Adapting a small watercraft to autonomously follow a lead boat

In cooperation with the Swedish Sea Rescue Society (SSRS)

Master's thesis in Systems, Control and Mechatronics

Björn Hallhagen
Eric Persson

Department of Signals and Systems
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2016

Master's thesis EX091/2016

Adapting a small watercraft to autonomously follow a lead boat

Björn Hallhagen
Eric Persson

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Signals and Systems
Division of Automatic control, Automation and Mechatronics
Chalmers University of Technology
Gothenburg, Sweden 2016

Adapting a small watercraft to autonomously follow a lead boat
Björn Hallhagen
Eric Persson

Supervisor: Fredrik Falkman, Swedish Sea Rescue Society (SSRS)
Examiner: Petter Falkman, Department of Signals and Systems

Master's Thesis EX091/2016
Department of Signals and Systems
Division of Automatic control, Automation and Mechatronics
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Image of the RescueRunner autonomously following a lead boat.

Adapting a small watercraft to autonomously follow a lead boat
Björn Hallhagen, Eric Persson
Department of Signals and Systems
Chalmers University of Technology

Abstract

The Swedish Sea Rescue Society (SSRS) has developed a water scooter for rescue operations called the RescueRunner. Due to complications with transporting the RescueRunner to and from accident sites, a solution has been proposed where it follows a lead boat autonomously.

In this paper, a complete hardware and software architecture is proposed for adapting the RescueRunner to autonomously follow a lead boat, and its performance is evaluated. The system includes a sensor fusion algorithm based on the extended Kalman filter, a GNSS based tracking system, a radio communication link and a simple control scheme based on PID controllers.

The sensor fusion algorithm estimates the RescueRunner's and the lead boat's orientations with the use of gyroscopes, accelerometers and magnetometers. Tests show that the system is responsive and precise, but that the magnetometers are sensitive to miscalibrations. The tracking system uses GNSS receivers to estimate the two boats' positions and velocities. Tests show that the system succeeds in estimating the relative distance between the two boats with an accuracy of 4.5 m. The radio link succeeds in achieving stable communications between the two boats for ranges up to 500 m, which was deemed sufficient. The system was evaluated using a simple control scheme that minimizes the angle to the lead boat from the RescueRunner's frontal perspective while attempting to keep a set distance from lead boat. This control strategy makes the RescueRunner trail behind the lead boat as expected and validates the functionality of the rest of the system.

Overall, the system shows some promising results and provides a platform for future work with all the necessary subsystems functioning. The control scheme is a prime candidate for future work, as a more advanced structure is needed to improve the performance.

Keywords: Swedish Sea Rescue Society, water scooter, RescueRunner, sensor fusion, extended Kalman filter, GPS, IMU, autonomous control.

Acknowledgements

We would like to thank our supervisor Fredrik Falkman for his insightful input regarding the RescueRunner as well as all the support and encouragement he has provided. A word of gratitude also goes out to all the other people at SSRS without whom the tests at sea would not have been possible. During the work we have had a very rewarding collaboration with Gustav Lindberg, John Skötte and Hampus Bergh who worked on their Bachelor thesis during the spring, which resulted in a very well functioning system for electronic control of the RescueRunner. Without you, this work would have been stranded. We would also like to thank Lukas Wikander for proof-reading and offering insightful comments and constructive criticism. Lastly, we would like to thank our examiner Petter Falkman for guiding us through the work and offering his much appreciated advice.

Björn Hallhagen and Eric Persson, Gothenburg, September 2016

Contents

List of Figures

List of Tables

Nomenclature

I^{2}C Inter-Integrated Circuit (I^{2}C) is a serial communication interface used in embedded systems.

Aft Towards the stern (rear) of a boat.

Beam The beam of a boat is the maximum width of its hull.

Bow The front part of a boat.

C++ A general-purpose programming language that was initially designed as an extension of the C programming language to achieve high-level programming features while retaining most low-level programming features of C.

Draught The draught of a boat is the vertical distance between the waterline and the bottom of the boat's hull.

EKF The Extended Kalman Filter (EKF) is a filtering method used for combining information from several sensors into estimates of a chosen set of system states. It is capable of handling nonlinear systems by continuously linearizing the system around the current states and then applying the principles of the Kalman filter.

FIFO First In, First Out (FIFO) is a method for organizing and manipulating a data buffer, where the oldest (first) entry is processed first.

GNSS A Global Navigation Satellite System (GNSS) is a system that uses satellites to estimate a GNSS device's position on Earth.

GPS Global Positioning System (GPS) is a global navigation satellite system that is used to provide position data.

IMU Inertial Measurement Unit (IMU) is an electronic device that measures a body's specific force and angular rate. They may also include measurements of the magnetic field around the body.

ISM The Industrial, Scientific and Medical (ISM) bands are a set of radio frequency bands available for unlicensed use.

LOA The Length Over All (LOA) is the maximum length of a boat measured parallel to the waterline.

NWU A North-West-Up (NWU) coordinate system with the x-axis pointing north, the y-axis pointing west and the z-axis pointing up.

PID controller A Proportional, Integral and Derivative (PID) controller that uses a weighted sum of the control error, its integral and its derivative to form the control output. Since the controller does not need a system model it is relatively simple to apply to a variety of applications.

Port Facing towards the front of a boat, this is the left side.

SPI Serial Peripheral Interface (SPI) is a synchronous serial communication interface used in embedded systems.

SSRS The Swedish Sea Rescue Society, Svenska Sjöräddningssällskapet (SSRS) in swedish, is a voluntary organization dedicated to saving lives at sea.

Starboard Facing towards the front of a boat, this is the right side.

Stern The rear part of a boat.

Sway The term used for side to side (port-starboard) motion of a ship.

Transceiver A device that can act as both transmitter and receiver.

UART Universal Asynchronous Receiver/Transmitter (UART) is an asynchronous serial communication interface used in embedded systems.

1

Introduction

1.1 Background

Founded in 1907, the Swedish Sea Rescue Society (SSRS) [29] is a voluntary organization dedicated to saving lives at sea. Along the coast and the larger lakes of Sweden, SSRS currently has 69 stations posted, ready to respond to emergency calls within 15 minutes at all hours of the day.

When traveling through rough sea, larger rescue boats are preferred for their stability and seaworthiness. However, once at the scene of an accident, their size tends to become a hindrance. They are typically too unwieldy to reach close enough to the people in need, both due to their limited maneuverability and high freeboard. Their size also limits their operation to deeper waters.

To remedy this, the SSRS has developed the RescueRunner [4], a small and flexible water scooter. With a soft plastic hull, this nimble craft can operate in shallow waters among cliffs and other obstacles, without the risk of being damaged. This makes the RescueRunner a perfect complement to the larger rescue boats.

Despite its advantages, the RescueRunner is currently mostly used for education, training and occasional operations near the rescue stations. Its current limitation lies in transporting the RescueRunner to and from the rescue site. There are several suggested solutions to this, each with their own perks and drawbacks. Driving the RescueRunner manually would require a designated driver and would be physically and mentally tiring. Carrying the RescueRunner on the larger boats would require modifications and take up much of the already limited space. Towing the RescueRunner behind the leading boat would limit the boat's ability to tow other boats and can damage the RescueRunner's jet propulsion system as water is being forced through. Another alternative that is currently being explored is to have the RescueRunner autonomously following the leading boat. This last alternative is the premise of the Follow Me project started by SSRS, and will be investigated in this paper. [8]

1.2 Purpose

The vision of the project is to develop the RescueRunner water scooter into an autonomous vehicle able to follow a lead boat, with little to no human interference necessary. For this, a state observer is needed that allows the RescueRunner to position itself relative to the lead boat while in motion.

Preliminary requirements have been found by studying similar control problems, and through observations made while driving the RescueRunner. These requirements represent a minimum in performance. The minimum requirement for the state observer's update rate is set to 20 Hz based on the control rate used in [7], where a control system for active steering of an autonomous ground vehicle is presented.

When the RescueRunner follows a lead boat, that boat provides an area of calmer water in its wake. Ideally, the RescueRunner should remain within this area to avoid being exposed to rough sea. Initial test runs have shown that a distance of about 20 m behind the lead boat is a suitable position for the RescueRunner. At that distance, the wake of the boat is approximately 8 m across. The minimum requirement for the state observer's horizontal positioning precision is thus set to ± 4 m, so that the RescueRunner is kept within the wake. This horizontal positioning error budget is shared with the control system used for positioning, meaning that the entire error budget should not be spent on the state observer.

The solution should be designed so that it is unaffected by which boat is currently used as leader. The task of switching to another lead boat should be simple. Ideally this should only be a matter of moving one self-contained device from one boat to the other.

1.3 Delimitations

Certain delimitations are formulated in order to complete the project within the limited timeframe. One such delimitation is that no control system for following the lead boat will be attempted until an operational state observer has been designed, tested and deemed satisfactory. The reason for this is that the performance of the control system will be dependent on the quality of the measurements upon which it bases its calculations.

A complete autonomous following system will require an intuitive and easy to use interface for the operator. This must function in a variety of situations such as different weather conditions and different times of the day. However, the development for such an interface will not be part of this thesis, since the Follow Me project is still in its early stages of development. In addition, no attempts will be made to make the system resistant to security intrusions.

State observer designs that are highly computationally demanding, such as particle filters, will not be investigated. This is due to limitations in the processing power of microcontrollers.

The control system will be aimed at following the lead boat at high speeds as the RescueRunner is easier to steer in this range. Thus, its performance at low speeds (around 20% throttle or less) will not be investigated.

The work will not include a system for avoiding obstacles or navigating in or out of port, only following the lead boat on open water.

1.4 Question formulation

- What physical quantities are interesting from a control perspective?
- What is a suitable level of detail for the estimation filter?
  - Are there any filters used for similar purposes in other projects to draw inspiration from?
- What level of precision is achievable from sensor fusion of the chosen set of sensors?
  - How much is this affected by the surrounding environment such as winds, waves and vibrations from an engine?
- What distance between the RescueRunner and the lead boat is preferable with respect to both maneuvering capability and safety? How fast does the state observer have to operate to allow a control system to position the RescueRunner at that distance?
- There needs to be a reliable method of communication between the lead boat and the RescueRunner. How can this be achieved?
  - What are the repercussions of communication loss? Can these be mitigated?

1.5 Related work

In [7] a control system for active steering of an autonomous ground vehicle is presented. The system could maintain stability throughout a double lane change maneuver for a 2000 kg car traveling on a slippery surface at speeds up to 21 m/s. This was achieved using a control system running at a rate of 20 Hz. While the RescueRunner's dynamics are much less predictable than that of a car, the task of following a lead boat requires much less precision than a double lane change. Thus, this work was used to provide a rough approximation of the control rate necessary to steer it. This will in turn set requirements on how fast the other components of

the system should operate.

In [27] the development of a small autonomous sail powered boat is presented. The system is intended to be used in longer autonomous missions e.g. to collect environmental data. To this end it is designed to be fully autonomous in both path planning and control, as well as being self sufficient in energy collection. The system uses GPS aided by an IMU to track movements. A wind vane-anemometer measures the current speed and direction of the wind. The boat is also equipped with a panoramic camera, sonar and hydrophones to aid with collision avoidance. The work details the process of designing both hardware and software implementation. Results from initial tests on a lake are presented and shows that the prototype system is capable of autonomous navigation between way-points.

In [14] a state observer for an unmanned aerial vehicle (UAV) is developed. The observer estimates the UAV's orientation and position by using accelerometers, gyroscopes, magnetometers, an ultra sound range sensor and a GNSS receiver. A similar state observer could be designed for use in this work to estimate the orientation and position of the RescueRunner. The small size of its sensors is an advantage, as the RescueRunner is a relatively small watercraft with little space for additional equipment.

In [17] a method for estimating relative horizontal positions of several GNSS receivers using their raw measurements is presented and compared to using their individual estimated absolute position to form the same relative position estimate. It is shown that sub meter relative positioning accuracy is possible. This is a clear improvement compared to the other method, which has a meter level accuracy. Implementing such a system along with the other tasks was not deemed feasible in the project's timeframe, however it remains as an interesting alternative to be explored in future work.

1.6 Thesis organization

The content of this thesis is organized as follows:

- Chapter 2 will give an introduction to the RescueRunner, as well as assess from a control perspective what physical quantities might be considered most relevant to estimate.
- Chapter 3 will describe the sensors that are used and explain why those sensors were chosen. It will also treat the background theory needed for the derivation of sensor fusion algorithms which are used to combine information from multiple sensors into one state vector.
- Chapter 4 will describe the derivation of a simple control scheme that can be used for testing of the complete system.
- Chapter 5 will describe the chosen system for wireless communication between

the RescueRunner and the lead boat.

- Chapter 6 will treat the implementation of the system on micro controllers.
- Chapter 7 will describe the tests conducted to evaluate the system's performance, as well as present their results.
- Chapter 8 will present a discussion of the results and conclusions.

2

Properties of the RescueRunner

This chapter will include a detailed description and specification of the RescueRunner. It will also more formally describe the task of making the RescueRunner autonomously follow a lead boat, as well as assess what physical quantities are relevant to estimate.

2.1 Introduction to the RescueRunner

The RescueRunner is a water scooter based on the powertrain of the Yamaha wave runner, modified with a soft plastic hull and an extended aft platform to make it suitable for rescue operations. The soft hull makes the RescueRunner durable against e.g. impacts from underwater rocks or other boats, enabling the operator to reach close to the people in need. Since the propulsion system is a pump-jet, there is no exposed propeller that can injure people in the water or take damage from grounding.

Like most pump-jet powered boats, the RescueRunner has no traditional rudder and turning is instead done by directing the output jet with a nozzle. Reversing is achieved in a similar manner by a reversing bucket that is lowered behind the output and redirects the water forward. This makes the total steering effect dependent on both the angle of the nozzle and the throttle. At low speeds the water provides little friction compared to the inertia of the RescueRunner, meaning that to stop its motion, be it rotational or translational, a thrust in the opposite direction has to be applied. At high speeds, the RescueRunner's flat bottom causes it to slide sideways as it makes a turn, which has to be compensated for by angling it further into the turn. Table 2.1 holds some figures of interest from the product leaflet [4]. Figure 2.1 shows how the soft plastic hull can be removed for reparations.

Table 2.1: Specifications of the RescueRunner.

Design	Fredrik Falkman
Developed by	SSRS
Engine	Yamaha four stroke, 140 [hp]
LOA*	3.6 [m]
Beam	1.5 [m]
Draught	0.3 [m]
Dry weight	350 [kg]
Load capacity	$>$ 400 [kg]
Top speed	40 [knots]
Fuel capacity	70 [l]
Range	$>$ 70 [NM]

*Length Over All

Figure 2.1: Drawing of the RescueRunner with its removable hull.

2.2 Adaption to automatic drive

In parallel with this work, a bachelor thesis is carried out to prepare the RescueRunner for autonomous drive [24]. This includes fitting actuators to perform the tasks of controlling the steering nozzle and throttle so that the RescueRunner can be controlled by microcontrollers. The nozzle control is achieved by rerouting the cable linking the nozzle to the handlebar to instead connect to a linear actuator. To preserve the ability to manually control the steering nozzle, a sensor is used to read the

handlebar's angle and the actuator's movement is set to follow. The throttle control is achieved by attaching an RC-style[1] servo to a cable connected in parallel with the manual throttle cable and the two can operate independently without interfering with each other. In its current form there is no electronic actuation of the reversing bucket. This will limit the possible control precision achievable at lower speeds, but will not affect the control system's performance at higher speeds, which is the area of interest in this work. The system's performance was deemed more than sufficient for the purposes of this work, with regards to both precision and responsiveness.

2.3 Formulating the control problem

Even though this work will not derive a complete control scheme for the RescueRunner the problem still needs consideration as it governs which physical quantities are required to be estimated.

When following, the RescueRunner should remain close to the lead boat. The reason for this is primarily to make it clear to other seafarers that the lead boat and RescueRunner move as one, so no one should attempt to go between the two boats. This also ensures that the RescueRunner arrives at the destination at the same time as the lead boat and not an arbitrary amount of time afterwards. This means that the RescueRunner must continuously adapt its path based on how the lead boat moves. A path built from sparsely spaced predefined way points as discussed in [25] can therefore not be used.

Another question is where the RescueRunner should place itself relative to the lead boat when following. Studying Figure 2.2, which shows the RescueRunner behind the lead boat, it can be observed that the wake of the lead boat provides an area of calmer water. Placing the RescueRunner in this area would reduce external disturbances and thus make stabilization of the system easier. This is good for high speed transport where rough sea may otherwise make it difficult for the small RescueRunner to keep up with the larger lead boat. However, if another boat is being towed this space will be occupied. It is therefore important that the rescue crew have control over where the RescueRunner is located. This might be done as a reference point in a 2D coordinate frame fixed to the lead boat, possibly in terms of a relative bearing (angle) and a distance. A few example reference points are shown in Figure 2.3, with their respective relative bearings drawn out. The RescueRunner still needs to calculate the path it should take to stay at this location, respecting that it cannot go through the lead boat, but the crew should not have to be involved in this.

[1] The type of servo motor typically found in a radio controlled (RC) model vehicle.

Figure 2.2: The figure shows how the lead boat provides an area of calmer water in its wake.

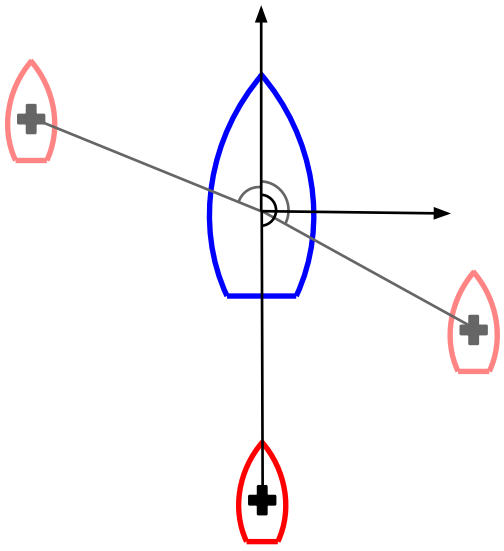


Figure 2.3: A drawing of how the RescueRunner's position relative to the lead boat can be changed by specifying different reference points. The RescueRunner should autonomously calculate the path it needs to take to stay at these positions.

While the control problem can be formulated in any frame, there is a clear advantage of viewing it from the perspective of the RescueRunner, as the effect of the control inputs (nozzle angle and throttle) are more consistent in this frame. This is also the perspective a human driver would have. To transform the reference point from the lead boat-fixed coordinate frame to the RescueRunner-fixed coordinate frame, both the relative orientation of the two frames and the distance between them must be known. There is also a clear benefit in knowing the two boats' velocities as this would allow a control system to respond quicker to changes and predict the boats' positions. Based on this, the state vector is defined in (2.1), where p_N denotes a latitudinal position, p_E denotes a longitudinal position, v_N denotes a latitudinal velocity, v_E denotes a longitudinal velocity and h denotes a heading relative to north. The RR and LB subscripts denote the states belonging to the RescueRunner or the lead boat respectively. How these quantities, hereafter known as system states can be estimated is further discussed in chapter 3.

$$\mathbf{x} = \begin{bmatrix} p_{N,RR} \\ p_{E,RR} \\ v_{N,RR} \\ v_{E,RR} \\ h_{RR} \\ p_{N,LB} \\ p_{E,LB} \\ v_{N,LB} \\ v_{E,LB} \\ h_{LB} \end{bmatrix} \tag{2.1}$$

3
Sensor fusion

This chapter describes the sensors that are used to estimate the relevant physical quantities (the system states) and explains why the specific set of sensors was chosen. It also presents theory used to develop the estimation algorithms, as well describing the process of deriving the system models used in these algorithms.

