sensors. There are two sensors, one calculating how many revolutions the steering motor has turned with a IR sensor and one measures when the handlebar has swung to maximum right.

  1. `void initCounter(void)` - This function will initialize the handlebar turning it maximum to left. After that it will start counting revolutions from the steering motor by calling `void degreeCount(void)` function. Now the handlebar tuning right until the amount of revolutions has been equal to the pre-calculated revolutions making it know where straight ahead is.
  2. `void degreeCount(void)` - This function will count the revolutions continuously.

- `steering.h` - This library handles the steering functions

  1. `void steerLeft(int U)` - This function steer the handlebar to left with an input voltage `U` that is set to 12V.
  2. `void steerRight(int U)` - This function steer the handlebar to right with an input voltage `U` that is set to 12V.
  3. `void stopMotor()` - Sets the steering to manual mode by slacking pulley
  4. `void steering_Man2Aut(void)` - Sets the steering to automatic control by extend the pulley.

5. `void steering_Aut2Man(void)` - Sets the steering to automatic control.
6. `steerInit(void)` - Define pin modes for slacking/extend motor
7. `steeringInit(void)` - Define pin modes for steering motor.

- `str2Num.h` - This library handles the string that are recieved from the program and convert it into digits.
- `variable.h` - This library contains all the parameters that is used.

## B.2 Ego boat

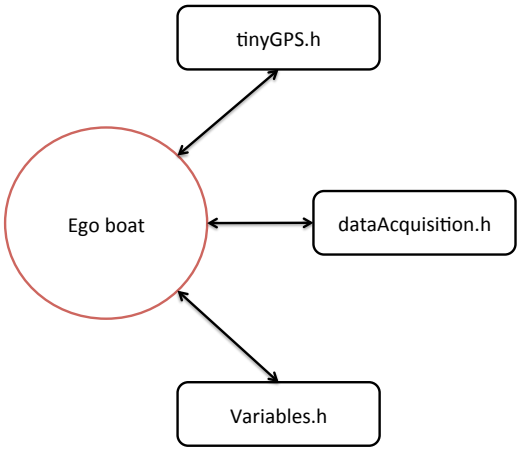



**Figure B.2:** Overview of the Arduino programs in the EB.

- `egoBoat` - This is the main function of the EB's Arduino program. This program connects to the network *Follow Me* through Ethernet, with static IP-address 192.168.0.100 with port number 2391. The program then listens if any UDP package where sent to the port. When a package is received there are several if statement depending which message that was received.

1. `if (message.startsWith("EBconnect"))` - If string was `EBconnect` return an acknowledge to the GUI through UDP that the EB was connected.
2. `if (message.startsWith("EB_getData"))` - If string was `EB_getData` invoke function `sendGPSdata()`, return the GPS data to the GUI.

- `tinyGPS` - This library contains code for reading the GPS unit. Returning GPS position in longitude and latitude with 10 digits each, i.e longitude can have the following value 57.68856048. It is also calculating the true speed in meter/second and heading angle in degrees.
- `dataAcquisition.h` - This library just contains one function
    1. `void sendGPSdata(void)` - this function builds up the string base in the GPS data and sends the GPS data from `tinyGPS.h` to the GUI through UDP protocol.
- `variable.h` - This library contains all the parameters that is used.

# C

# GUI - Follow me

The GUI invokes multiply functions during its operation, see Fig: C.1.

- `RR_try_connect()` - Tries to connect to RR. Returns `true` if it does, else `false` which means program needs to be connected again.
- `EB_try_connect()` - Tries to connect to EB. Returns `true` if it does, else `false` which means program needs to be connected again.
- `RR_steering_Aut2Man()` - Set steering into manual mode.
- `RR_steering_Man2Aut()` - Set steering into automatic mode.
- `RR_start()` - Start the start engine for four seconds.
- `RR_stop()` - Stop the engine.
- `RR_init_steering()` - Run the initialize steering routine.
- `RR_steering_correction()` - Makes sure the steering is between arbitrary values.
- `RR_enable_throttle(boolean)` - Switch between handlebar/digital throttle.
- `RR_getData()` - Request information from the RR's GPS and IMU sensor. Returns a string with GPS and IMU data, `RR_info`
- `EB_getData()` - Request information from the EB's GPS Returns a string with GPS data, `EB_info`

