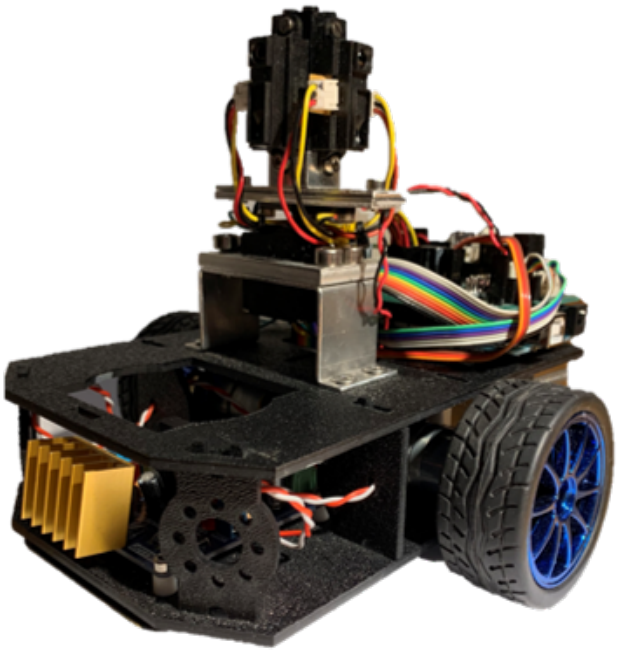
Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

Halvorsen, Karoline

Analysis of position estimation in a dead reckoning navigation robot

Master's thesis in Cybernetic and robotik
Supervisor: Onshus, Tor
January 2020



NTNU
Norwegian University of
Science and Technology

Halvorsen, Karoline

Analysis of position estimation in a dead reckoning navigation robot

Master's thesis in Cybernetic and robotik
Supervisor: Onshus, Tor
January 2020

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

NTNU
Norwegian University of
Science and Technology

To my mother: for all the support through my years of study

Preface

This master thesis marks the end of five years at the Norwegian University of Science and Technology (NTNU). The work is a part of the course, *TTK4900- Engineering Cybernetics, Master's Thesis*, at the Cybernetic Department.

The work conducted in this master thesis is based on knowledge from the specialization project [1], but with a different robot and another robot application. The specialization project focused on determining the precision in the robot's internally estimated position, and how to improve it through work on the gyroscope. The master thesis takes a closer look at all the sensors and the position estimation application. In the beginning of the thesis, time has also been spent on getting the system to a functional state.

Recognition and acknowledgement

I want to thank my supervisor Professor Tor Onshus, for guidance and interesting discussions through the work on this thesis.

A special thanks to my family and Amund Fjøsne, who has supported me and cheered me up trough the project.

Thanks to the employers at IKT and the Mechanical workshop at the Cybernetic Department, for letting me use the soldering equipment, fastening the motors to the robot and supplying materials to build the larger testing court.

Thanks to Torgeir Myrvang, for letting me use the software he had written to easy debug the robot with a USB cable, and to all the fellow students at the office for technical discussions.

Karoline Halvorsen
Trondheim, January 2021

Problem Description

Background
"*Knowledge is power*", is a phrase attributed by Thomas Hobbes, dating back to 1668 [2]. As the world gets more complex, inaccuracies and fake news can create chaos and division. It is not only in society where precision and truth are essential. In the navigation of robotics, it is crucial to have correct and accurate data, as companies work toward industry 4.0[1]. With more communication between different systems, and robots becoming more autonomous, the reliability of accurate information is more and more critical. An error in robot position or orientation can cost a firm time and money.

This robot project faces many of the same challenges. The precision in the position has to be accurate for the robot not to collide, or give false information. For some companies, the cost of high precision sensors may be prohibitively high. Even if the robot has low-budget sensors, without the highest accuracy, the problem of how to improve precision in the estimated position is highly relevant.

Objective
Evaluate the error, investigate its sources, and provide suggestions for improvements of the robots internal pose estimation.

List of tasks that have been carried out

1. Correct initial problems in software and hardware if they occur
2. Test gyroscope for drift and noise
3. Test the accelerometer
4. Test the encoder
5. Test the compass
6. Test the IR sensors
7. Test driving performance in a square test and testing different courts
8. Find solution to improve the position estimation

Tests on the autonomous parking have also been executed.

[1] Industry 4.0 is a term that represents the fourth stage in the industrial revolution. Here, the technology is linked more together in real-time systems, automation and machine learning [3].

Summary

In this master thesis, the focus area has been to investigate the sources for the error in the Arduino robot's position estimation. This has been done by evaluating position estimation in square tests, to find the position accuracy. In addition, the estimation application and all sensors connected to the robot, are evaluated.

In the beginning of the project, some bugs were detected. Most were fixed, but the problem with the lidar was not solved and was excluded from the project. In the continuous square test, the robot tended to get a larger error after driving multiple rounds. It was detected that the robot is stochastic in the way it behaves. This might be due to a problem with FreeRTOS or an initialization problem. This has not been investigated further but is left as a suggestion for future work. The robot was also tested in a more extensive track and in the round court, with varying results.

This project has, in addition, investigated the angular velocity, *dTheta*, of the robot's heading. *dTheta* is a direct result from the weighted data measurement from the gyroscope and encoder. The estimated robot heading is an integration of *dTheta*. The heading of the robot is found to have an increasing error over time, due to integration. The error varied when changing the weighting between the gyroscope and encoder.