The state observer is physically separated into two modules - one mounted on board the lead boat and the other on the RescueRunner. Each module tracks its own orientation and movement in an Earth fixed frame. The orientation estimate is achieved by fusing gyroscopes, accelerometers and magnetometers. The positioning and movement is derived from the internally fused measurements of GNSS receivers.

3.1 Choosing sensors

An early step in constructing a state observer is choosing the array of sensors that should be used. This array should be able to give the necessary information needed to perform the task specified in chapter 2, namely to transform a point in the lead boat-fixed coordinate frame to a point in the RescueRunner-fixed coordinate frame. One way to achieve this is to estimate the location and orientation of the two coordinate frames in an intermediate third Earth-fixed frame [17], such as the World Geodetic System 1984 (WGS84) which is the reference frame used by GPS [19]. With two GNSS receivers, one placed on board the lead boat and the other on the RescueRunner, the relative distance between the two can be calculated from the difference of their respective coordinates. What remains is then to find the orientation of the two. This is suitably done using a magnetometer with the Earth's magnetic field as reference. However, due to magnetic inclination (the angle between the horizontal plane and the magnetic field), the tilt of the magnetometer must be known in order to determine the northward direction in a plane parallel to Earth's surface. [33] To accomplish this, the magnetometer is assisted by accelerometers and gyroscopes.

3.1.1 GNSS

A GNSS receiver uses radio signals from satellites orbiting the Earth to determine its position. The satellites continuously transmit their location and current time provided by high precision clocks. Given these signals from four or more satellites, the receiver can calculate its position on Earth. [20] GNSS receivers are commonly

found in smartphones and other devices used for navigation. The average error of relative position estimates lies around 3.65-6.5 m for commercially available receivers intended for personal use. [11]

3.1.2 MEMS rate gyroscopes & accelerometers

With the manufacturing techniques from the semiconductor production, very small and cheap implementations of various sensors can be achieved. These microelectromechanical system (MEMS) sensors usually consist of measuring components that gathers raw data aided by a microprocessor for data processing and communication. The rate gyroscope and accelerometer, measuring angular velocity and acceleration respectively in three orthogonal axes, are examples of sensors that can be produced in this way. These are commonly found inside smartphones, where they are used for e.g. orientation estimation [34]. Some sensors have the option to change the measurement range and resolution as well as the bandwidth of the internal filters to suit the application. Compared to sensors used in high performance applications, these cheaper sensors often experience notably biased or drifting measurements and is therefore not suitable for use in dead reckoning systems.

3.1.3 Magnetometer

Magnetometers are sensors used to measure magnetic field strength. In this work they will be used to help determine the sensor frame's orientation relative to the magnetic north. However, the magnetometers' measurements will be affected not only by Earth's magnetic field, but also nearby fields generated by electrical equipment. The errors in the measurements are commonly grouped into two categories, hard iron offsets and soft iron offsets. Soft iron offset is caused by material that easily align their own induced magnetic fields with an external one. This causes the local magnetic field to be skewed. Hard iron offsets are biases caused by constant magnetic fields that applies additively to Earth's magnetic field. One example of a hard iron offset source would be a speaker magnet in a smartphone. To cancel the soft and hard iron offsets and achieve correct measurements from a magnetometer, it needs to be calibrated.

3.2 Bayesian filtering

This section describes the fundamental principles of Bayesian filtering. As described in [28], Bayesian filtering uses Bayesian statistics to formulate the problem of state estimation as a statistical problem and uses probability distributions to represent the knowledge of the system state. The filtering is done by recursively updating these distributions using mathematical models of the system's dynamics and sensor measurements. [28]

Each iteration is preformed in two steps, a prediction step where previous knowledge of the state and a motion model is used to form a prediction, and an update step where the prediction is corrected using the information obtained in the latest

measurements. To make this possible, the system is assumed to behave as an unobserved Markov process, meaning that the state $\mathbf{x}$ of the system at time k is only dependent on the state at the previous time index $k-1$. (3.1) shows a mathematical formulation of a Markov process using conditional probability distributions.

$$p(\mathbf{x}_k|\mathbf{x}_{k-1}, \mathbf{x}_{k-2}, ..., \mathbf{x}_0) = p(\mathbf{x}_k|\mathbf{x}_{k-1}) \tag{3.1}$$

Figure 3.1 shows how the states and measurements at different time indexes are related. The arrows show how the knowledge about one state is propagated throughout the graph through the use of a motion model $\mathbf{f}(\mathbf{x})$ and measurement model $\mathbf{h}(\mathbf{x})$. Note however that the measurement model does not directly give the influence that the measurement should have on the state estimate. It describes how the measurement $\mathbf{y}_k$ should be distributed given a certain state $\mathbf{x}_k$. To find the new information about $\mathbf{x}_k$ that can be extracted from $\mathbf{y}_k$, Baye's rule is applied.

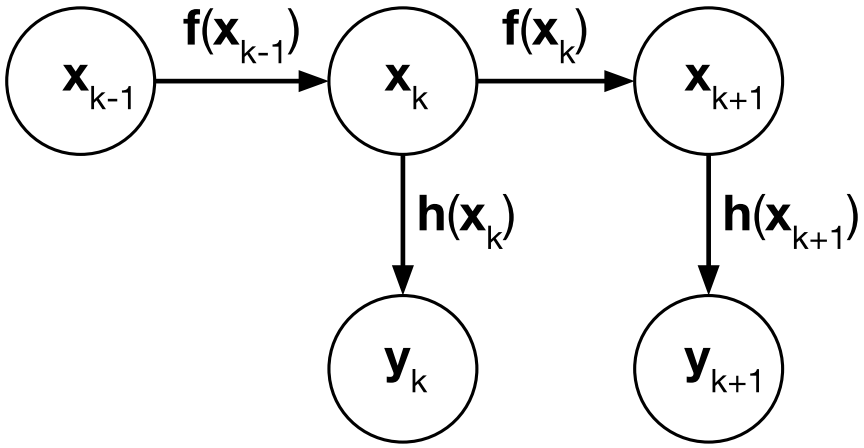


Figure 3.1: The figure shows how the state vector $\mathbf{x}$ and measurements $\mathbf{y}$ can be linked with each other over time using the motion model $\mathbf{f}(\mathbf{x})$ and measurement model $\mathbf{h}(\mathbf{x})$.

Baye's rule forms an integral part in Bayesian filtering as it describes the relationship between probabilities. Let P(A) and P(B) be the probabilities of the events A and B respectively without regard to each other. Furthermore, let P(A|B) be a conditional probability that represents the probability that event A occurs given that B is true, while P(B|A) denotes the conditional probability of event B occurring given that A is true. Then the four probabilities are related in the way shown in (3.2). This example uses discrete probabilities since they are more intuitive, but Baye's rule can also be applied to probability distributions.

$$\begin{aligned} \mathrm{P(B)P(A|B)} &= \mathrm{P(A)P(B|A)} \\ \Rightarrow \mathrm{P(A|B)} &= \frac{\mathrm{P(A)P(B|A)}}{\mathrm{P(B)}} \end{aligned} \tag{3.2}$$

In the filtering problem, the probability distribution $p(\mathbf{y}_k|\mathbf{x}_k)$ is used to denote the

measurement model $\mathbf{h}(\mathbf{x}_k)$ that describes the probability of each possible value of the measurement $\mathbf{y}_k$ given what is known about $\mathbf{x}_k$. All previous knowledge of $\mathbf{x}_k$ gathered from measurements up to $\mathbf{y}_{k-1}$, is denoted $p(\mathbf{x}_k|\mathbf{y}_{1:k-1})$, corresponding to the motion model $\mathbf{f}(\mathbf{x}_{k-1})$. Using (3.2), the distribution $p(\mathbf{x}_k|\mathbf{y}_{1:k})$ can be formed as (3.3).

$$p(\mathbf{x}_k|\mathbf{y}_{1:k}) = \frac{p(\mathbf{y}_k|\mathbf{x}_k)p(\mathbf{x}_k|\mathbf{y}_{1:k-1})}{p(\mathbf{y}_k|\mathbf{y}_{1:k-1})} \tag{3.3}$$

Note that $p(\mathbf{x}_k|\mathbf{y}_{1:k})$ is also dependent on $p(\mathbf{y}_k|\mathbf{y}_{1:k-1})$. However since the latter is independent of $\mathbf{x}$ it can be replaced by a scaling factor, meaning that (3.3) can be rewritten as (3.4).

$$p(\mathbf{x}_k|\mathbf{y}_{1:k}) = \alpha p(\mathbf{y}_k|\mathbf{x}_k)p(\mathbf{x}_k|\mathbf{y}_{1:k-1}) \Leftrightarrow p(\mathbf{x}_k|\mathbf{y}_{1:k}) \propto p(\mathbf{y}_k|\mathbf{x}_k)p(\mathbf{x}_k|\mathbf{y}_{1:k-1}) \tag{3.4}$$

Meaning that a distribution that is proportional to $p(\mathbf{x}_k|\mathbf{y}_{1:k})$ can be found. As probability distributions must conform to certain properties such as having a unity integral $\int_{-\infty}^{\infty} p(\mathbf{x}_k|\mathbf{y}_{1:k}) = 1$, the result only needs to be normalized to be equivalent to $p(\mathbf{x}_k|\mathbf{y}_{1:k})$.

What remains to do is to decide how an optimal point estimate of the system state can be extracted from $p(\mathbf{x}_k|\mathbf{y}_{1:k})$. This in turn requires a cost function to be formulated that expresses what is considered optimal. The most common choice for this is the estimate that gives the minimum mean square error (MMSE), which is the mean of $p(\mathbf{x}_k|\mathbf{y}_{1:k})$.[28] Another imaginable alternative is to choose the state that has the maximum probability, which may be favorable in the case where being a little wrong is considered just as bad as being very wrong. This is called the maximum a posteriori (MAP) estimator. In this work, the MMSE estimator will be used.

Above is a short summary of the basic principles shared by all Bayesian filter. What separates the different practical implementations of this theory is how these probability distributions are represented as numbers, what approximations are made and what assumptions are necessary. One famous implementation is the Kalman filter which assumes a linear system. In the case where all errors are also Gaussian distributed $p(\mathbf{x}_k|\mathbf{y}_{1:k})$ can be found exactly, and in this case the Kalman filter is the optimal estimator in the MMSE sense. In other cases such as if the system includes nonlinear dynamics, the distributions have to be approximated numerically. The extended Kalman filter (EKF) does this by linearizing the system in each iteration around the latest estimated state. Gaussian filters such as the Gauss-Hermite Kalman filter (GHKF) or cubature Kalman filter (CKF) uses Gaussian assumed density approximations. Another method is the particle filter which uses many Monte Carlo samples (or particles) generated by random number generators and distributed as the latest $\mathbf{x}$ estimate. These are then propagated through the system model and from this, quantities such as means can be calculated. Particle filters are versatile and can be applied to highly nonlinear systems but typically require high processing power as many particles have to be used to achieve an accurate result.

3.3 Kalman filters and the extended Kalman filter

The Kalman filter, also known as the linear quadratic estimator, is a common method for fusing information from several sensors and estimating system states for linear systems.[28] Apart from assuming linear motion and measurement models it also uses Gaussian distributions to represent the probability densities. The key aspect of this is that linear operations on Gaussian variables will result in a new set of Gaussian variables and therefore no further approximations of the densities are needed. The filtering calculations can also be derived exactly. [12]

In order to operate, the Kalman filter requires a model of the system. In the case of the regular linear Kalman filter, this model is constructed on the form shown in (3.5) and (3.6), where k represents the current time index, $\mathbf{x}_k$ is the current state vector, $\mathbf{x}_{k-1}$ is the state vector of the previous time index, $\mathbf{y}_k$ is the vector of measurements, $\mathbf{A}$ is the system's transition matrix representing how the system's states affect one another over time and $\mathbf{H}$ is the system's measurement model matrix representing how the measurements relate to the system states. $\mathbf{w}$ and $\mathbf{r}$ are independent zero mean Gaussian random variables with covariances $\mathbf{W}$ and $\mathbf{R}$ respectively, representing the uncertainty of the model and the measurements respectively. [12]

$$\mathbf{x}_k = \mathbf{A}\mathbf{x}_{k-1} + \mathbf{w}, \quad \mathbf{w} \sim \mathcal{N}(0, \mathbf{W}) \tag{3.5}$$

$$\mathbf{y}_k = \mathbf{H}\mathbf{x}_k + \mathbf{r}, \quad \mathbf{r} \sim \mathcal{N}(0, \mathbf{R}) \tag{3.6}$$

The probability distribution of the state vector $\mathbf{x}$ is stored by its mean $\hat{\mathbf{x}}$ and covariance $\mathbf{P}$ as in (3.7).

$$\mathbf{x} \sim \mathcal{N}(\hat{\mathbf{x}}, \mathbf{P}) \tag{3.7}$$

The filtering algorithm recursively uses two steps, the prediction step and the update step, to produce estimates of the state vector's mean $\hat{\mathbf{x}}_k$ and covariance matrix $\mathbf{P}_k$. The prediction step starts with $p(\mathbf{x}_{k-1}|\mathbf{y}_{1:k-1})$ from the previous iteration with properties as in (3.8) and uses the system's motion model (3.5) to predict the state vector's next value.

$$p(\mathbf{x}_{k-1}|\mathbf{y}_{1:k-1}) = \mathcal{N}(\mathbf{x}_{k-1}; \hat{\mathbf{x}}_{k-1|k-1}, \mathbf{P}_{k-1|k-1}) \tag{3.8}$$

Here $k-1|k-1$ indicates the time index of the variable and that all gathered information up to $k-1$ has been used. Using (3.5), (3.8) and knowledge about linear combinations of Gaussian variables, (3.9) can be formed.

$$p(\mathbf{x}_k|\mathbf{y}_{1:k-1}) = \mathcal{N}(\mathbf{x}_k; \mathbf{A}\hat{\mathbf{x}}_{k-1|k-1}, \mathbf{A}\mathbf{P}_{k-1|k-1}\mathbf{A}^\intercal + \mathbf{W}) \tag{3.9}$$

Thus the prediction step becomes (3.10).

$$\begin{aligned}\hat{\mathbf{x}}_{k|k-1} &= \mathbf{A}\hat{\mathbf{x}}_{k-1|k-1} \\ \mathbf{P}_{k|k-1} &= \mathbf{A}\mathbf{P}_{k-1|k-1}\mathbf{A}^\intercal + \mathbf{W}\end{aligned} \tag{3.10}$$

The update step can be found by considering the joint distribution (3.11)

$$\begin{bmatrix}\mathbf{x}_k \\ \mathbf{y}_k\end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix}\hat{\mathbf{x}}_{k|k-1} \\ \mathbf{H}\hat{\mathbf{x}}_{k|k-1}\end{bmatrix}, \begin{bmatrix}\mathbf{P}_{k|k-1} & \mathbf{P}_{k|k-1}\mathbf{H}_k^\intercal \\ \mathbf{H}_k\mathbf{P}_{k|k-1} & \mathbf{H}_k\mathbf{P}_{k|k-1}\mathbf{H}_k^\intercal + \mathbf{R}\end{bmatrix}\right) \tag{3.11}$$

From this, the conditional probability distribution $p(\mathbf{x}_k|\mathbf{y}_k)$ can be found as (3.12).

$$p(\mathbf{x}_k|\mathbf{y}_k) = \mathcal{N}\left(\mathbf{x}_k; \hat{\mathbf{x}}_{k|k-1} + \mathbf{K}_k\mathbf{v}_k, \mathbf{P}_{k|k-1} - \mathbf{K}_k\mathbf{S}_k\mathbf{K}_k^\intercal\right) \tag{3.12}$$

where $\mathbf{K}_k$, $\mathbf{v}_k$ and $\mathbf{S}_k$ are defined as in (3.13).

$$\begin{aligned}\mathbf{K}_k &= \mathbf{P}_{k|k-1}\mathbf{H}^\intercal\mathbf{S}_k^{-1} \\ \mathbf{v}_k &= \mathbf{y}_k - \mathbf{H}\hat{\mathbf{x}}_{k|k-1} \\ \mathbf{S}_k &= \mathbf{H}\mathbf{P}_{k|k-1}\mathbf{H}_k^\intercal + \mathbf{R}\end{aligned} \tag{3.13}$$

The final formulation of the update step is shown in (3.14).

$$\begin{aligned}\hat{\mathbf{x}}_{k|k} &= \hat{\mathbf{x}}_{k|k-1} + \mathbf{K}_k\mathbf{v}_k \\ \mathbf{P}_{k|k} &= \mathbf{P}_{k|k-1} - \mathbf{K}_k\mathbf{S}_k\mathbf{K}_k^\intercal\end{aligned} \tag{3.14}$$

The uncertainty of the Kalman filter's estimate increases in the prediction step (3.10), represented by an increase in the covariance matrix $\mathbf{P}_k$. This is due to the prediction simply being a guess of what the states might be, based on the motion model of the system. The covariance matrix decreases in the update step (3.14), representing an increased certainty of the estimate brought on by the use of sensor measurements to correct the prediction. It should also be noted that for a linear time invariant system where measurements are sampled at a consistent rate, the covariance $\mathbf{P}$ will converge towards a steady state value, which means that the Kalman gain $\mathbf{K}$ can be calculated beforehand for this steady state $\mathbf{P}$ and used as a constant parameter. This reduces the number of operations needed as only $\hat{\mathbf{x}}$ needs to be calculated.

3.3.1 The extended Kalman filter

The extended Kalman filter (EKF) is the nonlinear version of the Kalman filter. It adapts the Kalman filter's basic principles and attempts to apply them to nonlinear systems through repeated linearizations around the current estimate. [12]

Just like the Kalman filter, the EKF requires a model of the system. However, the EKF treats nonlinear system which means that the linear system model used in

(3.5) and (3.6) is no longer applicable. Instead, the system model is constructed on the form shown in (3.15), where $\mathbf{f}(\mathbf{x}_{k-1})$ is the nonlinear system's motion model and $\mathbf{h}(\mathbf{x}_k)$ is the nonlinear measurement model of the system.

$$\begin{aligned} \mathbf{x}_k &= \mathbf{f}(\mathbf{x}_{k-1}) + \mathbf{w}, \quad & \mathbf{w} &\sim \mathcal{N}(0, \mathbf{W}) \\ \mathbf{y}_k &= \mathbf{h}(\mathbf{x}_k) + \mathbf{r}, & \mathbf{r} &\sim \mathcal{N}(0, \mathbf{R}) \end{aligned} \tag{3.15}$$

With the system models in place, the EKF then operates similarly to the Kalman filter with the exception that it linearizes $\mathbf{f}(\mathbf{x}_{k-1})$ and $\mathbf{h}(\mathbf{x}_k)$ around the current estimate before applying the prediction and update steps respectively by using a first order Taylor expansion.