- `distance_between(RR_info EB_info)` - Calculates the distance between RR and EB returning `distance`. Input parameters is the GPS and IMU data from both RR and EB.
- `tolerance_acceptance(distance)` - Check if the RR is arbitrary close to the EB. Input `distance`.
- `RR_velocity()` - Calculates the RR's speed.
- `angel_between()` - Calculates the angle between the RR and EB.
- `steering_PID(steering_error,vel_RR)` - Calculates the RR's new heading angle. Returns `steering_signal`
- `throttle_PID(throttle_error,vel_RR)` - Calculates the RR's new throttle. Returns `throttle_signal`
- `setNavigation()` - Sends throttle and steering angle to RR.
- `Disconnect()` - Turns of the communication between the two Arduinos and the GUI.

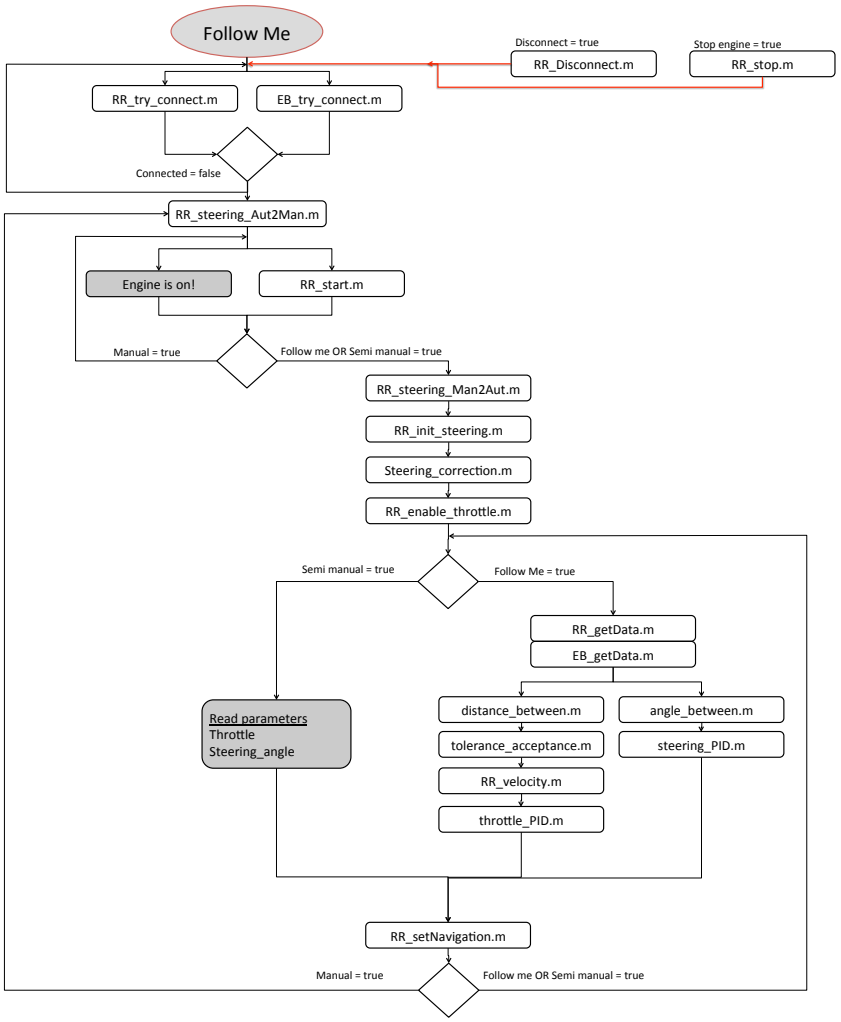



**Figure C.1:** Schematic overview of how the GUI invokes its functions

# D

# Digital appendix

The Digital appendix will include code, simulation models and similar packages that have been used through this thesis. The Digital appendix is divided into the four following parts:

1. `Simulation.zip` - Functions and files to run the simulations in Simulink.
2. `Arduino.zip` - Arduino programs for RR and EB.
3. `Follow Me.zip` - GUI that enables the user to control the RR.
4. `Movie.zip` - Movie clips testing the RR.