The thesis will also describe how all sensors are tested. The five sensors on the robot were tested and found usable. The **gyroscope** was found to have a drift of 0.02 degrees in 30 minutes. The bias was found to be around -5 to -6 dps, but two outliers have been found, with an increase of 300%. The placement of the **accelerometer** corrupts the data. When the robot is standing still, all axes can measure 1 g in the vertical direction, but had a measurement error of 0.49 to -0.0320 g. The new **encoders** were then tested and found to have between 90-110 ticks per wheel rotation. In the last period of this thesis work, it was discovered that the function used, to extract encoder ticks, was wrong. After it was fixed, the encoder ticks were found to be 225 ticks per wheel rotation. The **compass** was calibrated, which resulted in a more accurate measurement compared to the previous calibration. The new calibration of the compass gives an x offset of -456 milligauss and a y offset of 246 milligauss. To measure distance, the **IR** sensors were calibrated. The old calibration often measures too far, at shorter distances (up to 30 centimetres). The new calibration underestimated the distance. However, the new calibration was fluctuating more than than the old one, which resulted in the conclusion of keeping the old calibration.

To improve the position estimate, a design for an extended Kalman filter to fuse the sensor data, is presented. This has not been implemented and is left for future work.

Sammendrag

I denne masteroppgaven har fokusområdet vært å undersøke årsaken til feil i Arduino-robotens posisjonsestimering. Dette har blitt gjort ved å evaluere posisjonsestimering under firkanttesten, for å finne posisjonsnøyaktigheten. I tillegg evalueres estimeringsapplikasjonen og alle sensorer som er koblet til roboten.

I begynnelsen av prosjektet ble noen feil oppdaget. De fleste ble løst, utenom problemet med lidaren, derfor ble den utelatt fra prosjektet. Roboten har en tendens til å få en større feil etter kontinuerlige runder i firkanttesten. Det ble oppdaget at roboten tilsynelatende har en stokastisk oppførsel. Dette kan skyldes et problem med FreeRTOS eller et initialiseringsproblem. Det er ikke undersøkt nærmere, men er satt som et forslag for videre arbeid. Roboten er i tilegg testet i en mer omfattende bane, og i rundbanen, med varierende resultat.

Denne oppgaven har i tilegg undersøkt vinkelhastigheten, *dTheta*, i robotens heading. *dTheta* er et direkte resultat av vekting av målinger fra gyroskopet og enkoderen. Den estimerte robot-headingen er en integrasjon av *dTheta*. Robotens heading har en økende feil over tid, som følge av denne integrasjonen. Feilen varierte når vektingen mellom gyroskopet og enkoderen ble endret.

Oppgaven vil også beskrive hvordan alle sensorer blir testet. De fem sensorene på roboten ble testet og bedømt til å være brukbare. **Gyroskopet** hadde en drift på 0,02 grader i løpet av 30 minutter. Bias ble funnet å være rundt -5 til -6 dps, men hadde to avvik, med en økning på 300 %. Plasseringen av **akselerometeret** ødelegger tildels dataene. Når roboten står stille, kan alle aksene måle 1 g i vertikal retning, men hadde en målefeil på 0,49 til -0,0320 g. Videre ble de nye **enkoderne** testet, og funnet å ha et sted mellom 90-110 tikk per hjulrotasjon. I den siste perioden av arbeidet med masteren ble det oppdaget at funksjonen brukt til å hente ut antall enkoder-tikk, var feil. Etter at den var fikset, ble det funnet å være 225 steg per hjulrotasjon. **Kompasset** ble kalibrert, noe som resulterte i en mer nøyaktig måling sammenlignet med forrige kalibrering. Den nye kalibreringen av kompasset gir en x-offset på -456 milligauss og en y-offset på 246 milligauss. For å måle avstand ble **IR**-sensorene kalibrert. Den gamle kalibreringen måler ofte for langt på korte avstander (opptil 30 cm), og den nye kalibreringen underestimerte avstanden. Den nye kalibreringen fluktuerer mer enn den gamle, hvilket resulterte i konklusjonen om å beholde den gamle kalibreringen.

For å forbedre posisjonsestimatoren presenteres et design som bruker et utvidet Kalmanfilter til å fusjonere sensordataene. Dette er ikke implementert og er forelsått som fremtidig arbeid.

Conclusion

Different sources of error have been found in the robot's estimation of position and orientation (POSE), with the constraints of evaluating the robot's sensors and its position estimator. The work carried out for this thesis concludes that the robot is not optimized for accurate position estimation. The robot has an error in its position estimate, and it varies from a few, to more than 20 centimetres. The error also develops over time and is often increasing.

Today, only encoders and gyroscope are used to calculate the position estimate in the Arduino robot. None of the sensors have mechanisms that protect against noise, e.g., a filter to conserve signal integrity. Noise has been found in the gyroscope, and the full range of the encoders is not utilised. The encoder resolution can be improved by using both phases and recording all encoder ticks. In the robot application, integration and calculation of position, leads to errors in the estimator. This calculation error has not been quantified.

The robot can improve POSE estimation by utilising several of the sensors. It is concluded that the accelerometer works. However, to effectively use it in the robot application, the IMU must be moved or the accelerometer data mathematically transformed to the centre of rotation. The compass works as well, but must be calibrated if the robot is used near objects that disturb the magnetic field. Using the sensors above, an extended Kalman filter can improve the robot position and orientation estimate [4].

The calculation of distance using the IR sensors should be executed in millimetres, to avoid rounding error. In addition, using the full range of the ADC will improve the measurement, by increasing the resolution. The robot application could detect the docking station and re-calibrate its position, to improve the position estimate.