The mathematical formulation of the EKF prediction step is shown in (3.16), where $\mathbf{f}'(\hat{\mathbf{x}}_{k-1|k-1})$ is the motion model's derivative at the previous state estimate.

$$\begin{aligned} \hat{\mathbf{x}}_{k|k-1} &= \mathbf{f}(\hat{\mathbf{x}}_{k-1|k-1}) \\ \mathbf{P}_{k|k-1} &= \mathbf{f}'(\hat{\mathbf{x}}_{k-1|k-1})\mathbf{P}_{k-1|k-1}\mathbf{f}'(\hat{\mathbf{x}}_{k-1|k-1})^\intercal + \mathbf{W} \end{aligned} \tag{3.16}$$

A mathematical formulation of the EKF update step is shown in (3.17), where $\mathbf{h}'(\hat{\mathbf{x}}_{k|k-1})$ is the measurement model's derivative at the predicted state.

$$\begin{aligned} \hat{\mathbf{x}}_{k|k} &= \hat{\mathbf{x}}_{k|k-1} + \mathbf{K}_k\mathbf{v}_k \\ \mathbf{P}_{k|k} &= \mathbf{P}_{k|k-1} - \mathbf{K}_k\mathbf{S}_k\mathbf{K}_k^\intercal \\ \mathbf{v}_k &= \mathbf{y}_k - \mathbf{h}(\hat{\mathbf{x}}_{k|k-1}) \\ \mathbf{S}_k &= \mathbf{h}'(\hat{\mathbf{x}}_{k|k-1})\mathbf{P}_{k|k-1}\mathbf{h}'(\hat{\mathbf{x}}_{k|k-1})^\intercal + \mathbf{R} \\ \mathbf{K}_k &= \mathbf{P}_{k|k-1}\mathbf{h}'(\hat{\mathbf{x}}_{k|k-1})^\intercal\mathbf{S}_k^{-1} \end{aligned} \tag{3.17}$$

Contrary to the Kalman filter, the EKF is not an optimal estimator since it uses linearized approximations of the system to ensure that the filtering distributions remain Gaussian distributed. It requires that the Jacobians $\mathbf{f}'(\mathbf{x})$ and $\mathbf{h}'(\mathbf{x})$ can be formulated, which may not be possible for some system models. However, it is widely used, as it provides a computationally efficient filter given its performance for almost linear systems and will be used in this work to estimate the orientation of the RescueRunner. [28]

3.4 Frames of reference

In the following sections multiple coordinate systems will be used as frames of reference for vectors and orientation. To make it clear in which frame of reference each quantity is defined, a left-side superscript will be used. For example, a vector holding the nominal gravitational acceleration ${}^{NWU}\mathbf{g} = [0 \;\; 0 \;\; -9.82]^\intercal$ is expressed in a North-West-Up (NWU) system. Another frame that will be used extensively is the sensor-fixed frame that will be denoted by an ^{S}. The sensor-fixed frame will always

stay fixed with the boat, aligning the x axis with the stern-bow (forward) direction, the y axis with the starboard-port (left) direction and the z axis with the upward direction. This is the frame that most of the measurements will be expressed in and a large part of this chapter will be dedicated to finding the orientation of the sensor frame relative to the NWU frame.

3.5 Quaternions

In this work quaternions will be used to express rotations in 3-dimensional space. Quaternions can be regarded as an ordered set of four real numbers similarly to how complex numbers can be seen as an ordered set of two real numbers [5]. A quaternion q is defined as (3.18) [23].

$$q = a + bi + cj + dk \tag{3.18}$$

Where a, b, c, d are real numbers and i, j, k are imaginary units which are subject to the following laws (3.19) of combination regarding their squares and products [15].

$$\begin{gathered} i^2 = j^2 = k^2 = -1 \\ ij = k,\ jk = i,\ ki = j, \\ ji = -k,\ kj = -i,\ ik = -j \end{gathered} \tag{3.19}$$

As stated above quaternions can be used to represent rotations in 3D and can therefore be used to hold the orientation of an object relative to a given coordinate system. Similarly to how multiplying two complex numbers can be seen as a rotation in 2D (e.g. multiplying by $(0 + 1i)$ can be seen as a rotation by $\pi/2$ radians), multiplying two quaternions can be seen as applying the rotation and scale of one quaternion to the other. If the quaternions are of norm one, so called unit quaternions, the resulting product will also have a norm of one and the scale will remain the same. One thing to note is that quaternion multiplication is not commutative (the property that $xy = yx$, which holds for real and complex numbers), see for instance $ij = k$ while $ji = -k$ in (3.19), i.e. it matters which rotation is performed first.

Quaternions are often preferable to use over Euler angles to represent orientation since the latter can have problems with discontinuities, gimbal lock [32] and that a rotation defined by one set of Euler angles is not unique, meaning that the final rotation can be achieved by several different combinations. A drawback of the quaternions is that the unit length must be enforced and that they are less intuitive to use than Euler angles. [12]

Alternatively, a quaternion can always be written in terms of an axis-angle representation (3.20). q_0, q_1, q_2, q_3 will hereafter be used to denote the four real numbers

of the quaternion, corresponding to a, b, c, d in (3.18).

$$q = \begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{bmatrix} = \begin{bmatrix} \cos\left(\frac{1}{2}\alpha\right) \\ \sin\left(\frac{1}{2}\alpha\right) \begin{bmatrix} \hat{v}_x \\ \hat{v}_y \\ \hat{v}_z \end{bmatrix} \end{bmatrix} \tag{3.20}$$

Where a rotation by q can be seen as a rotation by angle α around the axis containing vector $\hat{v}$. This representation can be useful to relate the numbers of the quaternion to a rotation.

To convert a quaternion into a rotation matrix the formula in (3.21) can be used [16]

$$Q(q) = \begin{bmatrix} q_0^2 + q_1^2 - q_2^2 - q_3^2 & 2q_1q_2 - 2q_0q_3 & 2q_1q_3 + 2q_0q_2 \\ 2q_1q_2 + 2q_0q_3 & q_0^2 - q_1^2 + q_2^2 - q_3^2 & 2q_2q_3 - 2q_0q_1 \\ 2q_1q_3 - 2q_0q_2 & 2q_2q_3 + 2q_0q_1 & q_0^2 - q_1^2 - q_2^2 + q_3^2 \end{bmatrix} \tag{3.21}$$

This rotation matrix can then be used to rotate any vector in a reference frame to the local frame of an object, whose orientation in the same reference frame is expressed with the quaternion. One application of this could be to rotate a nominal gravity vector ${}^{NWU}\mathbf{g} = [0 \quad 0 \quad -9.82]^\intercal$ to the local frame of the accelerometers that measure this quantity. The difference between the rotated nominal ${}^{S}\mathbf{g}$ and the measured one can then be used to update the current orientation estimate (i.e. the quaternion).

Another thing to note is that the rotation described by a quaternion q is identical to the one described by $-q$. To see this, one can notice that all occurrences of q_i appear in pairs in (3.21), meaning that a common factor -1 will always be squared.

The quaternion conjugate can be written as (3.22) [16].

$$q^* = a - bi - cj - dk \tag{3.22}$$

The inverse of a quaternion can then be expressed as (3.23) [16].

$$q^{-1} = \frac{q^*}{|q|^2} \tag{3.23}$$

3.6 The orientation filter

The orientation filter is formulated as an EKF, meaning that the system is linearized around the latest estimate to achieve a linear system to which the theory presented in sections 3.2 and 3.3 can be applied. The orientation is expressed using quaternions, from which the state of interest (the boat's heading) can be extracted, along with its pitch and roll if so desired. To enable the use of the EKF structure, a motion

model of the system must be derived as well as measurement models for the sensors.

3.6.1 Motion model of the system using MEMS rate gyroscopes as inputs

The nonlinear kinematic model of a quaternion relating angular velocities $\omega = [\omega_x, \omega_y, \omega_z]$ to the quaternion time derivative, is presented in [13] as:

$$\dot{q} = -\frac{1}{2}S(\omega)q = -\frac{1}{2}\bar{S}(q)\omega \tag{3.24}$$

In which the matrices S and $\bar{S}$ are:

$$S(\omega) = \begin{bmatrix} 0 & -\omega_x & -\omega_y & -\omega_z \\ \omega_x & 0 & -\omega_z & \omega_y \\ \omega_y & \omega_z & 0 & -\omega_x \\ \omega_z & -\omega_y & \omega_x & 0 \end{bmatrix} \tag{3.25}$$

$$\bar{S}(q) = \begin{bmatrix} -q_1 & -q_2 & -q_3 \\ q_0 & q_3 & -q_2 \\ -q_3 & q_0 & q_1 \\ q_2 & -q_1 & q_0 \end{bmatrix} \tag{3.26}$$

The angular velocities are measured by MEMS rate gyroscopes. From (3.24) we can see that if the gyroscope measurements are considered as inputs to the system, rather than measurements relating to the current rate of change, then $\dot{q}$ does not need to be included as a state, reducing the computational complexity of the filter. The prediction step of the filter can then be found by discretizing (3.24). This is done by assuming the input to be piece-wise constant between samples and using forward Euler approximation. While this will introduce some errors in the model, their effects can be mitigated by sampling the gyroscopes often. [12] The resulting prediction step of the EKF is presented in (3.27) and (3.28) where T is the sampling time.

$$\hat{\mathbf{x}}_{k|k-1} = \left(I + \frac{1}{2}S(\boldsymbol{\omega}_{k-1})T\right)\hat{\mathbf{x}}_{k-1|k-1} \tag{3.27}$$

$$\begin{aligned}\mathbf{P}_{k|k-1} = &\left(I + \frac{1}{2}S(\boldsymbol{\omega}_{k-1})T\right)\mathbf{P}_{k-1|k-1}\left(I + \frac{1}{2}S(\boldsymbol{\omega}_{k-1})T\right)^{\intercal} \\ &+ \left(\frac{1}{2}\bar{S}(\hat{\mathbf{x}}_{k-1|k-1})T\right)\mathbf{R}_v\left(\frac{1}{2}\bar{S}(\hat{\mathbf{x}}_{k-1|k-1})T\right)^{\intercal}\end{aligned} \tag{3.28}$$

3.6.2 Measurement model for MEMS accelerometers

The general sensor model with additive noise for an EKF [28] has the form:

$$\mathbf{y}_k = \mathbf{h}(\mathbf{x}_k) + \mathbf{r} \tag{3.29}$$

The measurement model for the accelerometers is given as [12]:

$$ {}^{S}\mathbf{y}_k^a = Q^\intercal(\mathbf{q}_k)({}^{NWU}\mathbf{g} + {}^{NWU}\mathbf{f}_k^a) + \mathbf{e}^a \tag{3.30} $$

Where ${}^{NWU}\mathbf{g}$ is the nominal gravity vector. Comparing (3.29) and (3.30) we can identify the measurement model $\mathbf{h}(\mathbf{x}_k)$ as $Q^\intercal(\mathbf{q}_k){}^{NWU}\mathbf{g}$ and the measurement noise $\mathbf{r}$ as $\mathbf{e}^a$. ${}^{NWU}\mathbf{f}_k^a$, which denotes accelerations other than gravity, is assumed to be zero. To ensure this is true, the accelerometer update step will only be performed when $||{}^{S}\mathbf{y}_k^a|| \approx ||{}^{NWU}\mathbf{g}||$.

With the measurement model defined, the equations presented in (3.17) can be used as the accelerometers' update step. The Jacobian $\mathbf{h}'(\mathbf{x})$ evaluated at $\hat{\mathbf{x}}_{k|k-1}$ is built from the partial derivatives of the sensor-to-world-frame rotation matrix $Q(\mathbf{q})$.

$$ \frac{dQ(\mathbf{q})}{dq_0} = 2\begin{bmatrix} 2q_0 & -q_3 & q_2 \\ q_3 & 2q_0 & -q_1 \\ -q_2 & q_1 & 2q_0 \end{bmatrix} \tag{3.31} $$

$$ \frac{dQ(\mathbf{q})}{dq_1} = 2\begin{bmatrix} 2q_1 & q_2 & q_3 \\ q_2 & 0 & -q_0 \\ q_3 & q_0 & 0 \end{bmatrix} \tag{3.32} $$

$$ \frac{dQ(\mathbf{q})}{dq_2} = 2\begin{bmatrix} 0 & q_1 & q_0 \\ q_1 & 2q_2 & q_3 \\ -q_0 & q_3 & 0 \end{bmatrix} \tag{3.33} $$

$$ \frac{dQ(\mathbf{q})}{dq_3} = 2\begin{bmatrix} 0 & -q_0 & q_1 \\ q_0 & 0 & q_2 \\ q_1 & q_2 & 2q_3 \end{bmatrix} \tag{3.34} $$

With this we can form $\mathbf{h}'(\mathbf{q}_k)$ as:

$$ \mathbf{h}'(\mathbf{q}_k) = \begin{bmatrix} \left(\frac{dQ(\hat{\mathbf{q}})}{dq_0}\right)^\intercal {}^{NWU}\mathbf{g} & \left(\frac{dQ(\hat{\mathbf{q}})}{dq_1}\right)^\intercal {}^{NWU}\mathbf{g} & \left(\frac{dQ(\hat{\mathbf{q}})}{dq_2}\right)^\intercal {}^{NWU}\mathbf{g} & \left(\frac{dQ(\hat{\mathbf{q}})}{dq_3}\right)^\intercal {}^{NWU}\mathbf{g} \end{bmatrix} \tag{3.35} $$

3.6.3 Measurement model for the magnetometer

Similarly to ${}^{NWU}\mathbf{g}$, a ${}^{NWU}\mathbf{m}$ could be used for the nominal magnetic field. However, one problem using this approach is that unlike ${}^{NWU}\mathbf{g}$, ${}^{NWU}\mathbf{m}$ will vary noticeably with both location and over time. This is extra problematic in Sweden where the magnetic inclination is around 70° [10], which means that only a small component of Earth's magnetic field lies in the horizontal northward direction. An incorrect estimation of the nominal ${}^{NWU}\mathbf{m}$ may therefore result in some part of the downward pointing component being erroneously taken to belong to the northward pointing component.

In [33], a strategy is proposed where the measurement ${}^{S}\mathbf{m}$ is first rotated to align it with the horizontal plane ${}^{NWU}xy$ to cancel tilt of the sensors. This is done using an estimate of the sensor tilt provided by the accelerometer measurements. Then only the normalized components of the measurement lying in the horizontal plane is used,

resulting in that the magnetometer measurements only affect the estimates of the bearing (rotation about ${}^{NWU}z$) without affecting the estimates of the tilt. The filter used in [33] employs a different structure to the EKF and it does not use gyroscopes, as the filter is meant for static or slow-moving bodies, but nonetheless the principles can be applied in this work. If the tilt relative to the horizontal plane could be extracted from the current orientation estimate, then the rotated and normalized magnetic measurement could be used to correct the remaining yaw component.

First, a quaternion holding only the tilt of the sensor fixed frame must be created. This is done by first finding a quaternion $\mathbf{q}_{yaw}$ that rotates $\mathbf{q}$ about the *vertical* axis ${}^{NWU}z$ such that the ${}^{S}x$ axis would be parallel to the ${}^{NWU}xz$ plane, meaning that $\mathbf{q}_{yaw}$ will hold the current estimate of the bearing. The bearing is found by applying the $atan2$-function to the ${}^{NWU}x$ and ${}^{NWU}y$ components of the vector $[1\ 0\ 0]^\intercal$ after being rotated by $\mathbf{q}$. (3.20) can then be used to convert the angle into a quaternion. Since it only holds a rotation about ${}^{NWU}z$ it will have the form $\mathbf{q}_{yaw} = [q_0\ 0\ 0\ q_3]^\intercal$.

To subtract the yaw rotation from $\mathbf{q}$, the latter should be multiplied by $\mathbf{q}_{yaw}^{-1}$. Note that $|\mathbf{q}_{yaw}| = 1$, meaning that $\mathbf{q}_{yaw}^{*}$ can be used instead, recall (3.23). The quaternion expressing only the tilt of the sensor $\mathbf{q}_{tilt}$ can then be calculated as seen in (3.36).

$$\mathbf{q}_{tilt} = \mathbf{q}_{yaw}^{-1}\mathbf{q} = \mathbf{q}_{yaw}^{*}\mathbf{q} \tag{3.36}$$

The measured magnetic field $\mathbf{y}_k^m$ is then rotated by $\mathbf{q}_{tilt}$ as in (3.37) to cancel the tilt of the sensor.

$$\mathbf{y}_k^{m'} = Q(\mathbf{q}_{tilt})\mathbf{y}_k^m \tag{3.37}$$

This rotated $\mathbf{y}_k^{m'}$ will then be in a frame in which the xy plane aligns with the ${}^{NWU}xy$ plane. The z component of $\mathbf{y}_k^{m'}$ is then set to zero to ensure that the magnetometer's update step does not affect the estimated tilt. After this, the update step is performed similarly to that of the accelerometer, with the exception that $\mathbf{q}_{yaw}$ is used in place of $\mathbf{q}$ and $\mathbf{y}_k^{m'}$ is used as measurement. The final measurement model for the magnetometer is presented in (3.38), where $\mathbf{e}^{m'}$ is the measurement noise of the magnetometer rotated in the same way as $\mathbf{y}_k^{m'}$.

$$\mathbf{y}_k^{m'} = Q^\intercal(\mathbf{q}_{yaw})^{NWU}\mathbf{m} + \mathbf{e}^{m'} \tag{3.38}$$

This measurement model is then used in the EKF equations presented in (3.17) as the magnetometer's update step.

3.6.4 Compensating for magnetic declination

Magnetic declination is the name of the difference in angle between the geographical north and the magnetic north. It varies both with location and over time and can reach values of ± 10 degrees within Europe. [9] This causes a difference in what the GNSS interprets as north and what the magnetometer interprets as north, and could cause problems when using both sensor types in a control system. A similar problem can come from incorrect orientation of the sensors on board the RescueRunner,

meaning that the sensors' orientation do not perfectly align with its front.

Assuming the direction of the RescueRunner's front is the same as the direction of its motion (i.e. that there is little to no sway), the angle of the GNSS' velocity estimates can be used to account for these errors by continuously slowly correcting the orientation estimates. This causes the magnetometer's estimate of the northward direction to conform to that of the GNSS. How the correction angle is estimated and used is shown in (3.39), where ψ_{GNSS} is the GNSS' heading estimate, ψ_q is the orientation filter's heading estimate without compensation, ψ_{corr} is the correction angle, α is a factor to decide how fast the correction angle change, and ψ is the resulting heading estimate.

$$\begin{aligned} \psi &= \psi_q + \psi_{corr} \\ \psi_{corr} &= \psi_{corr} + \alpha(\psi_{GNSS} - \psi) \end{aligned} \tag{3.39}$$

3.7 Implementation in MATLAB

Before the algorithms above were implemented on microcontrollers they were first tested in MATLAB. The raw sensor data was collected by a microcontroller and forwarded to MATLAB. This simplified the visualization of results and facilitated troubleshooting by removing the need to repeatedly upload programs to the microcontroller.

The first step involved establishing the communication between the microcontroller and the sensors. A program was uploaded to the microcontroller that allowed MATLAB to directly send commands to the sensor chips and collect the returned data [21]. Since some of the sensors require that the measurement data registers are read in sequence and in a single read request, a small addition to MATLAB's `readRegister` function was necessary. For more details, refer to appendix A. This made configuring the sensors faster and easier since the process does require some trial and error. The sensors and microcontroller were mounted on a board as seen in Figure 3.2. To avoid later confusion the axis of the measurement for each sensor were made to conform with the same reference frame.

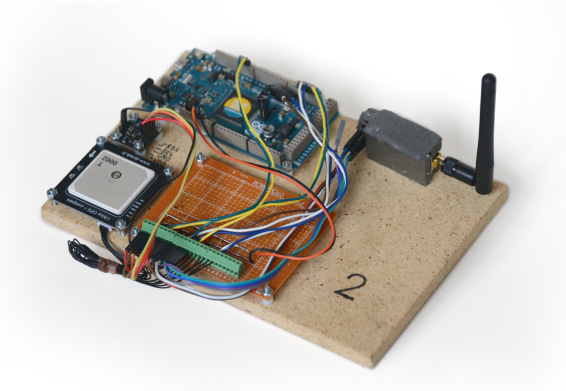


Figure 3.2: An image of one of the test boards made to hold the hardware during initial tests.