Table of Contents

List of Tables

List of Figures

Abbreviations

ADC	=	Analogue to Digital Converter
BLE	=	Bluetooth Low Energy
CCW	=	Counterclockwise
CPU	=	Central Processing Unit
CW	=	Clockwise
DC	=	Direct Current
dps	=	degrees per second
EMA	=	Exponential Moving Average
GPS	=	Global Position System
GUI	=	Graphical User Interface
IDE	=	Integrated Development Environment
IDP	=	Integrated Development Platform
IMU	=	Inertial Measurement Unit
INL	=	Integral Non-Linearity
IR	=	Infrared
NTNU	=	Norwegian University of Science and Technology
PCB	=	Printed Circuit Board
PI	=	Proportional Integral
POSE	=	Position and Orientation
RPM	=	Rotations per Minute
RTOS	=	Real-Time Operating System
RX	=	Receive
SLAM	=	Simultaneous Localisation and Mapping
SSNAR	=	System for Self-Navigating Autonomous Robots
TX	=	Transmit

Chapter 1

Introduction

The reader will find the thesis structured in the following way:

- Chapter 1: Introduces the project and describes the equipment used.
- Chapter 2: Describes how the system works in more detail. It also gives an explanation of the tasks that have been executed to fix initial problems.
- Chapter 3: Explains the experimental method and theoretical background needed to solve this thesis problem.
- Chapter 4: Reviews the results and findings from the experimental tests.
- Chapter 5: Analyses and discusses the results found.
- Chapter 6: Suggests relevant tasks for future work.

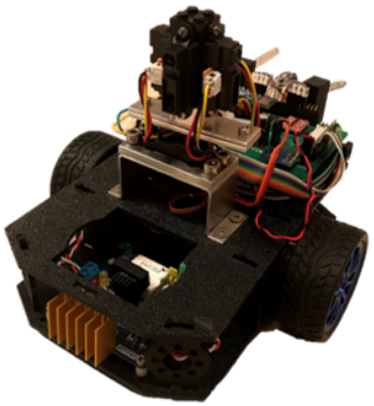


Figure 1.1: The Arduino robot

1.1 The Robot Project

The Robot project is a hands-on project available for the fifth-year students at the cybernetic department at NTNU. It is worked on as a specialisation project and master thesis, and is supervised by professor Tor Onshus.

The goal of the project is to navigate an unknown environment with a robot. The navigation method uses dead reckoning. This is a method where the position and orientation is based on initially given values and calculation from the driven distance and heading [5]. The system assumes there are no known markers in the area. The system consists of the robot collecting data and calculate its internal position and orientation. The robot sends the information to the server, that constructs a map and calculates target points and path. This information is then sent back to the robot for execution.

The robot project was founded in 2004 and consisted only of a single robot constructed of LEGO bricks, therefore called the LEGO - robot project. At this point the server was written for Linux, by Håkon Skjelten [6]. The project has grown from one robot to multiple, and today there exists six robots. Previously the different robots were built with Lego and with different microchips, like the AT91SAM7S256 chip used in the NXT robot [1, chapter 2.1.1].

In the last years the robots have been upgraded to use more off-the-self components. Today the robots use an Arduino board, connected to different commercial sensors. With no Lego robot parts left, the name *Lego project* is phased out and replaced by *the Robot project*. It is not only the robot that has changed since the beginning of the project. Furthermore, the server has also been under development. It was first written for Linux, then in Java and is now in C++.

Today the development of the different robots are at different stages. Some use different control systems and communicate only with certain servers. Nevertheless, the overall goal for the project is the same, to navigate in an unknown environment. In the end, the goal is to have multiple robots navigate autonomously[1] and collaborate in the mapping, without the use of the server. The server will then function as a graphical user interface (GUI) to display the map for the user.

In this master thesis, the robot used is the Arduino robot that can have both infrared (IR) or lidar sensors connected to it, to determine the distance to an obstacle. A future goal is to achieve autonomous parking of the robot. To achieve this the position error of the robot must not be over 15 millimetres (defined in the robot application). Improving the position estimate must first be solved before working on autonomous parking, therefore this thesis investigates reasons for the position error.

[1] Independent or self-government [7]

1.2 Motivation

Today in the digitized and more robotized world, position and navigation are widely researched. For many years Global Position System (GPS) has been used to find the position of an object, like in the navigation system of a car. However, the GPS is known to have some downsides when it comes to precision, as the accuracy is on meter-level and can be hard to use in a specific environment [8]. For example, where there are high buildings that blocks the signal or mountains that shields the signal to the satellite. In addition the GPS will not work inside a building. In this robot project, there is not possible to use GPS, as the accuracy must be on centimetres and millimetres-level, and the robot is used inside. Today projects that navigates without the use of GPS, is found in many applications, like robot vacuum cleaners or the MIT project where they use drones to look for lost hikers in an environment where there are no GPS [9].

The task of getting the most precise position estimate for a robot that has off-the-shelf-sensors, without the highest quality, is an interesting challenge. Today many projects want to have the best equipment and best sensors, but they are often too expensive. To be able to learn how to get the best out of the equipment that is given, is a skill that is useful for a future career.

In the last few years, several students have tried to improve the position estimation in the robot. This is a crucial part of the project, for the robot to operate with best possible accuracy. For me, it is motivating to find where errors occur, and if possible, find a way to counteract, and reduce, these errors.

1.3 Equipment

The equipment used in this project is described in this section. In addition to the hardware and software, access to the camera room, B333 [10] at NTNU was given, which was necessary to use the OptiTrack system. Here it was possible to get an objective measurement from the movement of the robot.

1.3.1 Hardware

The hardware was provided by the department of cybernetics at NTNU. In addition, a multimeter and soldering equipment was provided by the IKT employees. The following list summarises the hardware used in the project.

- Arduino robot with four IR sensors and one lidar.
- USB wire to program the robot.
- Server and periphery dongle of type nRF51.
- Tenergy, universal smart charger was shared between all students that worked with a robot.

1.3.2 Software tools

The software tools were already installed or downloaded, free of charge, due to the work from the specialization project [1]. The software was used on a 64-bit computer, with Windows and Intel Core i7-8700 CPU. In addition to this, NetBeans, and AVRdude was executed on a different computer, a 64-bit laptop, with Windows and Intel Core i5 CPU. The following list summarises the software tools used in the project.

- NetBeans IDE 8.02, to execute the server application.
- Java 8 , a requirement for downloading and running NetBeans.
- Atmel Studio 7, is an integrated development platform (IDP) [11], used in this project to develop and debug the robot application.
- Arduino IDE 1.8.13, is an integrated development environment (IDE) [12], used to check the Arduino board.
- AVRdude, is an open source software used to program Atmel AVR microcontrollers [13]. This was used to program the robot with a hex file.
- Matlab, with license given by NTNU. Matlab has been used in processing data for this thesis.
- Motive, is the software used by the camera setup OptiTrack, in room B333 at NTNU. The software is found on the stationary computer in the room.
- Microsoft Project [14] has been used to keep track of the tasks and schedule.

Chapter 2

Background

A description of the Arduino robot hardware and software is described in this chapter, along with the Java server application. The chapter also contains the work done to fix initial problems in the project.

Conventions

Firstly, here are some definitions used in this project. The robot is in previous reports called the Arduino robot or the Arduino - LIDAR robot. In this report, the LIDAR was not used. The robot will be referred to as the Arduino robot or the Arduino - IR robot, or simply the robot. The robot has its Cartesian coordinate system, shown in figure 2.1. The direction the robot is driving is called the robot heading. When the robot turns, the direction is defined around the z-axis, the clockwise or counter-clockwise direction is used to describe the rotation.

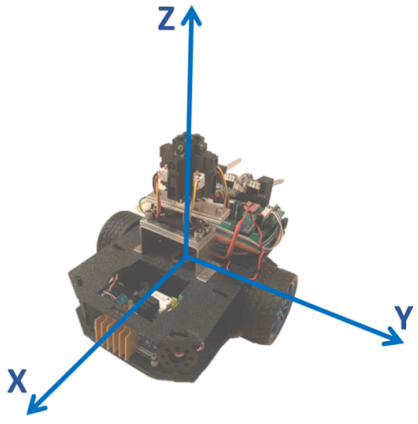


Figure 2.1: Definition of the body coordinate frame of the robot

2.1 Robot description

The robot in this master thesis is best described in two parts, hardware and software. Firstly, a brief summary of the previous work is given. Before the hardware, the construction of the physical components of the robot, is presented. After this the software is covered. The software consists of code written in C. Finally, the steps required to program the robot are listed.

2.1.1 Previous work

The Robot project has been existing for many years. The first robot was built in 2004 by Skjelten [6]. The robot used in this master project was build in 2017.

- 2016: Ese [15] developed the robot application.
- 2017: Jensen [16] ordered parts and builds the robot based the design by Andersen and Rødseth from 2016 [17].
- 2018: Jensen [18] implemented the lidar and made the necessary software changes.
- 2020: Dypbukt [19] investigated the position estimation and implemented error factors on the gyroscope and encoders.

2.1.2 Hardware

The Arduino robot is a two-wheel driven robot that rests on supporting wheel which is fastened in the rear of the robot. The robot has a length of 195 millimetres and a width of 125 millimetres. It weighs 1.1 kg and consists of two plastic plates, where every component is attached. A detailed explanation of the choices made in selecting the hardware is found in Jensen 2017 [16, chapter 2].

Arduino
The main component is the Arduino board, seen in figure 2.2. It has multiple digital inputs and outputs, along with some analogue inputs. The Arduino board is based around the ATmega 2560 microcontroller, from Atmel [20]. The robot application runs on this controller. The Arduino is programmed using a USB-cable. An analogue to digital converter (ADC) for transforming an analogue input signal to a digital reading, is integrated into the microcontroller. The internal ADC has 10-bit resolution [20], but the existing robot application uses only the eight most significant bits. The reason for this is not fully known, but it is assumed that it simplified the implementation of the bit-to-centimetre look-up table for the IR sensors. Eight bits of resolution will give a table of 256 items, while 10 bits of resolution would require a table with 1024 entries. The design requires four such tables (one pr. IR sensor), and each entry is stored as an eight-bit value, giving a total required memory of 4 kB, if using 10 bits resolution. The Arduino-board is connected directly to the battery, since it has a recommended input voltage of 7 to 12 V [21, Technical specs]. The Arduino outputs both 3.3 V and 5 V, which is necessary for the different sensors used by the robot. For more information, the reader can find this in the datasheet for the Arduino [20] and the ATmega 2560 [22].

Figure 2.2: Arduino ATmega 2560, image is taken from the Arduino homepage [21]

Printed circuit board - PCB

The robot has two PCBs in addition to the Arduino board. On top of the Arduino, the self–made PCB is connected. This PCB was produced by Jensen in 2017 [16], with an iterative design from Andersen and Rødseth 2016 [17]. It functions as an interface between the sensors and the Arduino. A Bluetooth Low Energy (BLE) communication device is fastened, and connected, to the PCB. It allows communication between the robot and the server. The self-made PCB from Jensen also distributes power from the battery. At the same time, the PCB interfaces with the motor driver.

The third PCB is a commercial, UK1122 H-Bridge motor driver [23], that powers the wheels motors. The technical specification states that a power source from 6 to 35 V DC can be connected, it is therefore connected directly to the battery. The motor driver can also supply 5 V DC to other components. Andersen and Rødseth concluded that the motor driver could give the system enough power while the Arduino board could not. Therefore, the self-developed PCB uses the Motor Driver as a 5 V source, Andersen and Rødseth [17, p. 75].

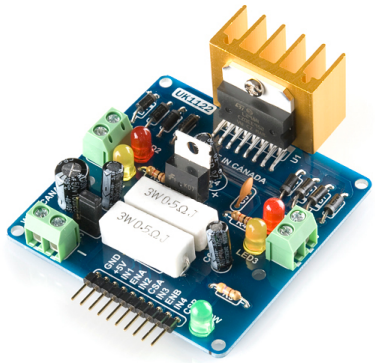


Figure 2.3: UK1122 H-Bridge Motor Driver, image is taken from [24]

Motor and encoder

The robot has two wheels, each driven by a separate motor. The motor was changed by Jensen in 2018 to DAGU with 120:1 gear ratio (DG01D, [25]), from DAGU 48:1 (image 2.4). The reason for this change, explained by Jensen [16, p. 6], was due to spin and slip, causing problems in the robot's estimated travel distance.