The next step was to analyse the noise characteristics for the sensors. This was done by logging data from the sensors and from this decide their respective covariance matrices for use in the EKF. Once this was done, the work of implementing the filter began. Since MATLAB is specialized in matrix operations, the code can be written similarly to how the the equations were formulated above. Implementation was done one sensor at a time, starting with the gyroscope and the prediction step while the update steps for the accelerometer and magnetometer were simply omitted. After ensuring a functioning prediction step, the update steps were added one at a time, verifying a proper behavior before adding the next. A helpful tool when developing the filter is a way to visualize the estimated orientation, Figure 3.3 shows an example of this. This particular code is derived from an exercise in the course *Sensor Fusion (TSRT14)* at Linköping University [18].

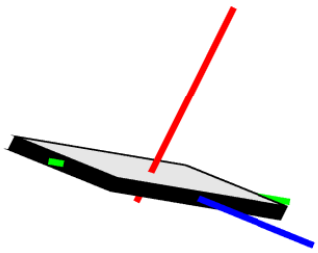


Figure 3.3: Figure showing the tool used for visualizing the orientation estimate.

Once the performance of the filter in MATLAB was deemed satisfactory, work began on translating the code to the microcontroller. This is presented in chapter 6. By comparing the numerical result of the microcontroller code to that of the MATLAB filter, the process of verifying the translated code was simplified. A system was developed to transfer data from the microcontroller to MATLAB over USB so that the functions for visualization and monitoring could still be used.

4

Derivation of a simple control scheme

This chapter will present a simple control scheme aimed at making the RescueRunner follow a lead boat autonomously. Focus lies on its performance at higher speeds, since lower speeds are seldom used when travelling to the scene of an accident. The control scheme was intentionally kept simple since it was made primarily with the intention of testing the performance of the state observer, as well as due to time constraints. This means that the behavior of the control system presented in this chapter will differ from the one described in section 2.3. The behavior is simply that the RescueRunner continuously aims itself towards the last known location of the lead boat and attempts to keep a set distance. This should result in the RescueRunner trailing behind the lead boat as it moves.

The RescueRunner has two control inputs, the jet propulsion system's nozzle angle and its throttle position. The reverse bucket is omitted since it is only used for low speed maneuvers. Additionally the control structure will assume that the two inputs affect the RescueRunner independent of each other. This behavior would be ill suited for low speeds, as a low throttle would hinder the RescueRunner's ability to turn as mentioned in section 2.1. However, at high speeds the throttle should never reach levels low enough to significantly affect the nozzle angle's effect on turning the RescueRunner.

4.1 Controlling the nozzle angle

The nozzle angle is controlled with the objective of keeping the lead boat in front of the RescueRunner. This behavior is achieved by applying a PID controller on the angle to the lead boat from the RescueRunner's point of view, denoted $\psi_{RR \to LB}$. The angle is calculated using both boats' positions and the estimated heading of the RescueRunner as seen in (4.1), where ψ_{RR} is the RescueRunner's estimated heading relative to north, while d_N and d_W are the northward and westward distances between the two boats respectively.

$$\psi_{RR \to LB} = \psi_{RR} - arctan\left(\frac{d_W}{d_N}\right) \tag{4.1}$$

The output of the nozzle angle PID controller y_ψ is a number between -1 and 1

signifying a full left turn and a full right turn respectively. It is calculated as seen in (4.2), where the control error e_ψ is equal to the angle $\psi_{RR \to LB}$, since the desired angle is zero, meaning the lead boat is located exactly in front of the RescueRunner. K_p, K_i and K_d are the proportional, integral and derivative weights respectively. A block diagram of the nozzle angle control system is shown in Figure 4.1.

$$\begin{aligned} y_\psi &= K_p \cdot e_\psi(t) + K_i \cdot \int_0^t e_\psi(\tau)d\tau + K_d \cdot \frac{de_\psi(t)}{dt} \\ e_\psi &= \psi_{RR \to LB} \end{aligned} \tag{4.2}$$

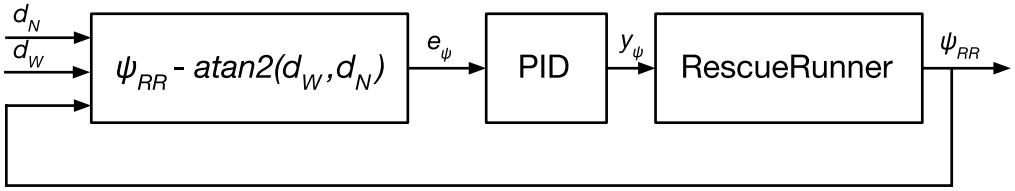


Figure 4.1: Block diagram over the nozzle angle control system.

4.2 Controlling the throttle

The throttle is controlled to achieve a speed similar to the lead boat, with an additional dependency on its distance from the RescueRunner. This behavior is achieved by applying a PID controller on the velocity of the RescueRunner, where the desired velocity is a sum of the lead boat's velocity and the output of a PID controller applied on the distance between the boats. This is presented as a block diagram in Figure 4.2, where e_d is the distance error, i.e. the difference between the actual distance between the two boats and a reference distance d_{ref} representing the ideal.

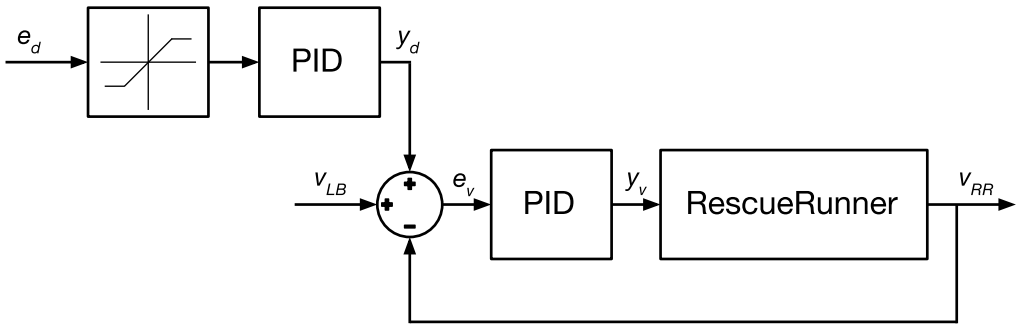


Figure 4.2: Block diagram over the throttle control system.

The output of the throttle PID controller y_v is a number between 0 and 1 signifying no throttle and full throttle respectively. It is calculated as seen in (4.3), where the control error e_v is the difference between the RescueRunner's velocity v_{RR} and the desired velocity which in turn is given by the lead boat's velocity v_{LB} and the compensation for the distance between the two boats y_d.

$$
\begin{aligned}
y_v &= K_p \cdot e_v(t) + K_i \cdot \int_0^t e_v(\tau)d\tau + K_d \cdot \frac{de_v(t)}{dt} \\
e_v &= v_{LB} + y_d - v_{RR}
\end{aligned}
\tag{4.3}
$$

The compensation y_d is calculated by applying a PID controller on the distance between the two boats as seen in (4.4). e_d, as mentioned above, is the error between the actual distance and its reference. To avoid unreasonable compensation at long distances, e_d is limited to a maximum value of 20 m.

$$
\begin{aligned}
y_d &= K_p \cdot e_d(t) + K_i \cdot \int_0^t e_d(\tau)d\tau + K_d \cdot \frac{de_d(t)}{dt} \\
e_d &= \sqrt{d_N^2 + d_W^2} - d_{ref}
\end{aligned}
\tag{4.4}
$$

5

Communication between the two boats

This chapter will detail how the communication between the lead boat and RescueRunner is handled, what hardware is used and how the system responds to communication loss.

5.1 Alternatives for communication

There are several alternatives for establishing wireless communication between the lead boat and the RescueRunner. The solution should be physically small and have a range of at least a few hundred meters to ensure a stable connection. The demands on data throughput are less strict, as the messages being sent between the two are short. The messages are shorter than 100 bytes, and need not be sent more often than at 20 Hz (the estimated required control rate). Thus, a rate of 16 kb/s is estimated to be sufficient.

The nRF24L01+ from Nordic semiconductor is a radio frequency (RF) transceiver chip that operates in the 2.4 GHz ISM band [26]. The chosen implementation of the unit has an additional amplifier that allows it to communicate at a range of up to 1 km with free sight with data rate set to 250 kbps [6]. Communication between units is done in half duplex, meaning that each unit can both transmit and receive, but not at the same time. Data is sent in the form of packets of up to 32 bytes at a time. The chips handle packet buffering and verification of data integrity automatically, and is supported by open source Arduino libraries.[3] Figure 5.1 shows the nRF24L01+.

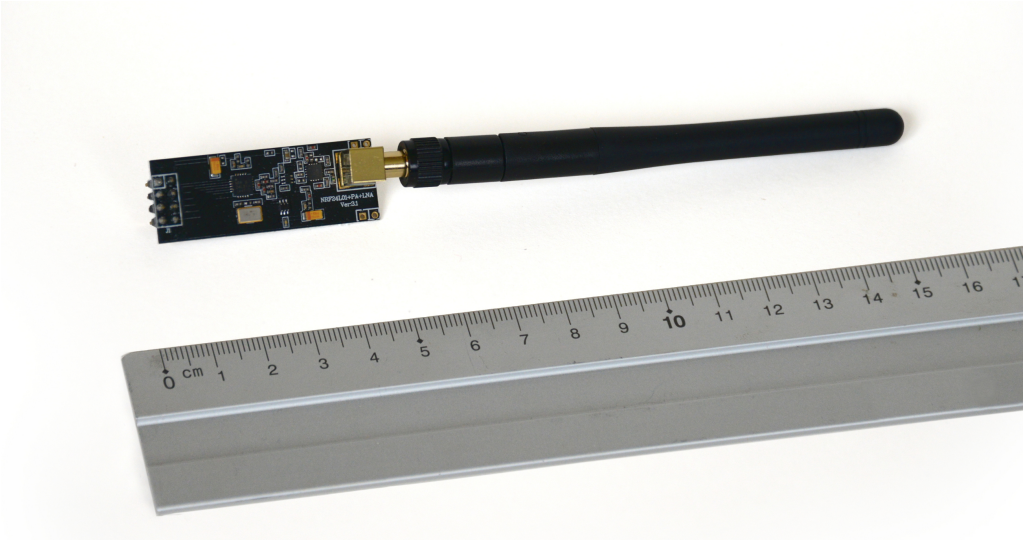


Figure 5.1: The nRF24L01+ transceiver and a ruler for scale.

5.2 Behavior

The radio link between the two boats needs to fulfill multiple tasks. Firstly, it is used to inform the RescueRunner of the lead boat's current position and orientation. It is also used to send performance data used for e.g. debugging. Lastly, the link is used to control in which mode of operation the RescueRunner should be.

The system has several modes implemented. One for manual control, in which it disables all autonomy and lets a driver steer the RescueRunner as normal. Another is the autonomous mode, where the RescueRunner follows the lead boat autonomously by employing the control scheme presented in chapter 4. A third is a remote control mode, where the RescueRunner is controlled from the lead boat through the use of a remote control. The mode is decided by the lead boat, giving the user control over it. However, if either GNSS loses contact with its satellites, or the communication link is broken, the RescueRunner should return to the manual control mode after a short while, to prevent it from steering with incorrect data. When the link is reestablished and the GNSS has reconnected, the mode should once again be decided by the lead boat.

Since the radios operate in only half duplex, the idea is to use the lead boat as primary sender and the RescueRunner as primary listener. The lead boat will continuously send its position, orientation and the mode of operation selected by the user. The lead boat should be able to request information from the RescueRunner, to assist in troubleshooting and enable visualization of the RescueRunner's state and the control signals it is currently applying. Figure 5.2 shows an outline of how the two boats communicate.

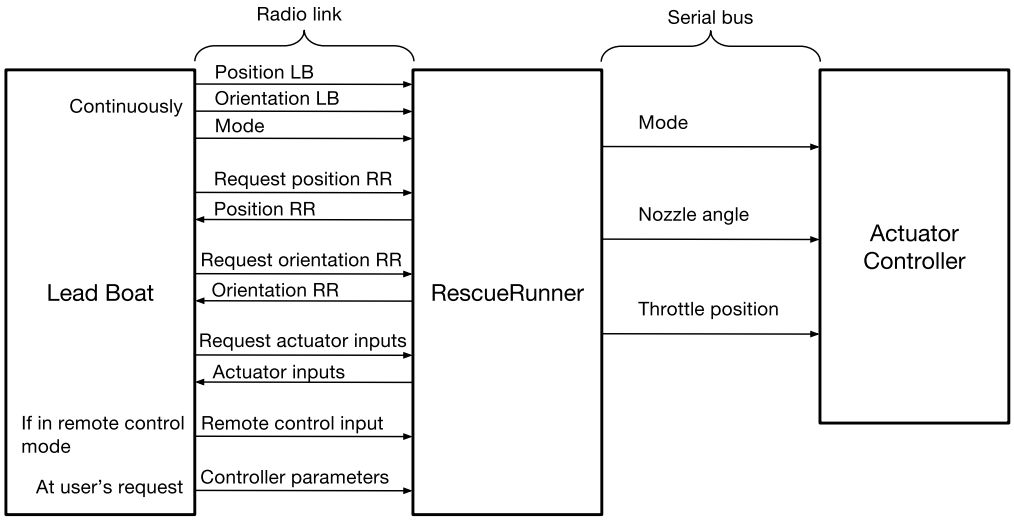


Figure 5.2: An outline of how the two boats communicate. The lead boat continuously send its position, orientation and desired mode to the RescueRunner. The RescueRunner sends data back to the lead boat upon receiving a request. The RescueRunner sends its mode and control inputs to the microcontroller dedicated to controlling the actuators.

6

Implementation on microcontrollers

This chapter will treat the implementation of the algorithms derived in chapters 3, 4 and 5 onto microcontrollers, as well as the necessary supporting code.

6.1 Arduino Due

The Arduino Due was used as the development platform for this work. It is a versatile platform that fulfills this work's demands on computational resources (speed and memory). It also has the benefit of having a large online community, meaning that many guides, assisting functions and other useful libraries are readily available. It is built around the 32-bit Atmel SAM3X8E ARM Cortex-M3 CPU [1]. The Due differs from many other Arduino boards in that it runs at 3.3 V instead of 5 V. This means that there may be compatibility problems with older sensor chips that operate at 5 V. On the other hand, many newer sensors uses 3.3 V. It offers several different hardware communication interfaces such as UART, I^2C and SPI.

6.2 Interfaces

The Arduino Due is required to communicate with several different chips to obtain the required measurement data and achieve communication between the two boats. The Due must make use of several different interfaces for serial communication, as not all chips use the same. Below is a short summary of the three interfaces used, and Figure 6.1 shows the complete system.

6.2.1 UART

Universal asynchronous receiver/transmitter (UART) is a two-wire asynchronous serial communication format. UART allows for multiple different transmission speeds and data formats, thus the two communicating devices must be configured to use the same. The Arduino Due has four sets of full-duplex serial ports. One of these is connected to the USB interface that is used for programming the chip as well as for communication with computers. Incoming data is first placed in a buffer from which it can later be read, giving some flexibility as to when new data is processed provided that this buffer is never allowed to overflow, in which case data will be lost. In this work, UART is used for reading data from the GNSS, communicating with

a computer for visualization and troubleshooting, as well as for issuing commands to the RescueRunner's actuators.

6.2.2 SPI

Serial Peripheral Interface (SPI) is a four-wire, single master, multiple slave, synchronous serial communication interface. The four wires are labeled: Serial Clock (SCLK), Master Out Slave In (MOSI), Master In Slave Out (MISO) and Slave Select (SS). Each slave has its own SS wire and the master selects which slave should listen to the next command by pulling the corresponding SS wire low. There are a number of different configurations for SPI called modes, that governs e.g. clock polarity. The mode must match between the master and slave. If multiple slaves which specify different modes are used, the mode of the master must be changed as the different slaves are selected. SPI is used in this work to communicate with the nRF24L01+ transceiver chip.

6.2.3 I^2C

Inter-Integrated Circuit (I^2C) is a multiple master, multiple slave serial communication interface. Data is communicated over a single wire at one direction at a time, with specific bits for signaling start, stop and acknowledgement. Instead of a dedicated wire for each slave as with SPI, each data transaction starts with a 7 bit long address call specifying a slave. This means that I^2C only uses two wires: Serial Data Line (SDA) and Serial Clock Line (SCL). To avoid problems with conflicting addresses, some chips allow for setting an alternative address. A feature found in many I^2C implementations is an auto incrementation of the address pointer. This allows allows multiple registers to be written to or read from in one I^2C call. In this work, I^2C is used to communicate with the gyroscope, accelerometer and magnetometer.

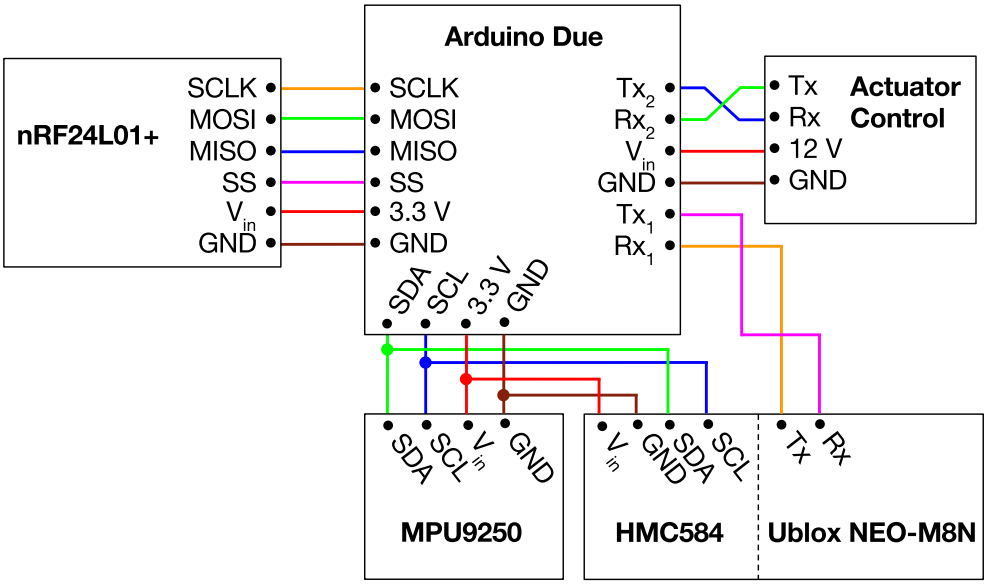


Figure 6.1: This figure shows a circuit diagram for the hardware mounted on board the RescueRunner. The setup on the lead boat is similar but will of course not include the actuator control. Not included in this figure is the level shifting circuit between the Arduino Due and actuator control that is needed since they use different signal levels, 3.3 V and 5 V respectively.

6.3 The implemented code

Previously, the components that will make up the complete system has only been discussed separately. This section will attempt to explain how they were implemented on microcontrollers to work in unison.

6.3.1 Differences between MATLAB script and C++

The code for the Arduino Due microcontroller is written in C++ and there are several differences between coding in MATLAB script and coding in C++. For example, MATLAB treats each variable as an array, and matrix operations can be expressed very intuitively. This stands in contrast to C++, where arrays and matrices are constructed using pointers (and pointers to pointers respectively) and there is no inherent support for matrix operations, which have to be built manually. Another difference is the handling of data types. By default, MATLAB uses 64-bit double precision floating-point numbers for most variables [22], while the data types used in C++ must be specified by the user. This difference stems from the available hardware, as MATLAB is designed to run on a personal computer while C++ should also be able to operate on smaller and less powerful devices such as microcontrollers.