Figure 2.4: DAGU motor image from Sparkfun [26]

Data from two encoders are used to calculate the distance travelled. They are attached to the back of the motors. The encoders are from DAGU and consists of two neodymium 8-pole magnets and use the hall-effect [27]. They count the transition between the positive and negative poles, and travel with the same velocity as the motors. The encoders can operate with a voltage source from 3 V to 24 V, specified in the datasheet [27].

A SO5NF STD (see image 2.5) - servo is connected to the sensor tower, placed on top of the robot. The Arduino powers the servo with 3.3 V. The servo can rotate the sensor tower 90 degrees. By rotating the sensor tower, the robot can scan 360 degrees around itself. The sensor tower will only be rotated while the robot is standing still, and the tower remains at a fixed position when the robot is driving. The technical specification of the servo is found in the datasheet [28].

Figure 2.5: SO5NF STD servo, image is taken from DigiKey [29]

Light sensors

To detect the surroundings and measure the distance to obstacles the robot uses light sensors. The operation is based on emission and detection of a light beam. The sensor emits and detects light in the infrared range, meaning the light beam is not visible to the human eye. When the sensor detects the light beam, it gives a voltage output that corresponds to a distance.

The Arduino robot has two different setups to measure the distance to an obstacle, infrared (IR) sensors or lidar. These sensors are placed on top of the robot, on what is called the sensor tower (figure 2.6). The two different possible setups have resulted in two different source codes for the robot.

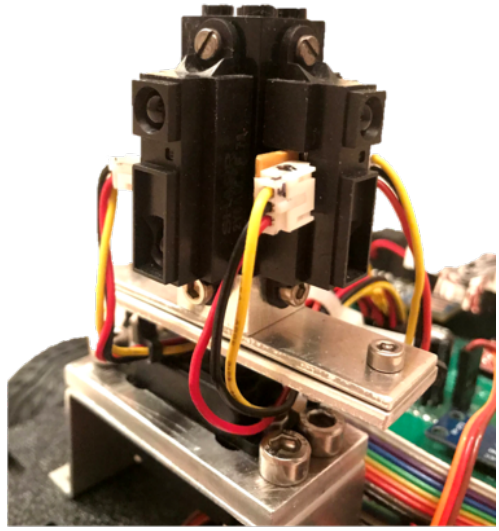


Figure 2.6: Sensor tower on top of the robot with the IR sensors

The standard configuration is using four IR sensors (*GP2Y0A21YK*) mounted orthogonal to each other. The IR sensors have a detection range of 10 to 80 centimetres [30]. Due to increasing error with larger distance, the server only uses measurements up to 40 centimetres to create the map. The IR sensor generates a voltage based on the distance to the object. The ADC on the Arduino then converts the analogue voltage into a digital value. Figure 2.7 shows a visual representation of how an ADC works. This is an example of a signal, represented by three bits.

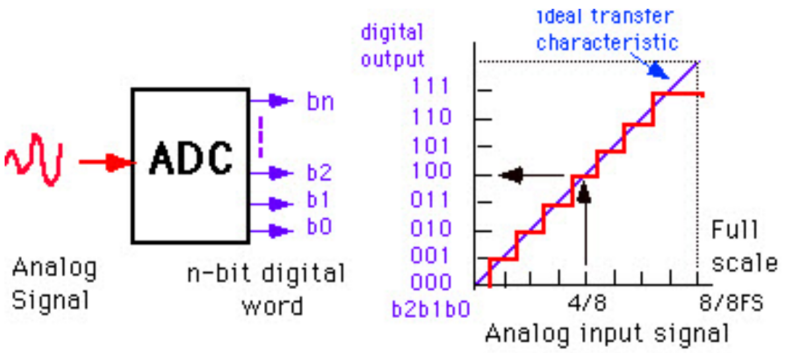


Figure 2.7: Analog to Digital converter, image taken from [31]

The robot can also use the *LIDAR lite v3* (figure 2.8), from Garmin, to detect obstacles. The datasheet specifies that the lidar has a measuring distance up to 40 meters with a 70% reflective target [32]. The lidar is placed on top of the sensor tower and replaces the forward heading IR sensor. How to mount and connect the lidar is described in appendix A.7.

Figure 2.8: The Garmin lidar v3 sensor, image from Sparkfun [33]

Heading and acceleration sensors

The Arduino robot has an inertial measurement unit (IMU), LSM6DS3, with a 3-axis digital accelerometer and a 3-axis digital gyroscope [34]. It is powered with 3.3 V from the Arduino board. The IMU is attached beneath the robot, towards the front. The accelerometer measures linear acceleration. In the software application of the robot, the accelerometer data is not utilised. Dypbukt 2020 [19] describes usage of the accelerometer to estimate the distance traveled but did not see an improvement and decided therefore not to use it [19, chapter 4.5].

The gyroscope on the other hand, measures angular velocity at each iteration of the robot application. It has a range of $\pm$ 125 dps (degrees per second). The gyroscope is only

used to measure rotation around the z-axis, as this is the only rotation relevant for controlling the robot. The raw gyroscope data and the angular rate sensitivity, (4.375 mdps/LSB) given from the datasheet [34], are multiplied together to obtain the measured value from the raw data. This means the raw data is converted to physical quantities before being used in the position estimation.

The robot also has a compass. It is fastened on top of the self-made PCB. The compass is a 3-Axis digital compass IC, HMC5883L type GY-273 [35]. A compass, or magnetometer, measures the strength and direction of the magnetic field. The magnetic field is produced from the earth and other, local magnetic sources, such as electric motors or magnets. The earths magnetic field can be said to be constant over the relevant timescales, even if it can have slight changes from day to day. The strength of earths magnetic field is more dependant on global position than point in time [36]. As the robot project has only been used in Trondheim, the earths magnetic field is expected to be constant. The GY-273 compass is designed for low–field magnetic sensing [35]. There exists drivers and application code for reading data from the compass, but this is presently not used in the application. Nilsen 2018 argues that the compass introduced more noise and did not clearly improve the heading estimation [37, p. 14]. Jensen 2018 on the other hand, states that the robot navigates considerably better when using the compass [18, p. 30].

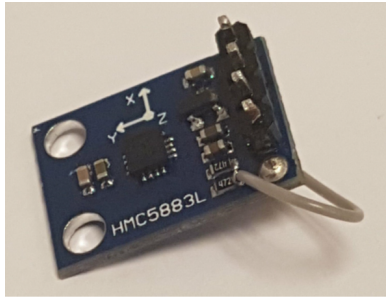


Figure 2.9: HMC5883L compass, image is taken from Jensen 2018 [18] after the repairs

Communication device

The server and robot communicates over Bluetooth Low Energy (BLE), using the nRF51 dongle from Nordic. The server, running on a host computer, is connected to the peripheral dongle (ID: 680316134). At the same time, the robot is connected to the server dongle (ID: 680840037). Both can receive and send messages with BLE. The distance between the server dongle and peripheral dongle can be up to 10 meters in an indoor environment [38]. The dongle is programmed with a hex file and flashed through nRFgo Studio, from Nordic Semiconductor.

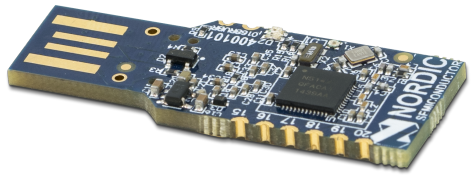


Figure 2.10: To communicate between the robot and server, both systems must have an nRF51 dongle (image is taken from [39])

Battery

The robot is powered by a Li-Ion Battery, *H2B181*. The battery is fastened to the chassis of the robot, in the rear. It is directly supplying the Arduino and the motor control board. The nominal capacity is 4.6 Ah, and the nominal voltage is 11.1 V [40]. How to charge the robot is explained in appendix A.4.

2.1.3 Software

To get a multi-threading system the robot uses FreeRTOS. FreeRTOS is a real-time operating system for microcontrollers [41], letting the ATmega2560 schedule different tasks. This allows the processor to switch between tasks even though the ATmega2560 is a single core processor, and can only handle one thread of execution at any instant of time.

The robot application has five key parts, initialisation of the sensors, and four real-time tasks. By having the initialisation of the sensors before starting FreeRTOS, errors may be detected before the robot starts driving.
The four main real-time tasks running on the ATmega2560 are:

I. vMainCommunicationTask

II. vMainSensorTowerTask

III. vMainPoseEstimatiorTask

IV. vMainPoseControllerTask

The communication task (I) has two main responsibilities. It is responsible for maintaining the connection to the server, as well as for sending and receiving data. Today the robot application works using Cartesian coordinates, which means the commands from the server are received in x and y coordinates, given in centimetres. The robot will also receive a handshake from the server. The robot will send messages to the server that contains the name of the robot, the size of the robot, the position and rotation of the robot, and measurement data from the sensor tower.

The sensor tower task (II) gathers measurements from the IR sensors and the LIDAR, and sends them to the server. In addition, it rotates the sensor tower using the servo.

The job of estimating the position and rotation is performed in the pose estimator task (III). The task uses data from the encoder and IMU to calculate its predicted Cartesian coordinates and the predicted robot heading. The robot uses dead reckoning [5] to estimate its POSE. Currently the accelerometer and compass are not used in the estimation task, while the encoders and gyroscope are used. In the calculation of the position and rotation of the robot, the gyroscope and encoders are weighted differently. If the rotation of the robot is less than 10 degrees per second, the gyro weight is 0. If the sensor value is over the threshold, then the gyro weight is 1. This is further explained by Dypbukt in [19, chapter 4.7.3].

The pose controller task (IV) is responsible for making the robot reach the right position. It uses a PI-controller, which receives the target coordinates from the communication task. The pose controller task sends the desired magnitude and direction of actuation to the motors, driving the wheels.

2.1.4 Programming with Atmel Studio

The robot application is written in C. It is developed and debugged with Atmel Studio 7. Since the robot application is running on an ATmega-based Arduino board, some modification must be done in Atmel Studio, to be able to program the robot. The instructions that were used for this installation was documented by Jensen in 2017 [16, chapter 6.1.2]. The following operations were done in Atmel Studio under *Tools* → *External Tools*.

- ”Title”:
  Write “Deploy code”. The title is the name that will be shown in Atmel Studio as the name for the operation.
- ”Command”:
  The path to where AVRdude.exe is installed.
- ”Argument”:
  -F -v -p atmega2560 -c wiring -P COM6 -b 115200 -D
  -U flash:w:”$(ProjectDir)Debug\Test.hex”:i
  -C ”C:\Program Files(x86)\Arduino\hardware\tools\avr\etc\avrdude.conf”
- “Use Output window”: Check this box

After the setup in Atmel Studio was executed, it was possible to flash the robot. First by building the code then pressing *Deploy code*, found under *Tools*. To give the reader a better understanding of the AVRDude commands [42], used for flashing from Atmel Studio, a short explanation is given below.

– *F*
  This operation overrides the signature check, in case the device signature is wrong.

– *v*
  Enable verbose output.

– *p atmega2560*
  Sets the command for which type of device is programmed.

– *c wiring*
  Sets the options to program the device over wiring, like USB.

– *P COM6*
  Sets the COM-port where the device is connected to the computer. To find the COM-port, check the device manager on the computer.

– *b 115200*
  Sets the communication speed over the data channel.