6.3.2 Multitasking

Since the Due on each boat has to execute multiple tasks concurrently a simple scheme to divide the processing time was devised. The task that need to be performed most frequently is the orientation filter's prediction and update steps which runs at 200 Hz. The main loop was therefore set to run at this frequency. In the beginning of the loop, a timer is started and at the end there is a section of code that measures the elapsed time. This is then subtracted from a 5000 μs delay which is applied to ensure a stable rate of 200 Hz. With the use of counters, this 200 Hz base rate can then be divided to provide other rates. An example would be a counter which ensures a specific task is only executed once every tenth loop iteration, resulting in a rate of 20 Hz instead. To distribute the workload more evenly, tasks that share the same rate can be made to run out of phase, ensuring that only one of them are performed in a single iteration of the main loop. This reduces the worst case execution time of a single iteration of the main loop.

6.3.3 Communication with MATLAB

One of the first implemented functions was a system for communication between the microcontrollers and MATLAB. Using the UART that is connected to the USB, simple messages are sent with unique characters marking the start followed by a predetermined number of data bits. The communication was initially used to forward the gyroscope, accelerometer and magnetometer measurements to test the orientation filter in MATLAB as explained in chapter 3. Later, the link was used to send the orientation as estimated by the microcontroller for visualization. When testing began on the full system, messages included GNSS data (position and velocity), orientation, control signals sent to the RescueRunner's actuators and a measure of whether or not the radio communication between the two boats was functioning. The link was also used during testing when setting the different modes of the system as explained in section 5.2, and a gamepad connected to the computer provided remote control inputs. A final use for the communication link between MATLAB and the microcontrollers was to enable tuning of the controller parameters without having to reprogram the microcontroller. The microcontroller on the lead boat forwarded these messages over the radio link so that no physical connection was needed between the RescueRunner and the computer.

6.3.4 Acquiring sensor data

Before an estimate can be formed, the sensor data upon which it is based must be acquired. Both the gyroscope, accelerometer and magnetometer are accessed over I^2C using the *Wire* library [2] for the Arduino. At every start up, a setup routine is executed to configure the sensors, e.g. setting their measurement ranges and sample rates. The sensors are then sampled at the start of the main loop running at a rate of 200 Hz. The samples are sent as 16-bit signed integers that are transformed into their respective units using a scaling factor given by their set measurement range.

Unlike the sensors above, the GNSS data is not sent upon a request. Instead it automatically sends its data, in the form of a 98 bytes long message, over one of the UART ports at a rate of 10 Hz. The data is collected in a 64-bytes long buffer associated with this port. Thus, a full message cannot be contained within the buffer without extending it. This was solved by checking the buffer often and reading parts of the message at a time. At each iteration of the main loop, the buffer is checked to see if it contains any data. If so, the program starts searching for the byte sequence signaling the start of a message. The program then spends up to 100 μs reading from the buffer and saving its message to a temporary storage. This maximum duration was introduced as it limits the time the sensor reading can block other tasks, while still ensuring the important property that the buffer is read faster than the GNSS can write to it (avoiding buffer overflow). If the time limit is reached or the buffer is emptied before the entire message has been read, the program continues reading in the next iteration of the main loop. Once a full 98 bytes message has been read, the data is copied from its temporary storage to the variables used by the control system.

6.3.5 Filter code

Once the sampling of the sensors was working, the implementation of the filter algorithms derived in chapter 3 could start. As mentioned there, the orientation filter was already implemented in MATLAB and implementing it on the Arduino Due was a matter of translating the MATLAB script into C++ code. As the extended Kalman filter is the most computationally demanding part of this work, the translation should have computational efficiency in mind to avoid overloading the Due.

The Due unfortunately lacks dedicated hardware for floating point operations, which means that these has to be built up by multiple instructions (which is done automatically by the compiler). The result is that floating point operations will take a comparably long time to execute and initial tests indicated that it would be difficult to keep within the time limits required by the 200 Hz rate. Apart from obtaining a more powerful microcontroller, solutions to this would be either lowering the rate at which the observer operates, or converting the operations to use binary fixed point (utilizing integer operations). Since the EKF approximates a nonlinear system by repeated linearizations around the current state, a lower sample rate would mean a courser approximation and possibly worse performance. Converting the code to use integers reduces the execution time without affecting the filter's performance. However, it decreases the readability of the code and makes debugging more difficult. Nevertheless, it was seen as the more attractive option and was thus implemented.

Adapting the code to using integers was achieved by scaling up the variables such that their values could be handled by integer operations without a loss of precision due to rounding off. The variables were scaled up by a power of two (2^N) which is equivalent to shifting its bits left N times. This is a computationally cheap operation. All variables are not scaled with the same factor, as some variables hold too high or too low values. A too high value would result in an overflow, while a

too low value would result in a loss of precision. The scales are thus customized to fit the values held by each variable. A simple example of how a multiplication operation could be handled with integers is shown in (6.1).

$$0.5 \cdot 0.5 = 50/100 \cdot 50/100 = 2500/10000 = 25/100 = 0.25 \tag{6.1}$$

6.3.5.1 Implementing the compensation for magnetic declination

The heading compensation algorithm described in section 3.6.4 and presented in equation (3.39) was implemented as part of the orientation filter. The factor α that regulates the speed at which the correction angle change was chosen to be $\alpha = 0.0001$. This assures that the correction angle changes slowly so as not to corrupt the heading estimate when the RescueRunner experiences short periods of swaying motions.

If the speed of the RescueRunner is low, then the GNSS receiver cannot form an accurate heading estimate. Because of this, the GNSS heading estimate is only used when the RescueRunner's speed is greater than 5 m/s (18 km/h) and has been for at least ten seconds. When the RescueRunner first reaches the speed threshold, a timer is started. If the timer reaches ten seconds, the correction process is started. If the RescueRunner slows down below the threshold, the correction is stopped and the timer reset.

6.3.6 Radio

As mentioned in chapter 5, the nRF24L01+ radio was used for communication between the two boats, with the lead boat as primary sender and the RescueRunner as primary listener. The radio was accessed through the SPI bus, made simpler by the RF24 [31] library.

The Enhanced ShockBurst™ protocol that the chips use is based on data being sent as packets. It automatically handles most of the tasks associated with using packets, and offers several assisting functionalities. For example, it offers an auto-acknowledgement feature, in which the chip automatically sends a response upon receiving a packet to verify a successful transmission. This response message can be customized to contain a short message of the user's choosing. The chip can also be configured to retransmit the last packet a number of times if no acknowledgement was returned. In this implementation, the number of retries was set to three, with a delay of 1500 μs between each one. The delay was chosen because it is the minimum time required to send a full 32 byte packet using the Enhanced ShockBurst™ protocol at 250 kbps. The received messages are checked every 5000 μs and therefore there is only time for three retries.

The communication between the boats was implemented through the use of multiple message types. To simplify the handling of the messages, only one is sent per packet. The first byte of a packet indicates what type of message it holds, so that the data within it is interpreted correctly. Internally the radio uses two FIFO buffers, one for

received packets and one for those queued to be sent. These can hold a maximum of three packets each and the oldest packet is overwritten should a fourth packet arrive. To avoid this scenario and to simplify implementation, the transmissions were fixed to one per 200 Hz loop. Similarly to how the multitasking was handled in section 6.3.2, the transmission was divided into ten 20 Hz slots (some of which were divided further into four 5 Hz slots). Each process using the radio could then use one of these slots. This made the transmissions very predictable and the processes will not interfere with each other.

To acquire data from the RescueRunner, the lead boat must actively request it. When a data request message arrives, the RescueRunner loads the requested data into the auto acknowledgement register. The next time the RescueRunner receives a message (of any type), the auto acknowledgement will be sent to the lead boat. Note that no data sent from the RescueRunner to the lead boat is of critical importance, as they are used solely for visualization and debugging, making this delay acceptable.

Timers keep track of how much time has passed since the last message of each type arrived and prevents the system from using old data. Each message sent from the lead boat should be answered by an acknowledgement from the RescueRunner. If no acknowledgement has arrived within 5 ms the transmission is assumed to have failed. To give the user an indication of the communication status, a counter increments with every failed transmission and decrements with every successful one. As long as the counter stays at (or close to) zero the connection is considered good.

6.3.7 Controller

With the state observer and the radio link done, all the data necessary to implement the control scheme derived in chapter 4 is available. However, some modifications to the control scheme formulation is necessary to make it suitable for implementation in C++ code.

The transfer function of a PID controller is shown in (6.2). In practice, however, it is necessary to low-pass filter the derivative since it is sensitive to high frequency noise. The transfer function of the PID controller with a filtered derivative is shown in (6.3), where $T_f = 1/\omega_c$ and ω_c is the cutoff frequency of the low-pass filter.

$$\text{PID(s)} = K_p + \frac{K_i}{s} + K_d \cdot s \tag{6.2}$$

$$\text{PID(s)} = K_p + \frac{K_i}{s} + K_d \cdot \frac{s}{1 + T_f s} \tag{6.3}$$

Since the microcontroller operates in discrete time, the PID controllers are discretized. The derivation of a discrete time PID controller using the Tustin method is shown in appendix B. The discrete time formulation is shown in (6.4), where h is the sampling time of the discretization, which is 0.05 s since the control rate is 20 Hz. Once the parameters b_0, b_1, b_2, a_0, a_1, a_2 are initialized, the control output

$y(n)$ can be calculated using the current and previous control errors $e(n)$, $e(n-1)$, $e(n-2)$ and the previous control outputs $y(n-1)$, $y(n-2)$.

$$\begin{aligned}
y(n) =& \frac{e(n)b_0 + e(n-1)b_1 + e(n-2)b_2 - y(n-1)a_1 - y(n-2)a_2}{a_0} \\
b_0 =& (2T_f + h)(2K_p + hK_i) + 4K_d \\
b_1 =& -8K_d - 2K_p(2T_f + h) + hK_i(2T_f + h) \\
& - 2T_f(2K_p + hK_i) + h(2K_p + hK_i) \\
b_2 =& 4T_f K_p - 2hT_f K_i - 2hK_p + h^2 K_i + 4K_d \\
a_0 =& 4T_f + 2h \\
a_1 =& -8T_f \\
a_2 =& 4T_f - 2h
\end{aligned} \tag{6.4}$$

The three PID controllers derived in chapter 4 can all be implemented in C++ code using the formulation in (6.4). To facilitate further tuning of the PID controllers during tests in water, the code was designed so that the values of the parameters K_p, K_i, K_d, T_f could be updated remotely for each controller using the radio link.

7

Tests and results

A number of tests were conducted to evaluate the performance of the system. First, the quality of the state observer's estimates were evaluated. Then, tests were performed on the radio transceivers to determine their maximum range, and finally the system in its entirety was tested in practice by using the estimated states in the control algorithm derived in chapter 4.

7.1 Evaluating the state observer

Tests were performed to evaluate the quality of the state estimates provided by the observer. The tests focused on the qualities most relevant for the control system, such as the correctness of the relative position estimates of the two boats and relative bearing.

7.1.1 Testing the GNSS

A test of the GNSS receivers was conducted to evaluate their ability to determine their positions relative to each other. The two test boards were placed outside in protective plastic covers at a distance of 15.4 m apart. The radio link was used to transmit the data collected by one board to the other which in turn was connected to a computer where data from both boards was saved. Figure 7.1 shows a histogram of the error in estimated horizontal position. 95 % of the measurements had errors of less than 4.5 m.

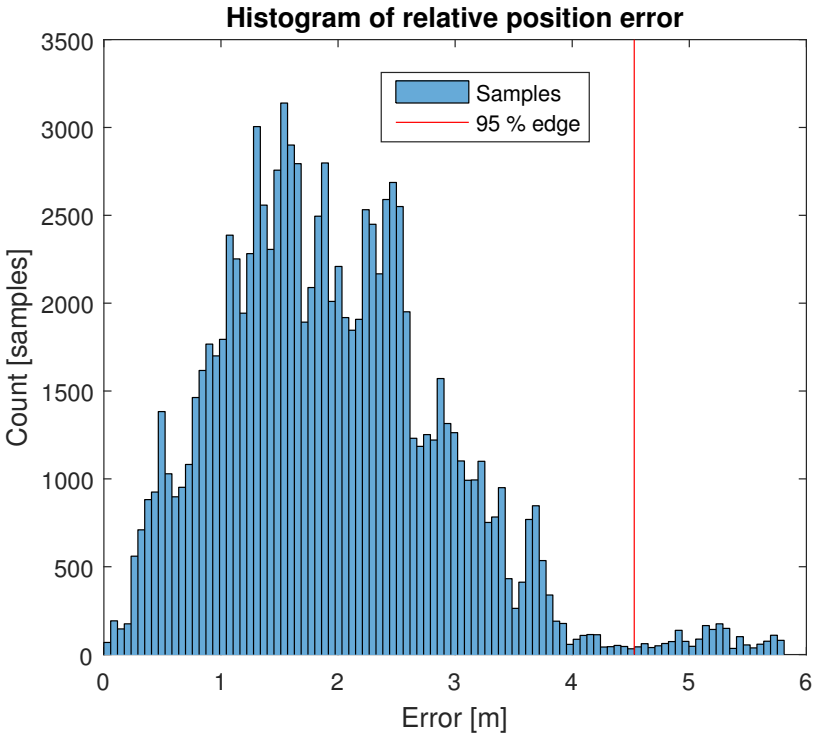
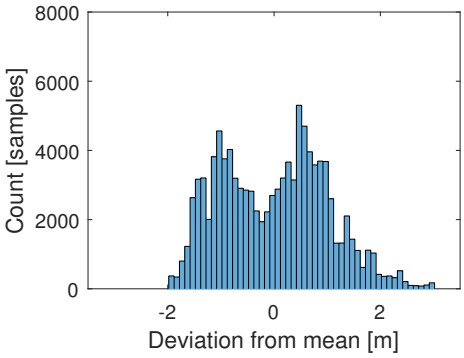
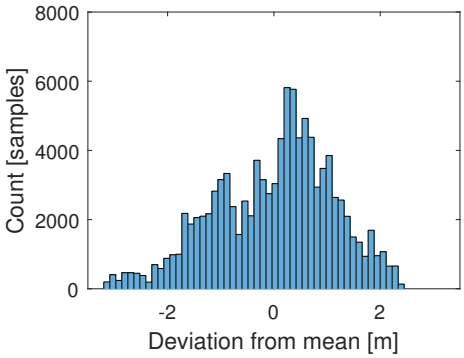


Figure 7.1: A histogram of the magnitude of the relative horizontal position error between the two GNSS receivers, positioned 15.4 m apart. 95 % of all samples are located to the left of the red vertical line at 4.5 m.

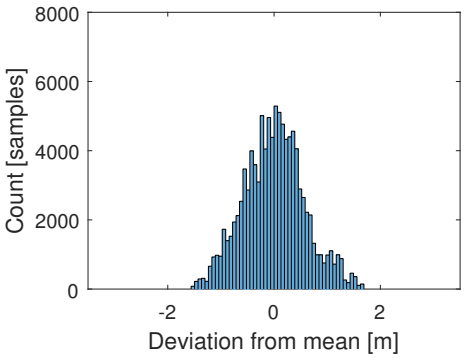
During most of the test, the weather was rainy. This should not affect the performance of the GNSS receivers, unless a pool of water forms above the antenna [30]. However, the computer had to be placed under the cover of a roof and the board connected to the computer was located approximately 7 m from a building, giving it a less clear view of the sky. This can affect the receiver's performance, and is noticeable as its measurements had a higher variance than the other receiver, as shown in Figure 7.2.



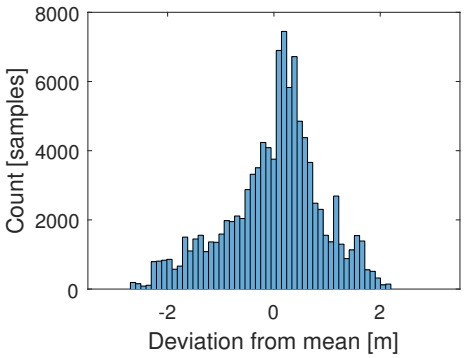
(a) Longitudinal measurements for the receiver with slightly obstructed view of the sky.



(b) Latitudinal measurements for the receiver with slightly obstructed view of the sky.



(c) Longitudinal measurements for the receiver with clear view of the sky.



(d) Latitudinal measurements for the receiver with clear view of the sky.

Figure 7.2: Histograms over the deviations from the means of the latitudinal and longitudinal measurements for the two GNSS receivers. The measurements of the receiver with a clear view of the sky has noticeably less variance than the receiver with a slightly obstructed view of the sky.

7.1.2 Testing the orientation

The observer continuously estimates the orientation of the boats in a NWU frame. From this estimate, the current heading is extracted and then used by the control system to steer the RescueRunner. A test was conducted to determine if the state observer's heading estimate matches that of the GNSS i.e. if the northward direction of the magnetometer matches the northward direction of the GNSS receivers.

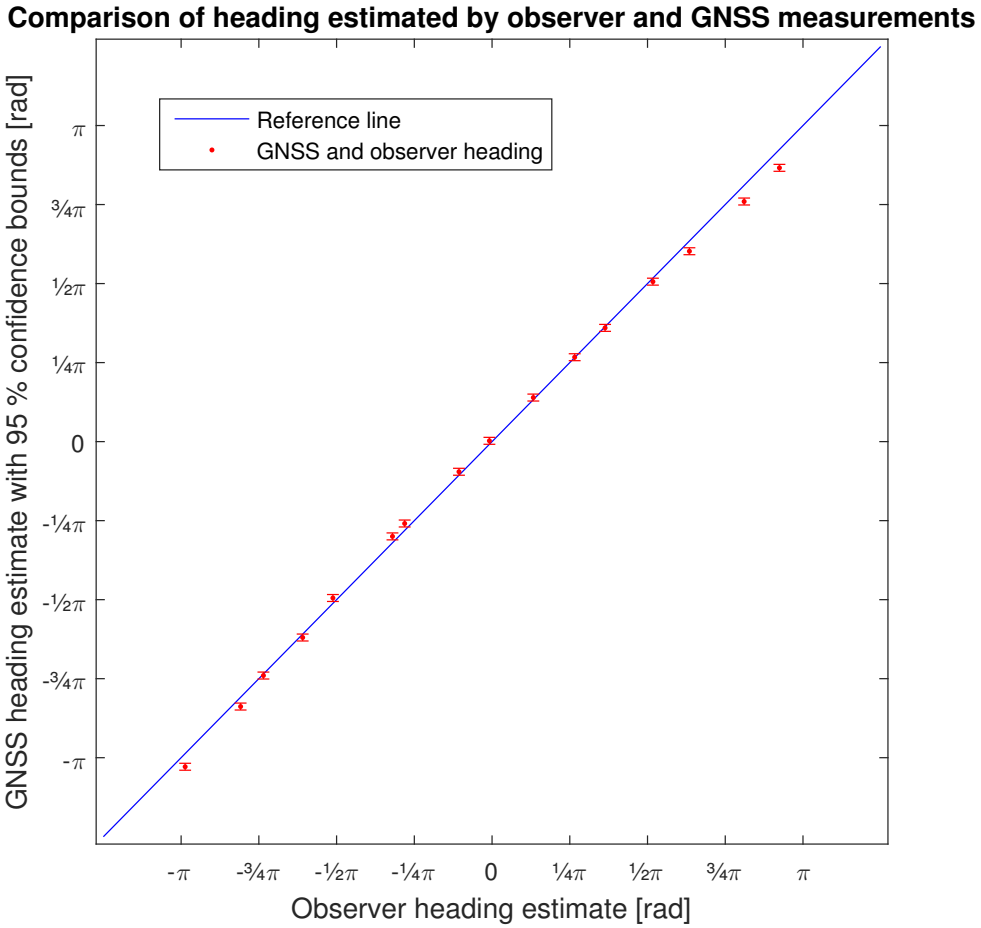


Figure 7.3: A plot of the state observer's heading estimate compared to that of the GNSS. The blue line represents the expected values if the two estimates match perfectly. A 95 % confidence bound for the angle calculated from the GNSS measurements is included. It can be seen that some points fall outside this bound, hinting at an error in the state observer's heading estimate.

The test boards are first separated from each other a distance of about 35 m. Next, one of the test boards is aimed towards the other by hand. The boards' coordinates along with the state observer's heading estimates are sampled for 5 s and their means are saved. Next, the angle of the straight line between the two coordinates (as given by the GNSS) is compared to the heading angle estimated by the state observer. This is done for 16 points distributed around a complete circle. The result is shown in Figure 7.3. The blue line indicates where the points would be if the heading of the GNSS and the state observer match perfectly. As the figure shows, this is not the case. Included in the figure is a 95 % confidence bound for the angle calculated from the GNSS measurements. This is derived from the data collected for section 7.1.1 by approximating the spread of the estimated angle by assuming an identical GNSS position spread, but at a distance of 35 m. In addition, it is assumed that the mean angular error from this data set (collected over longer time) is close to zero. As the figure shows, some of the data points deviate from the blue line beyond this

bound and in these cases, the error is likely to lie in the observer's heading estimate or the test board not facing the other one perfectly. The mean magnitude of the errors among the 16 points is 0.0653 radians (or 3.74°) with the largest being 0.1825 radians (or 10.46°). The difference between the observer's heading estimates and the GNSS based ones has a mean bias of 0.0301 radians (or 1.72°), meaning that the observer tends to have a counter-clockwise horizontal rotational bias. The variance of this difference is 0.00653 radians (or 0.347 °). It can also be noted that the errors seems to follow a curvature that tends to be above the blue line around 0 radians and below around $\pm\pi$ radians. It is not unlikely that this is due to a calibration error resulting in a systematic bias that is rotated with the sensor.

7.1.2.1 Testing the compensation for magnetic declination

The heading compensation algorithm described in sections 3.6.4 and 6.3.5.1 was tested to verify a correct behavior. One of the test boards was intentionally held at an incorrect angle of about 45°. The speed threshold for activating the correction was temporarily lowered so that it could be tested by jogging in a straight line while keeping the heading of the board fixed. During the jog, the value of the correction angle ψ_{corr} was monitored. The rate at which it changed was deemed satisfactory, with it correcting about a third of the error in about 20 seconds. As the error $(\psi_{GNSS} - \psi)$ decreases, so does the correction angle's rate of change.

7.2 Radio range

A test of the radio transceiver units was conducted in an attempt to determine the maximum range of communication. The test was done at sea by continuously increasing the distance between the two transceivers by driving the RescueRunner away from the lead boat while monitoring the signal quality. The quality was evaluated using the lost packet counter described in section 6.3.6 which increments with every lost packet and decrements with every successfully acknowledged one. The signal was deemed acceptable if the counter remained at or around zero, indicating that at least half of all messages are successfully received. The two boats had a free line of sight between them. The result of the test is shown in Figure 7.4. Both the distance between the boats and the connection status is shown. It can be seen that the signal quality is reduced around a distance of 550 m. Since the distance between the two boats is communicated using the same radio link, the distance curve becomes noticeably coarser beyond this point. As the RescueRunner should attempt to follow the lead boat closely, this performance in radio range is deemed more than sufficient.

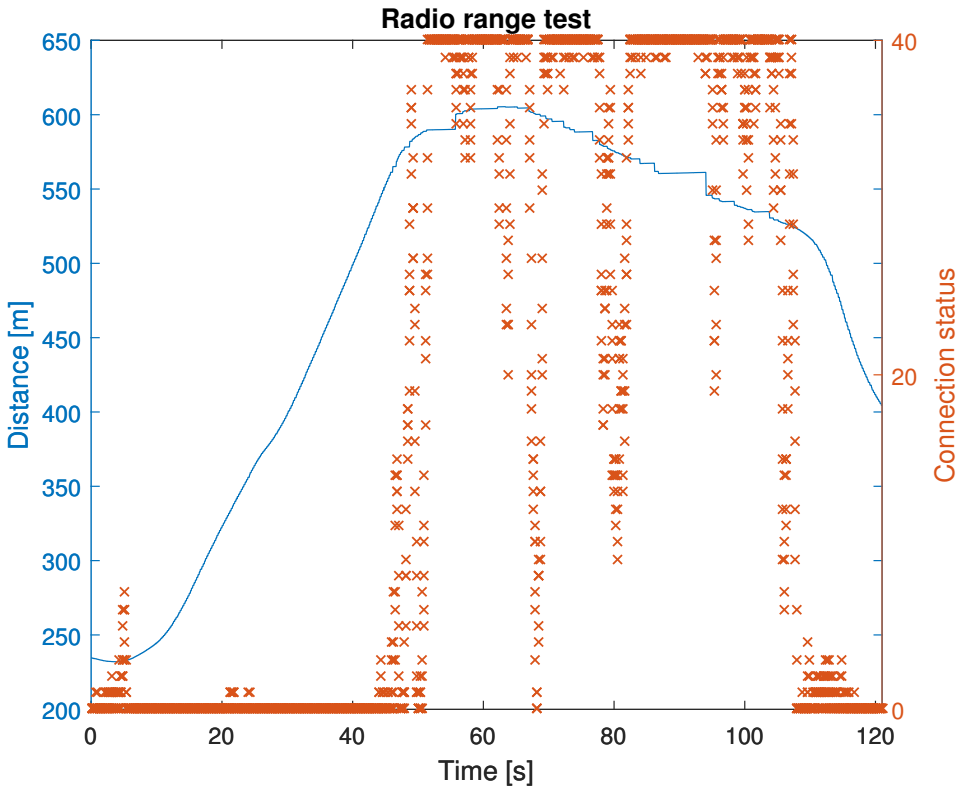


Figure 7.4: A plot showing the distance between the boats and the connection status over time. The connection status is measured as described in section 6.3.6, where the connection status counter is incremented for each failed radio message, and decremented for each successful one, while being limited between 0 and 40. The signal quality is reduced for distances above 550 m, which also can be seen in that the blue line becomes noticeably coarser.

7.3 Tests with the RescueRunner

As a final evaluation, the complete system was assembled for test runs of the autonomous mode. These tests should reveal how well the individual components of the system function together as well as test them in the environment they should operate in. As there at this time was no RescueRunner available, the tests were instead carried out on a prototype boat similar in size and powertrain. The boat that was used can be seen in Figure 7.5. Behind the driver's seat is a box holding the Arduino due, sensors and radio transceiver. An extensions cable connects the box to the antenna mounted in the superstructure above. Another cable connects it to the box mounted below the driver's seat. This is the system developed in [24] which converts the controller outputs to physical movements. For the sake of simplicity, the prototype boat will henceforth be referred to as the RescueRunner. The lead boat used in the tests was one of SSRS Postkodlotteriet-class boats, seen in Figure 7.6. The equipment for the lead boat was mounted on the aft part of the wheelhouse roof for a good reception for the GNSS receiver and free line of sight between the radios. This is the reference point for all angles and distances presented in this section.

Figure 7.5: The prototype boat that was used instead of a RescueRunner. It is designed to carry more people than the RescueRunner and is therefore slightly larger (4 m LOA as opposed to 3.6 m). Behind the driver's seat is a box holding the Arduino Due, sensors and radio transceiver. An extensions cable connects the box to the antenna mounted in the superstructure above. Another cable connects it to the box mounted below the driver's seat. This is the system developed in [24] which converts the controller outputs to physical movements.

Figure 7.6: A boat of the Postkodlotteriet-class that was used as lead boat during the tests.

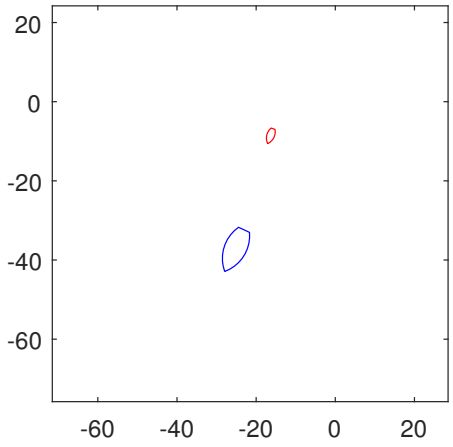
A first test was to check if the system was working and behaved as expected. This was done by driving the RescueRunner about 300 m away from the lead boat and then switching over to the autonomous mode to let it steer its way back. As the lead boat laid still, the only component contributing to the throttle input was the distance regulating PID (in these test only K_p was non zero, in effect a P-controller). This should result in an initial speed of 3 m/s which should be reduced to idle speed as the RescueRunner comes near the lead boat. The RescueRunner accelerated and aligned itself on a course towards the lead boat as can be seen in Figure 7.7. When the RescueRunner was around 30 m away it started to slowly sway back and forth, in the end making almost 180° turns. This was the expected result as the controller assumes that the effect of the nozzle angle is independent of the throttle level. Consequently, it only works well around the throttle level it is currently tuned to. Overall the result was deemed satisfactory and the testing could proceed.

Figure 7.7: The RescueRunner as it drives autonomously towards the lead boat.

7.3.1 First test of following the lead boat

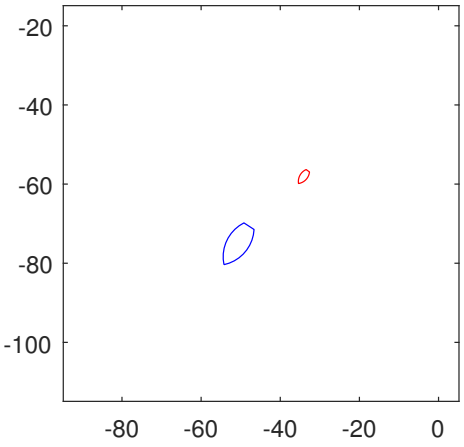
The lead boat accelerated to a speed of 8.2 m/s (16 knots) while holding a constant course. The RescueRunner's autonomous mode was activated just before the lead boat's acceleration began, and the results were filmed and the sensor data logged. Around 75 seconds into the test, the lead boat makes a sharp starboard turn. Around the 125 seconds mark, the lead boat makes another less sharp starboard turn. The PID parameters used for this test were $K_p = 1.2$, $K_i = 0$, $K_d = 1.5$, $T_f = 0.26$ for the angle controller, $K_p = 0.15$, $K_i = 0.5$, $K_d = 0$ for the velocity controller and $K_p = 0.15$, $K_i = 0$, $K_d = 0$ with $d_{ref} = 20$ m for the distance compensation. The correction of the magnetometer's heading estimation described in section 3.6.4 was running during the test. In Figure 7.8, the RescueRunner can be seen as it follows behind the lead boat, both from the perspective of a camera on board the lead boat, and through a plot using the sensor data. A plot of the positions of the lead boat and the RescueRunner during the test is shown in Figure 7.9.

(a) 7 seconds into the test. The RescueRunner approaches the stationary lead boat at low speed.



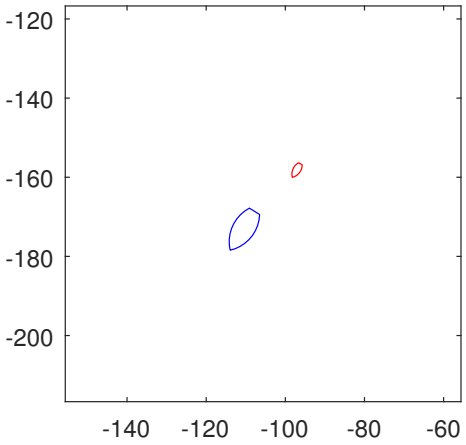
(b) Estimated position of the two boats after 7 seconds.

(c) 21 seconds into the test. The RescueRunner follows behind, decreasing its distance to the lead boat.



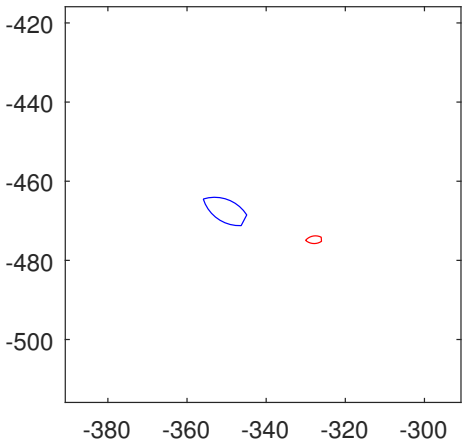
(d) Estimated position of the two boats after 21 seconds.

(e) 36 seconds into the test. The RescueRunner has reached its steady state distance behind the lead boat. The RescueRunner stays at this position until the lead boat makes a turn.



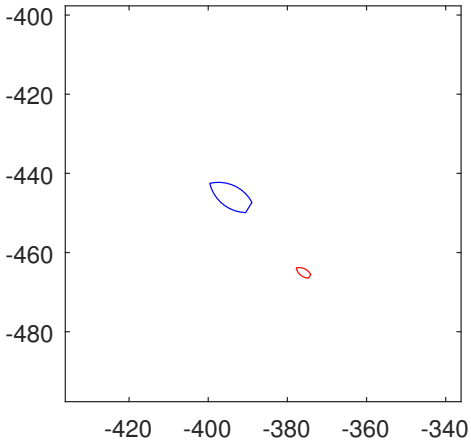
(f) Estimated position of the two boats after 36 seconds.

(g) 81 seconds into the test. As the lead boat turns, the RescueRunner goes over the wave formed by the lead boat's wake.



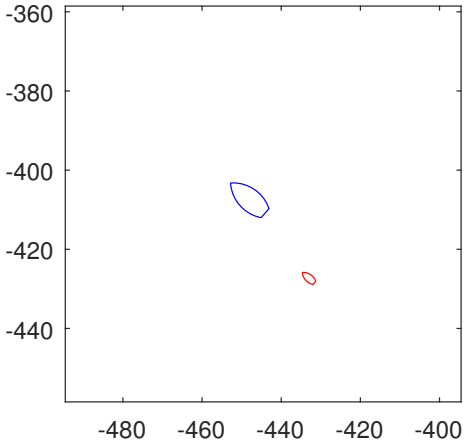
(h) Estimated position of the two boats after 81 seconds.

(i) 87 seconds into the test. The RescueRunner starts to fall behind as it struggles to make it over the wave formed by the lead boat's wake.



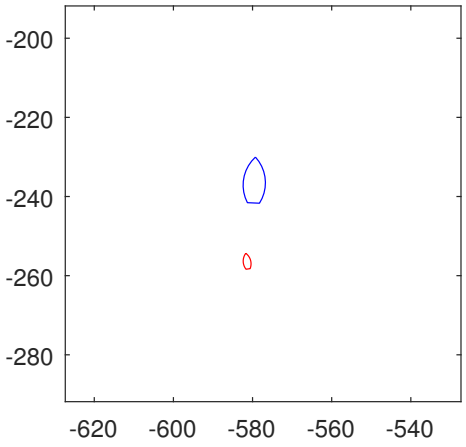
(j) Estimated position of the two boats after 87 seconds.

(k) 95 seconds into the test. At the third attempt the RescueRunner clears the wave and subsequently stabilizes behind the lead boat as before.



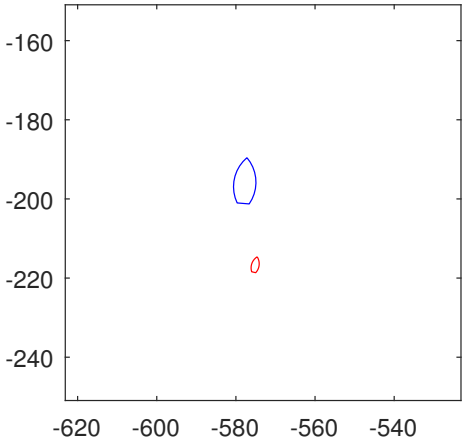
(l) Estimated position of the two boats after 95 seconds.

(m) 128 seconds into the test. After successfully completing another turn the RescueRunner makes an unexpectedly hard starboard correction. 7.8n shows that at this time the estimated orientation does not seem to match the actual one.



(n) Estimated position of the two boats after 128 seconds. The RescueRunner estimates its angle relative to the lead boat to be close to zero, which is clearly not the case as shown in 7.8m.

(o) 132 seconds into the test. The RescueRunner experiences oscillations back and forth from which the system does not recover. The test is stopped for further tuning soon afterwards.



(p) Estimated position of the two boats after 132 seconds. The heading estimations are clearly incorrect.

Figure 7.8: Images from the first test performed at Långedrag August 23rd 2016 along with plots created from the estimated heading and positions of the boats. The test shows that the autonomous system is able to steer the RescueRunner behind the lead boat, but also reveals that the system can be set into a state of oscillating back and forth behind the lead boat.

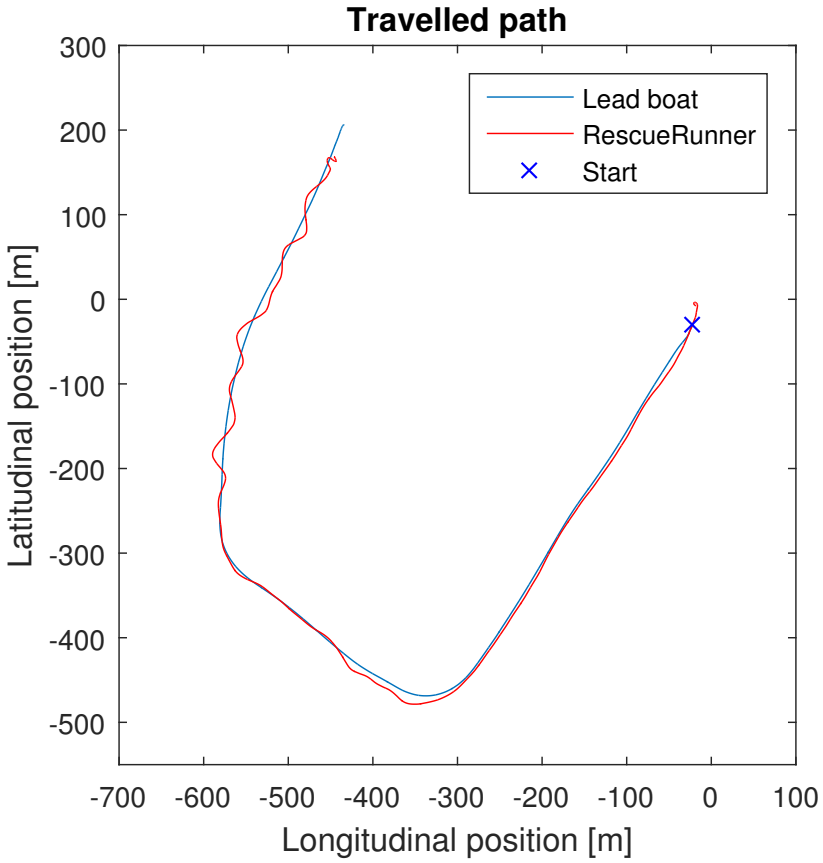
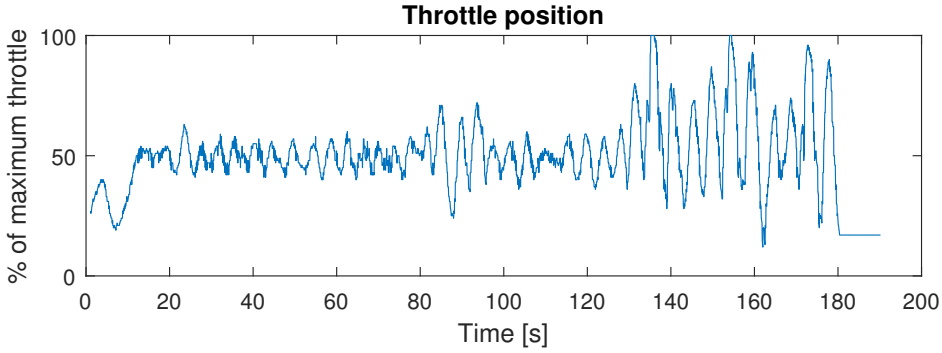
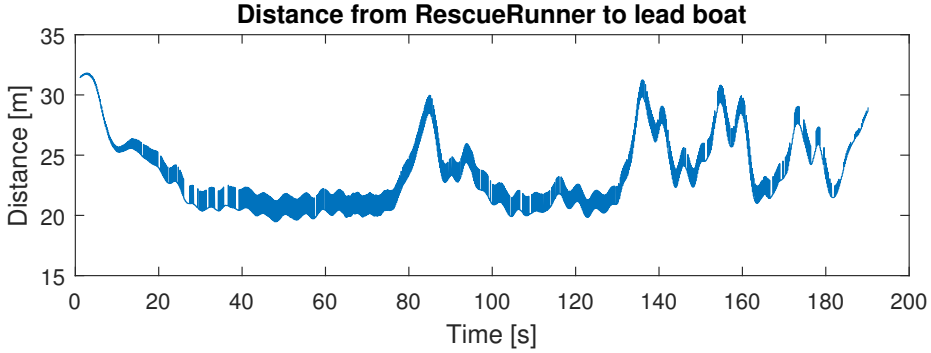


Figure 7.9: A plot of the positions of the lead boat and the RescueRunner sampled during the first test.