– *U flash:w:”$(ProjectDir)Debug\Test.hex”:i*
  A memory operation, to flash the device by reading the specified file from the computer, and writing it to the device memory. The final *:i* indicates the file-format, *Intel hex*.

– *C ”C:\Program Files (x86)\Arduino\hardware\tools\avr\etc\avrdude.conf”*
  Specifies the path to the AVRdude configuration file

2.2 Server application

The server used in this master thesis is the Java server. Most of the code is written by Thon 2016 [43]. The server is called System for Self-Navigating Autonomous Robots (SSNAR). The project-file last used by Dypbukt was used in the work on this thesis. The file is found on One Drive, access is given by Professor Tor Onshus. A full guide on how to use the server is found in appendix A.2.

The server application can operate in three modes, *manual*, *navigation* and *simulation*. In manual mode, the user will manually set the x- and y-coordinates. The coordinates will be sent to the robot and treated as a new target. The second mode is the navigation mode. In this mode the system uses Simultaneous Localization and Mapping (SLAM). SLAM is defined as a system that can continuously construct a map of the environment while the robot calculates its position and orientation in the environment [44]. This allows the system to recognize previously visited coordinates, compared to odometry, where the system does not know when an already known location is seen again. A visual representation of this is shown in figure 2.11.

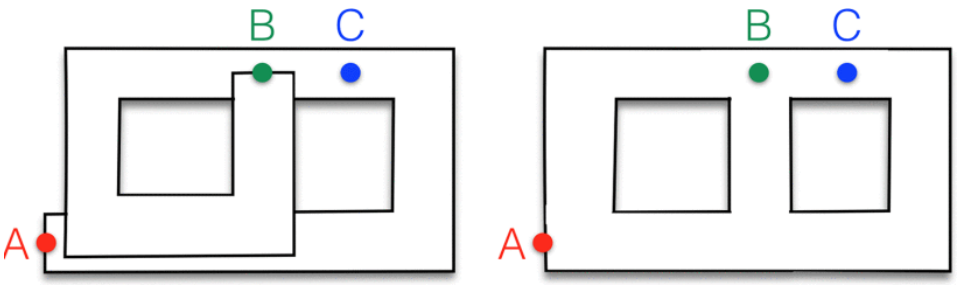


Figure 2.11: Points A and B are detected as a previously visited points in SLAM. With the use of odometry, this information is lost, image from [44]

In the navigation mode, the server will calculate new targets for the robot. This calculation is based on the mapping data received from the robot. In both modes, the server will receive the robots position, heading, and measurements from the sensor tower. The server uses this data to build up a map that is represented graphically. The map consists of a grid of cells, the cells are marked as either cleared, restricted, or weakly restricted. A clear cell will be white indicating that there are no obstacles there. A restricted cell will be black and mark that there is an obstacle in that position. A weakly restricted cell will be dark grey and is considered a danger-zone for the robot, because they are close to restricted cells meaning a collision is probable there. The third mode is the simulator mode, which can simulate how a virtual robot will navigate a virtual court. The simulator feature will not be used in the work behind this thesis.

2.3 Tracking the robot

To be able to validate the navigation performance of the robot, a solution from Optitrack is used. This system is found at NTNU room B333 [10]. The concept is to track the robot when it is moving in a specific area. This allows post-analysis of the robot's movement. Running the robot in manual mode, it is possible to compare the data from where the robot was supposed to travel (the given command), to where it actually travelled (Optitrack measurement). If the robot was in navigation mode, it is only possible to see where the robot has driven, as the commands are typically not known.

The Optitrack system consists of 16 cameras mounted on the ceiling, in the shape of a square around the testing area. *Motive*, witch is the accompanying software system, is installed on the desktop computer in B333. Motive takes multiple 2D-images and trajectorises them to a 3D-image, also known as *structure from motion*. By having several 2D-points, tracked between images taken from different cameras, a 3D-image can be constructed [45]. To be able to do this, the cameras must be calibrated. This means to compute the position, orientation and distortion of each camera, as explained in the documentation, chapter *System setup* $\rightarrow$ *Calibration* [46]. In this master thesis a series of steps were executed when using the tracking software. This is found in the appendix A.6.

2.4 Initial work

To ensure that the robot system worked as intended, and as described in previous reports, some initial tests were done. This included communicating with the robot, programming the robot, checking that the robot application coincides with what was written in previous reports, and in the end, testing the robot on the office floor. During this time, some problems were discovered. Some of these problems were fixed, and some are not important for this master thesis.

2.4.1 Challenges

Connection with the charger
Initially the robot would not charge, it was thought that there was a short-circuit in the electrical wiring. No error was found when probing the robot with a multimeter. Securing the chargers connection with two crocodile clips solved the problem.

Atmel Studio on laptop

Both a desktop computer and a laptop has been used in the master thesis. The laptop was brought to the testing facilities to run the server, and have the possibility of reprogramming the robot on-site.

It was desirable to program the robot on the testing site, using Atmel Studio. The laptop available fulfilled the requirements needed to install Atmel Studio [47]. After installation, it did not work. After troubleshooting, using debug steps from Atmel studio [48], and contact with Atmel support team, the problem was not resolved.

To be able to work with the robot and not be delayed further in the project, another way to program the robot on-site was found. The solution was to flash the robot application hex-file generated by the desktop computer, using *AVRdude* on the laptop. To make the process as efficient as possible, a batch script that runs the AVRdude command was written. This script is found in appendix A.3.

Troubleshooting the Arduino board

Another problem was discovered when trying to program the robot. After following Jensen [16, chapter 6.1.2], Atmel Studio gave an error with the note "stk500v2_ReceiveMessage(): timeout". To troubleshoot, the example code of a blinking light in Arduino IDE, was attempted flashed on the Arduino. It was seen that the receiver (RX) led on the Arduino blinked, but not the transmitter (TX). This confirmed the suspicion that there was something wrong with the Arduino. Probably a bug in the bootloader, since the development program could be read but did not have two-ways communication. No further investigation has been done into troubleshooting the Arduino board. The solution was to replace the Arduino. A second Arduino was tested, first with the simple blinking light example from Arduino IDE, then with the complete robot application. Both tests were successful.