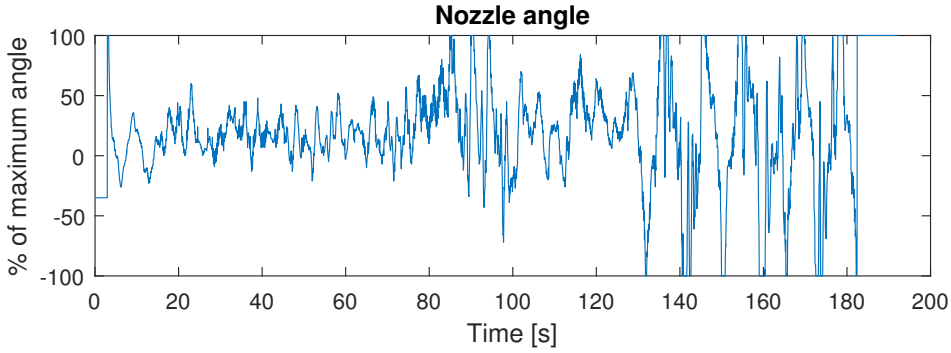


(a) The throttle input as requested by the controller sampled from the first test.

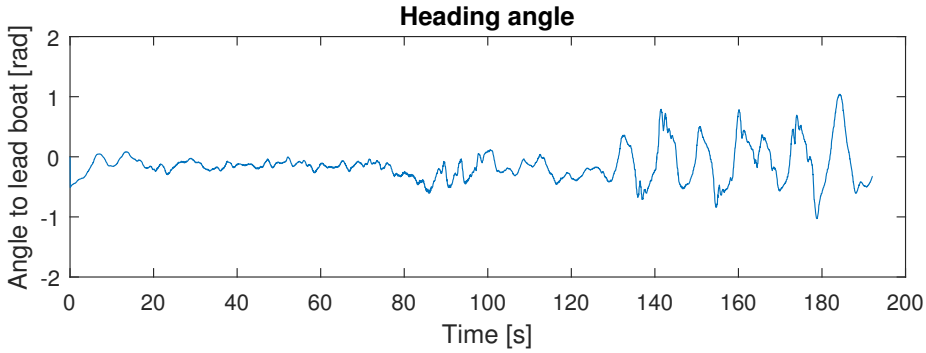


(b) The estimated distance between the two boats sampled from the first test. The high-frequency noise is due to the two GNSS receivers' positions not updating simultaneously.

Figure 7.10: A comparison between the estimated distance between the two boats and the throttle input as requested by the controller, sampled from the first test. The RescueRunner clearly succeeds at keeping a distance of approximately 20 m during the stretches where the lead boat is driving straight, with a throttle input of around 50 %. The curve made around the 75 seconds mark causes the RescueRunner to temporarily fall behind, resulting in more fluctuating throttle inputs. The same behavior is seen towards the end of the test, where the RescueRunner is experiencing oscillations back and forth. The test is ended around the 180 seconds mark, as seen by the input throttle no longer being updated. The oscillations experienced towards the end of the test do not seem to grow, hinting at a marginally stable characteristic. Nevertheless, it is a highly unwanted behavior.



(a) The nozzle angle input as requested by the controller sampled from the first test.



(b) The angle to the lead boat from the RescueRunner's perspective sampled from the first test. The angle is somewhat steadily around zero until the end of the test where the RescueRunner experiences its oscillatory behavior.

Figure 7.11: A comparison between the estimated heading angle of the RescueRunner relative to the lead boat and the nozzle angle input as requested by the controller, sampled from the first test. The RescueRunner appears to succeed keeping the angle close to zero during the stretches where the lead boat is driving straight, with a nozzle angle input of about ±15 % of the nozzle angle's range. This would be equivalent to turning the handlebar of the RescueRunner approximately ±17°. The relatively high derivative part of the nozzle angle PID controller makes the control system faster in reacting to errors and reduces overshoot, but amplifies the high frequency components of the heading angle estimate. Comparing these figures to the images in 7.8 it is easy to identify some key moments such as where the RescueRunner goes outside the wake at 81 seconds, or where it starts to oscillate at around 128 seconds.

Plots of the throttle input and the estimated distance between the two boats are shown in Figure 7.10, which shows that the throttle controller mostly succeeds at keeping the RescueRunner at a distance of around 20 m from the lead boat. It also shows that the RescueRunner tend to fall behind when the lead boat makes a turn, and when the RescueRunner experiences its oscillating behavior towards the end of the test.

Plots of the nozzle angle input and the estimated heading angle of the RescueRunner relative to the lead boat are shown in Figure 7.11, which shows that the angle controller succeeds at keeping the heading around zero until it experiences its oscillating behavior. The fact that this behavior is noticeable in the estimated heading angle plot means that the system notices it happening, but fails at stabilizing it.

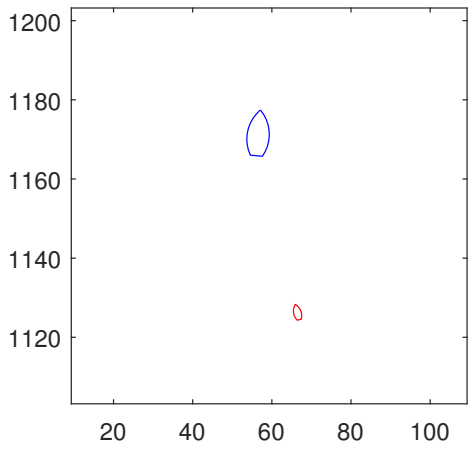
A second test is deemed necessary to find parameters which results in a more stable system that suppresses the oscillations. The idea is to first make the system experience oscillations, then switch to a set of parameters that successfully eliminates them.

7.3.2 Second test of following the lead boat

The lead boat once again accelerated to a speed of 8.2 m/s (16 knots) and the RescueRunner's autonomous mode was activated. Whenever the RescueRunner started experiencing oscillations like in the first test, the PID parameters of the nozzle angle controller were changed to find a set of parameters that results in a more stable behavior that eliminates the oscillations while still achieving an acceptable performance in following the lead boat. The PID parameters of the nozzle angle controller were changed to $K_p = 1.0$, $K_i = 0$, $K_d = 1.2$, $T_f = 0.26$, resulting in a slightly less aggressive system that recovered from the oscillations brought on by the previous set of parameters. The distance compensation was changed so that the reference distance d_{ref} is 45 m, giving the system more margin before falling outside the waves in the lead boat's wake. Apart from this, the throttle control system's parameters remained the same, as its previous behavior was deemed acceptable. In addition, the correction of the magnetometer's heading estimation described in section 3.6.4 was deactivated for the duration of the test, to assess the system's performance without it.

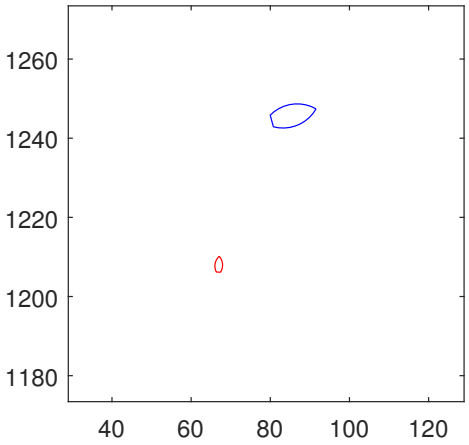
Once the stable set of parameters had been found, the lead boat made several turns to test the system's stability and performance. In Figure 7.12, the pictures and plots of the RescueRunner are shown. The first picture is 204 seconds into the test, where the RescueRunner was recently reprogrammed with the new parameters and the oscillations have just been suppressed. A plot of the positions of the lead boat and the RescueRunner during the test is shown in Figure 7.13.

(a) 204 seconds into the test. The RescueRunner has just been reprogrammed with the new parameters and the oscillations have been suppressed.



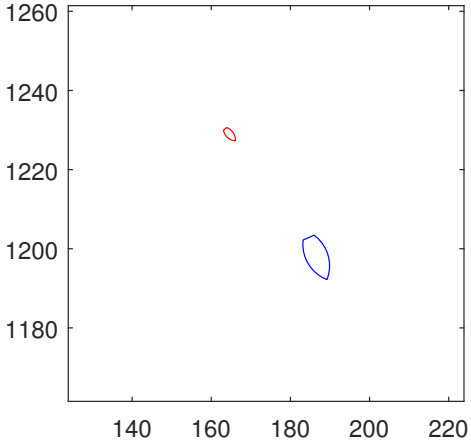
(b) Estimated position of the two boats after 204 seconds. The increase in d_{ref} is apparent from the RescueRunner's position further away from the lead boat.

(c) 214 seconds into the test. A turn is made, which the RescueRunner successfully navigates through.



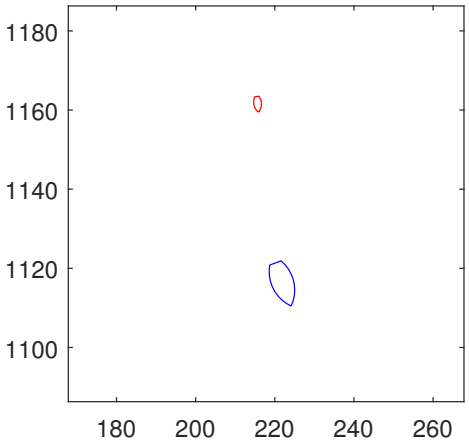
(d) Estimated position of the two boats after 214 seconds.

(e) 230 seconds into the test. More turns are performed in an attempt to induce oscillations.



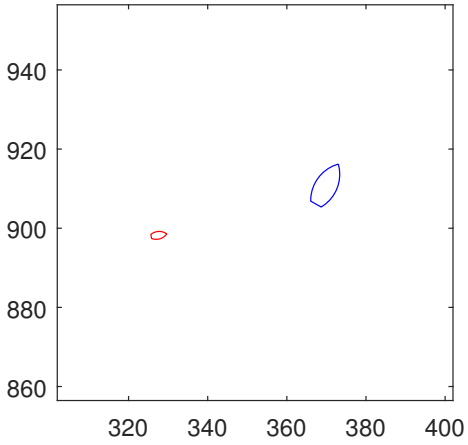
(f) Estimated position of the two boats after 230 seconds.

(g) 240 seconds into the test. The control system is less aggresive which makes the RescueRunner take wider turns, but it does not become unstable. The wider turns are compensated by the RescueRunner following behind at a greater distance.



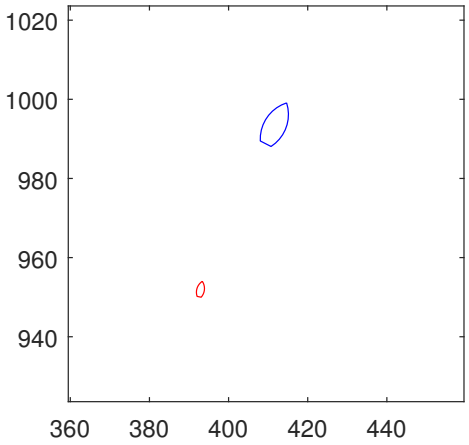
(h) Estimated position of the two boats after 240 seconds.

(i) 274 seconds into the test. The RescueRunner is hit by interfering waves.



(j) Estimated position of the two boats after 274 seconds.

(k) 287 seconds into the test. The RescueRunner cleared the waves without problems and has returned to the desired position behind the lead boat.



(l) Estimated position of the two boats after 287 seconds.

Figure 7.12: Images from the second test performed at Långedrag August 23rd 2016 along with plots created from the estimated heading and positions of the boats. The test shows that the newly tuned autonomous system is able to steer the RescueRunner behind the lead boat and it shows no susceptibility to the oscillations found in the first test.

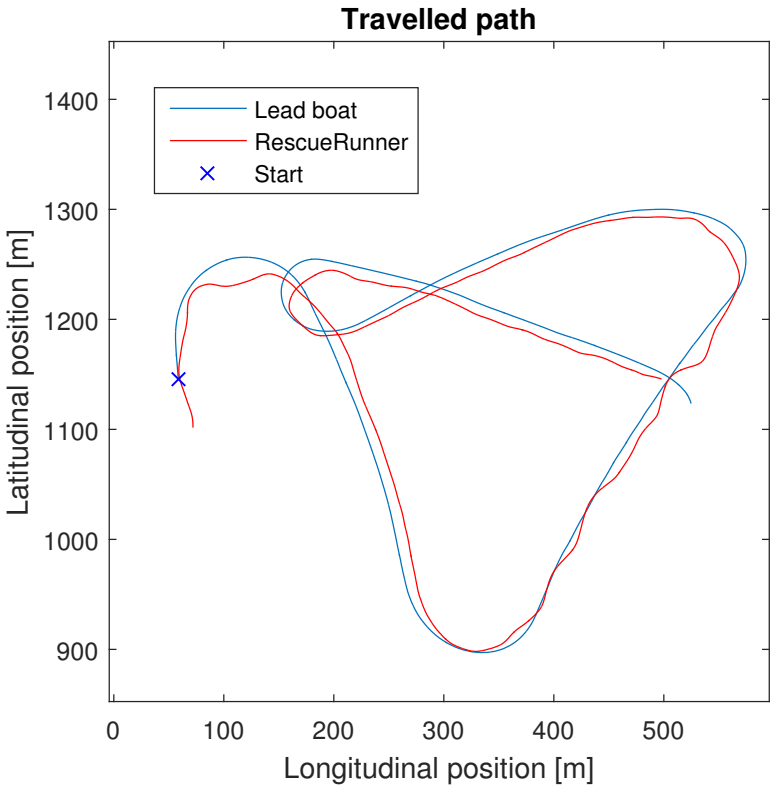
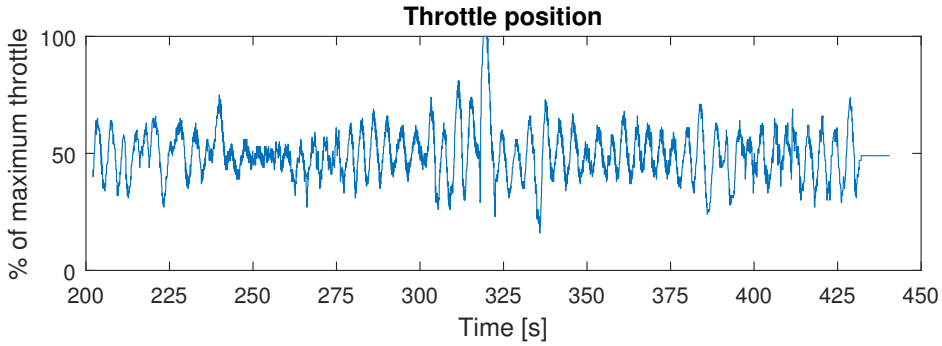
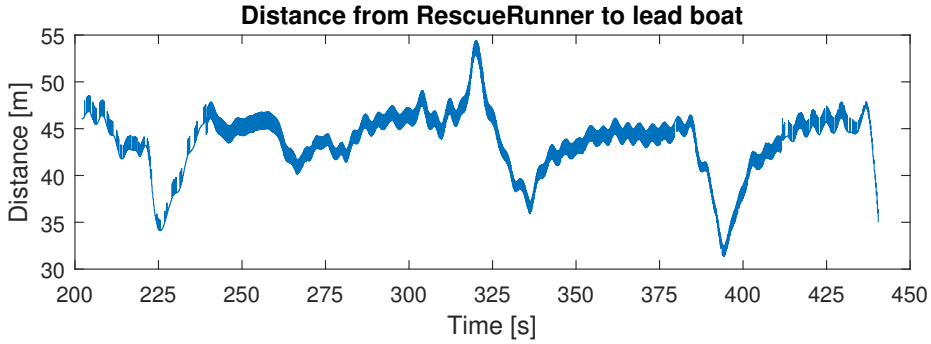


Figure 7.13: A plot of the positions of the lead boat and the RescueRunner sampled during the second test.

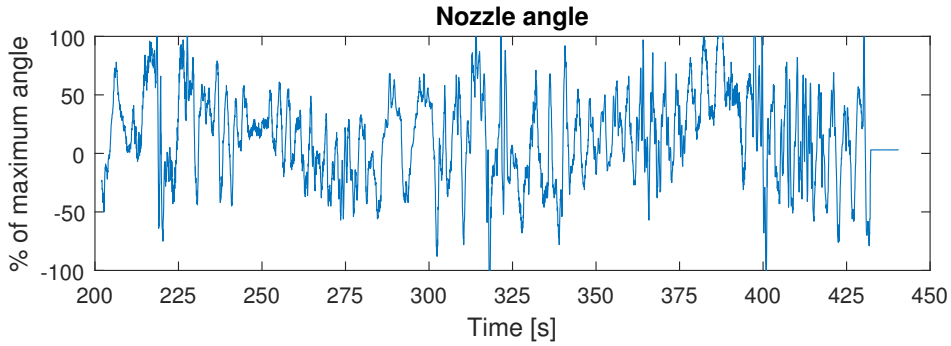


(a) The throttle input as requested by the controller sampled from the second test.

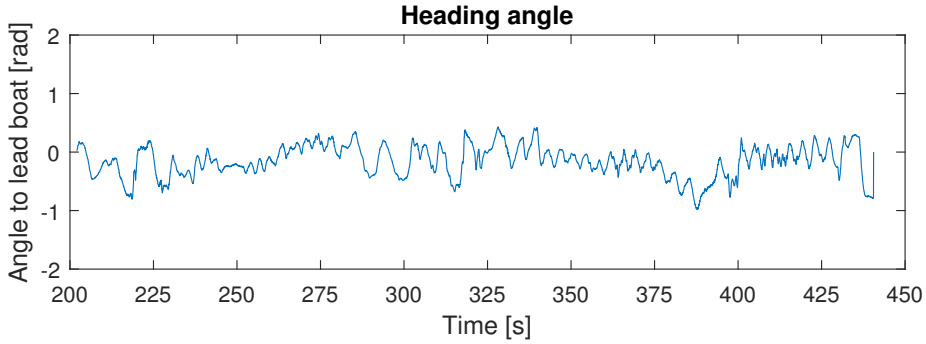


(b) The estimated distance between the two boats sampled from the second test. The high-frequency noise is due to the two GNSS receivers' positions not updating simultaneously.

Figure 7.14: A comparison between the estimated distance between the two boats and the throttle input as requested by the controller, sampled from the second test. The RescueRunner clearly succeeds at keeping a distance of approximately 45 m during the stretches where the lead boat is driving straight, with a throttle input of around 50 %. The curves that the lead boat make can be identified by a sharp increase or decrease in distance between the two boats, depending on where the RescueRunner is positioned relative to the lead boat when the turn is initiated. The turns also give rise to a more fluctuating throttle input. The test is ended around the 440 seconds mark, as seen by the input throttle no longer being updated. The oscillating behavior found towards the end of the first test is not present here due to the increased stability of the newly tuned system.



(a) The nozzle angle input as requested by the controller sampled from the second test.



(b) The angle to the lead boat from the RescueRunner's perspective sampled from the second test.

Figure 7.15: A comparison between the estimated heading angle of the RescueRunner relative to the lead boat and the nozzle angle input as requested by the controller, sampled from the second test. The newly tuned, slightly less aggressive system is slower to compensate for heading angle errors, resulting in the heading angle curve fluctuating around zero with a higher amplitude compared to the first test. However, the oscillating behavior found towards the end of the first test is not present here due to the increased stability of the tuned system.

Plots of the throttle input and the estimated distance between the two boats are shown in Figure 7.14, which shows that the throttle controller mostly succeeds at keeping the RescueRunner at a distance of around 45 m from the lead boat. It also shows that an error from the desired distance tends to arise when the lead boat makes a turn.

Plots of the nozzle angle input and the estimated heading angle of the RescueRunner relative to the lead boat are shown in Figure 7.15, which shows that the angle controller succeeds at keeping the heading around zero, but not as well as the more aggressive system used in the first test. This weakness is offset by the newly tuned system's increased stability and robustness, as it shows no susceptibility to the oscillating behavior shown in the first test.

7.3.3 Comparison to manual control

As a last test the RescueRunner was driven in manual mode to collect data for comparison to the autonomous system. The driver attempted to stay as close behind the lead boat as possible while it performed manoeuvres like in the previous tests. Figure 7.18 shows the RescueRunner in position behind the lead boat. In Figures 7.16 and 7.17 the distance between the two boats as well as the angle representing the relative bearing from the RescueRunner to the lead boat is presented. The control inputs are not included since in manual mode these are never treated by the control system and therefore not easily available for logging. The figures show that when driving manually, the distance between the two boats is around 23 m. This can be seen as a measure of the distance which intuitively feels ideal when following a lead boat. The angle of the RescueRunner relative to the lead boat is around zero for the majority of the test, which hints at the heading estimation being correct, and that controlling that angle to zero should result in a behavior similar to having a driver manually maneuvering the RescueRunner.

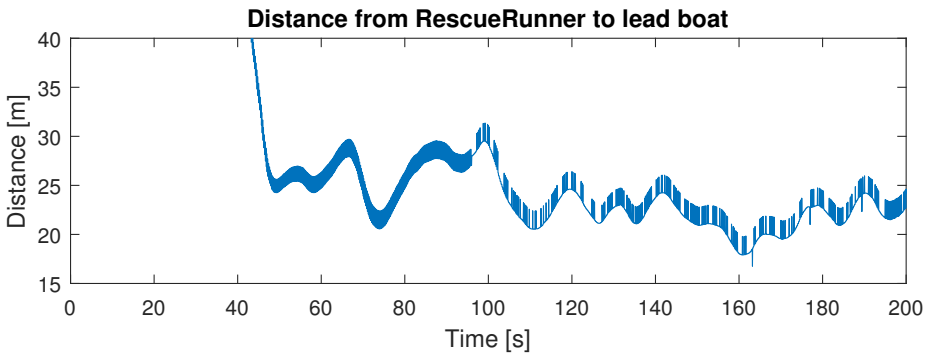


Figure 7.16: The estimated distance between the two boats while the RescueRunner was driven manually.

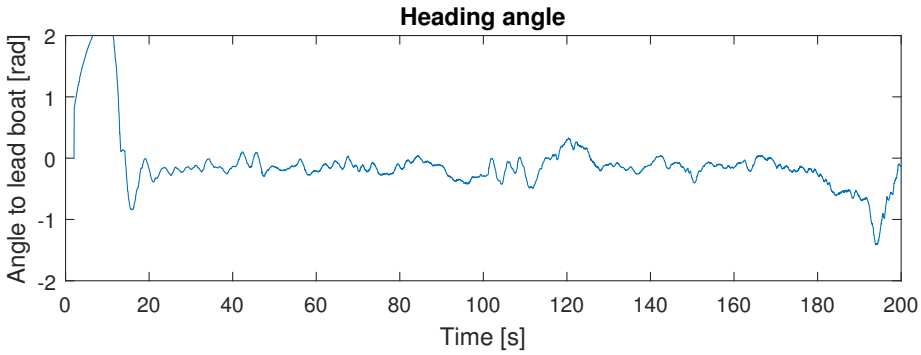


Figure 7.17: The angle to the lead boat from the RescueRunner while the RescueRunner was driven manually.

Figure 7.18: An image of the RescueRunner following behind the lead boat while driven manually.

8

Discussion and conclusions

A state observer and a control algorithm that successfully steers the RescueRunner to follow a lead boat have been developed. The system's performance was evaluated through a series of tests to evaluate the individual components as well as the system in its entirety. The work proves that the estimated states (positions, velocities and headings of the two boats) are enough to achieve an autonomously driven water scooter.

While the autonomous system is mostly successful at following the lead boat, it has some weaknesses. One of the most important weaknesses discovered during the tests performed in water is that the system can be put into a state of constant oscillations back and forth. This was remedied by increasing the desired distance and slightly decreasing the control system's aggressiveness, but meant that the system could not reliably stabilize the RescueRunner at the ideal distance of around 20 m from the lead boat. The oscillating behavior was at the time of the tests believed to be due to an incorrect controller tuning, but after studying the data one can conclude that the heading of the RescueRunner relative to the lead boat is incorrect in some places where the RescueRunner is turning. The cause of these errors are so far unknown, but one possibility is that the rough sea causes the orientation filter to lose track of the tilt of the RescueRunner. This would result in the measurements of the gyroscopes and magnetometer being interpreted incorrectly.

The tests show that the GNSS receivers most of the time provide estimates of the relative position with a precision better than ± 4 m, which was deemed necessary to place the RescueRunner within the wake formed by the lead boat. However, the level of uncertainty will limit how fast a control system can respond to errors. Furthermore, since the velocity estimates are derived from the same GNSS data there is a delay before a change is detected which makes them less useful for reducing future errors.The precision could be improved by using the method for estimating the relative positions of several GNSS receivers presented in [17]. Or, instead of using the internal filters of the GNSS receivers, which are limited in terms of motion model options, a tailor-made filter for the RescueRunner could be used. This would allow the use of the orientation estimates and accelerations measured by the IMU to improve the estimates. However, either of these alternatives would likely have taken longer time, and was not seen as feasible within this project's timeframe.

The radio link used in the system proved reliable for ranges beyond those necessary during the tests. However, there are functionalities that may be implemented in the

future which could require a longer range. An example of this would be to call on the RescueRunner from long distances. The range of the specific radio transceivers is limited by the maximum power allowed for public use at that frequency band. However, should a longer range be necessary, a directed antenna could increase the range without increasing the signal power. Such an antenna would need a system to actively direct it. The system was designed to allow for short communication outages before the RescueRunner exits the autonomous mode. This solution successfully avoids the potential problems that would arise from the RescueRunner quickly switching between modes, while at the same time ensuring a safe operation. The ability to request data from the RescueRunner proved very useful during the testing, as the control signals could be monitored. The same principles could be used in later versions of the system to provide the user with feedback and warnings for e.g. communication losses or a low fuel level in the RescueRunner.

While the current orientation filter performs mostly acceptable, additional sensors could be implemented in future work to assist with estimating the heading of the RescueRunner relative to the lead boat. This could improve the filter's responsiveness, robustness and precision. An example would be a frontal camera mounted on the RescueRunner with software to recognize the lead boat and measure its relative bearing.

There are several aspects that can be improved about the control scheme. An example of this is that the nozzle angle is controlled independently of the throttle, which means that a given nozzle angle control output give rise to widely varying angular accelerations of the RescueRunner depending on the current throttle. Another example is that the control scheme does not take any waves into account, including the ones in the lead boat's wake. Should a more advanced control system be attempted, there may be a benefit to knowing the RescueRunner's pitch and roll as they have an impact on its steering capabilities. As mentioned in chapter 3, both of these can be extracted from the estimated quaternion. Nevertheless this first attempt, mainly intended to test the other parts of the system, shows that the task presented by the Follow Me project is definitely achievable.

Bibliography

[1] Arduino. Arduino due overview. https://www.arduino.cc/en/Main/ArduinoBoardDue. 2016-09-12.

[2] Arduino. Wire library. https://www.arduino.cc/en/Reference/Wire. 2016-09-12.

[3] Arduino. nRF24L01+. http://playground.arduino.cc/InterfacingWithHardware/Nrf24L01, 2015.

[4] Safe at Sea. rescuerunner.pdf. http://www.rescuerunner.com, 2016.

[5] H.S.M. Coexter. *Regular Complex Polytopes*, pages 64–67. Cambridge university press, 1974.

[6] Elecfreaks. 2.4G Wireless nRF24L01p with PA and LNA. http://www.elecfreaks.com/wiki/index.php?title=2.4G_Wireless_nRF24L01p_with_PA_and_LNA, 2015.

[7] P. Falcone, F. Borrelli, J. Asgari, H.E. Tseng, and D. Hrovat. Predictive Active Steering Control for Autonomous Vehicle Systems. *Control Systems Technology, IEEE Transactions on*, 15(3):566–580, May 2007.

[8] F. Falkman. Follow-me, little rescue boat! How might we make the Rescuerunner follow a bigger rescue boat automatically, on a safe distance? http://www.surtsey.org/projects/follow-me/, 2014.

[9] National Centers for Environmental Information (NCEI). Further understanding of geomagnetism. http://www.ngdc.noaa.gov/geomag/geomaginfo.shtml. 2016-09-12.

[10] National Centers for Environmental Information (NCEI). Magnetic field calculator. http://www.ngdc.noaa.gov/geomag-web/#igrfgrid. 2016-09-12.

[11] R. Garnett and R. Stewart. Comparison of gps units and mobile apple gps capabilities in an urban landscape. *Cartography and Geographic Information Science*, 42(1):1–8, 2015.

[12] F. Gustafsson. *Statistical Sensor Fusion*. Studentlitteratur, second edition, 2012.

[13] F. Gustafsson. *Statistical Sensor Fusion*, pages 349–351. Studentlitteratur, second edition, 2012.

[14] K. Gustavsson. UAV Pose Estimation using Sensor Fusion of Inertial, Sonar and Satellite Signals. Master's thesis, Uppsala universitet, June 2015.

[15] Wm. R. Hamilton. Theory of quaternions. *Proceedings of the Royal Irish Academy (1836-1869)*, 3:1–16, 1844.

[16] Andrew Hanson. *Visualizing quaternions*. Morgan Kaufmann, Amsterdam;San Francisco, CA;Boston;, 2006.

[17] W. Hedgecock, M. Maroti, J. Sallai, P. Volgyesi, and A. Ledeczi. High-Accuracy Differential Tracking of Low-Cost GPS Receivers. In *ACM*, pages 221–234, 2013.

[18] G. Hendeby, F. Gustafsson, and N. Wahlström. Teaching sensor fusion and kalman filtering using a smartphone. *{IFAC} Proceedings Volumes*, 47(3):10586 – 10591, 2014. 19th {IFAC} World Congress.

[19] B. Hofmann-Wellenhof, Herbert Lichtenegger, Elmar Wasle, and SpringerLink (e-book collection). *GNSS–global navigation satellite systems: GPS, GLONASS, Galileo, and more*, page 17. Springer, Wien;New York;, 2008;2007;.

[20] B. Hofmann-Wellenhof, Herbert Lichtenegger, Elmar Wasle, and SpringerLink (e-book collection). *GNSS–global navigation satellite systems: GPS, GLONASS, Galileo, and more*, pages 8–12. Springer, Wien;New York;, 2008;2007;.

[21] The MathWorks Inc. Arduino Support from MATLAB. `http://se.mathworks.com/hardware-support/arduino-matlab.html`.

[22] The MathWorks Inc. Data Types. `http://se.mathworks.com/help/matlab/data-types_data-types.html`.

[23] J. Law and R. Rennie, editors. *A Dictionary of Physics (Oxford Quick Reference)*. Oxford University Press, seventh edition, 2015.

[24] G. Lindberg, H. Bergh, and Skötte J. Elektronisk manövrering av rescuerunner, 2016. Chalmers University of Technology.

[25] T. Marius Jensen. Waypoint-following guidance based on feasibility algorithms. Master's thesis, Norwegian University of Science and Technology, 2011.

[26] Nordic Semiconductor. *nRF24L01+ Single Chip 2.4GHz Transceiver Product Specification v1.0*, September 2008. Rev. 1.

[27] P. Plumet, C. Pêtrès, M.A. Romero-Ramirez, B. Gas, and S.H. Ieng. Toward an autonomous sailing boat. *JOURNAL OF OCEANIC ENGINEERING*, 40(2):397–405, 2015.

[28] S. Särkkä. *Bayesian Filtering and Smoothing*. Cambridge University Press, 2013.

[29] Swedish Sea Rescue Society. `http://www.sjoraddning.se/ine-english/`, 2016.

[30] Fredrick S. Solheim, Jothiram Vivekanandan, Randolph H. Ware, and Christian Rocken. Propagation delays induced in gps signals by dry air, water vapor, hydrometeors, and other particulates. *Journal of Geophysical Research: Atmospheres*, 104(D8):9663–9670, 1999.

[31] TMRh20. Optimized High Speed Driver for nRF24L01(+) 2.4Ghz Wireless Transceiver. `http://tmrh20.github.io/RF24/index.html`, 2014.

[32] Gergely Vass. Avoiding gimbal lock. *Computer Graphics World*, 32(6):10–11, 06 2009. Copyright - Copyright COP Communications, Inc. Jun 2009; Document feature - Photographs; Illustrations; Last updated - 2013-06-02.

[33] X. Yun, E. R. Bachmann, and R. B. McGhee. A simplified quaternion-based algorithm for orientation estimation from earth gravity and magnetic field measurements. *IEEE Transactions on Instrumentation and Measurement*, 57(3):638–650, March 2008.

[34] P. Zhou, M. Li, and G. Shen. Use it free: Instantly knowing your phone attitude. In *Proceedings of the 20th Annual International Conference on Mobile Computing and Networking*, MobiCom '14, pages 605–616, New York, NY, USA, 2014. ACM.

A

Multiple read I2C in MATLAB

Below are two modifications made to the i2cdev.m file that allowed for reading a set number of consecutive bytes.

Before the modification, the function contained the following lines:

```
function [out,output] = readRegister(obj, register, precision)
numBytes = obj.SIZEOF.(precision);
```

After the modification, those lines were changed to the following:

```
function [out,output] = multiReadRegister(obj, register, numbytes, precision)
numBytes=numbytes;
```

In other words an extra input to the function was added which specifies the number of bytes to read, instead of being determined by the size of the data type.

B

Discretizing a PID controller using the Tustin method

To enable a PID controller being used in discrete-time computer code languages such as C++, its continuous-time formulation must first be discretized. This appendix will show how a PID controller with a filtered derivative (also known as a PIDf controller) is discretized using the Tustin method.

The Tustin method, also known as the bilinear transform, is a method for transforming a continuous-time system representation to discrete-time. The basic principle is that given a Laplace formulation of a continuous-time system, the variable s is replaced with an approximation using the discrete-time variable z.

z has a well defined behaviour in discrete systems, which is that z^{-1} is a unit delay. If given a discrete time variable $y(n)$, then multiplying its Z-transform $Y(z)$ with z^{-1} is equivalent to using its value at the previous time index $y(n-1)$. This is presented mathematically in (B.1).

$$Y(z)z^{-1} = \mathcal{Z}\{y(n-1)\} \tag{B.1}$$

The discrete-time replacement for s used in the Tustin method is presented in (B.2), where h is the sampling time of the discrete system.

$$s = \frac{2(z-1)}{h(z+1)} \tag{B.2}$$

The discretization of the PID controller starts from its continuous-time transfer function shown in (B.3), where $Y(s)$ is the PID controller's output and $U(s)$ is its input (also known as the control error).

$$\frac{Y(s)}{U(s)} = K_p + \frac{K_i}{s} + K_d \cdot \frac{s}{1+T_f s} \tag{B.3}$$

The Tustin method is then applied by replacing the variable s as described in (B.2), resulting in the discrete-time system shown in (B.4).

$$\frac{Y(z)}{U(z)} = \frac{4K_d(-1+z)^2 + (2T_f(-1+z) + h(1+z))(2K_p(-1+z) + hK_i(1+z))}{2(-1+z)(2T_f(-1+z) + h(1+z))} \tag{B.4}$$

Both sides of the equation are then multiplied by the denominators. Next, each side is rewritten as polynomials of z, resulting in an equation on the form shown in (B.5).

$$a_0Y(z) + a_1z^{-1}Y(z) + a_2z^{-2}Y(z) = b_0U(z) + b_1z^{-1}U(z) + b_2z^{-2}U(z) \tag{B.5}$$

The inverse Z-transform is then applied using the rule presented in (B.1). This results in an equation on the form shown in (B.6).

$$y(n)a_0 + y(n-1)a_1 + y(n-2)a_2 = u(n)b_0 + u(n-1)b_1 + u(n-2)b_2 \tag{B.6}$$

Finally, the output of the PID controller $y(n)$ is found as a function of its past outputs $y(n-1)$, $y(n-2)$ and its current and past inputs $u(n)$, $u(n-1)$, $u(n-2)$ as shown in (B.7). Solving the equation in (B.4) results in the coefficient values shown in (B.8)-(B.13).

$$y(n) = \frac{u(n)b_0 + u(n-1)b_1 + u(n-2)b_2 - y(n-1)a_1 - y(n-2)a_2}{a_0} \tag{B.7}$$

$$b_0 = (2T_f + h)(2K_p + hK_i) + 4K_d \tag{B.8}$$

$$b_1 = -8K_d - 2K_p(2T_f + h) + hK_i(2T_f + h) - 2T_f(2K_p + hK_i) + h(2K_p + hK_i) \tag{B.9}$$

$$b_2 = 4T_fK_p - 2hT_fK_i - 2hK_p + h^2K_i + 4K_d \tag{B.10}$$

$$a_0 = 4T_f + 2h \tag{B.11}$$

$$a_1 = -8T_f \tag{B.12}$$

$$a_2 = 4T_f - 2h \tag{B.13}$$

Given a sampling time h and a set of PID design parameters K_p, K_i, K_d, T_f, this equation can be formulated and solved recursively in a discrete-time computer code language to generate outputs of the PID controller. These outputs can then be applied to control a plant.