New motors

When first testing the robot, the robot was not able to drive straight forward. The software application was the same as the on used by Dypbukt 2020 [19], and he described nothing of this sort. A hypothesis was that there where something wrong with the hardware. More specifically, something was wrong with the motors.

To check the motors, both were removed from the robot, then tested individually with the use of a power supply and a multimeter. The left motor had no problem spinning with the help of the power supply. On the other hand, nothing happened when running the right motor, even when the voltage was increased. With the use of the multimeter, connected in series between the power supply and the motor, it was found that no current flowed through the motor. By Ohm's law it is known that current equals voltage divided by resistance. This implies that the resistance must be infinitely high, or relatively much larger than the voltage, to get the current low enough to be shown, by the multimeter, as zero. Following this, the conclusion was that the motor was broken and had to be replaced. At this time, new components had been ordered for the other students working on their robots. Eivind Jølsgard had already made the order, but had a set of unused, spare motors. It was decided to replace the old, broken motors with the ones provided by Jølsgard.

The new motors are of the type Uxcell DC 12 V 220 RPM with encoder and gear [49]. The swap between the DAUG motors and the Uxcell 12 V motors was not without complications. The old motors were fastened so that the primary axis was in parallel with the side of the robot, whereas the new motors had to be pointed inwards. This made it challenging to place the motors, without getting in the way of already placed electronics inside the robot. The old mounting brackets could not be used either. With the help of the Mechanical lab at the cybernetic department, the new motors were attached to the robot.

The new motors were successfully attached to the robot and connected to the motor control card. Image 2.12 shows the new placement of the motors, fastened inside the robot.

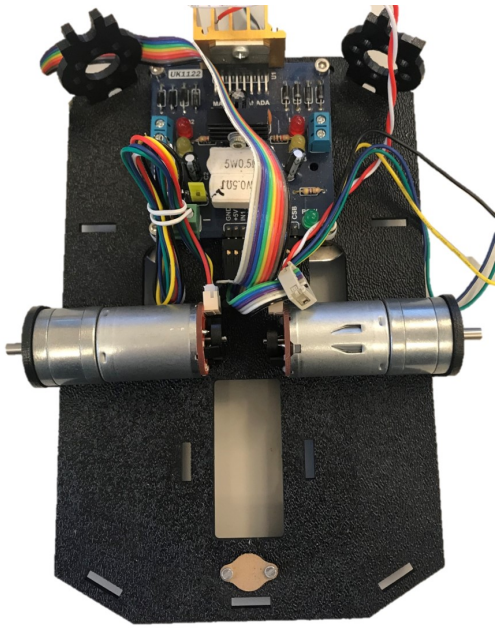


Figure 2.12: Placement of the new motors

New encoders

The old encoders were not longer needed with the new motors, because they have encoders pre-installed. The new sensors are quadrature encoders, in contrast to the hall effect encoders that were previously used. The new encoders have two outputs, meaning they can be used to detect, not only rotational speed, but also the direction of rotation. The old encoders operated on 5 V [27], compared to the new encoders that operates on 3.3 V, see 2.13.

Figure 2.13: Connector sheet for the new motor and encoder, image is from [49]

A modification was done to the self-made PCB from Jensen 2018 (seen in figure 2.14), to power the encoders with 3.3 V. Removing the trace that powered the encoder connection with 5 V and soldering a wire from a 3.3 V source (see figure 2.15), solved the power source problem. Another situation was that the new encoders have two data wires, one for each of the outputs. As the self-made PCB from Jensen was made for the old encoders, it was only possible to use one of the phases from the new encoders. If both phases had been used, it could improve the accuracy in measured rotation, see section 3.3.3.

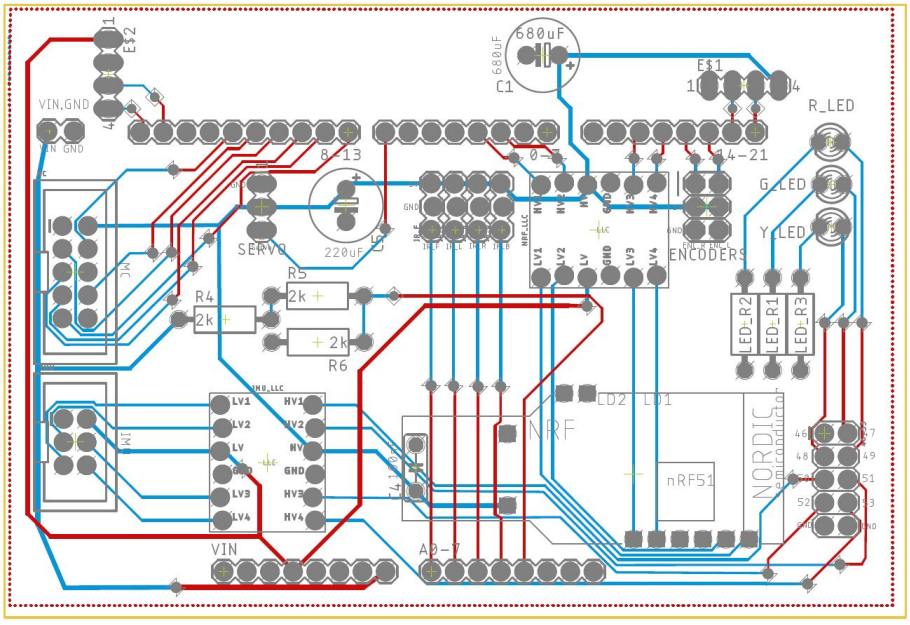


Figure 2.14: Original PCB-design from Jensen [18]

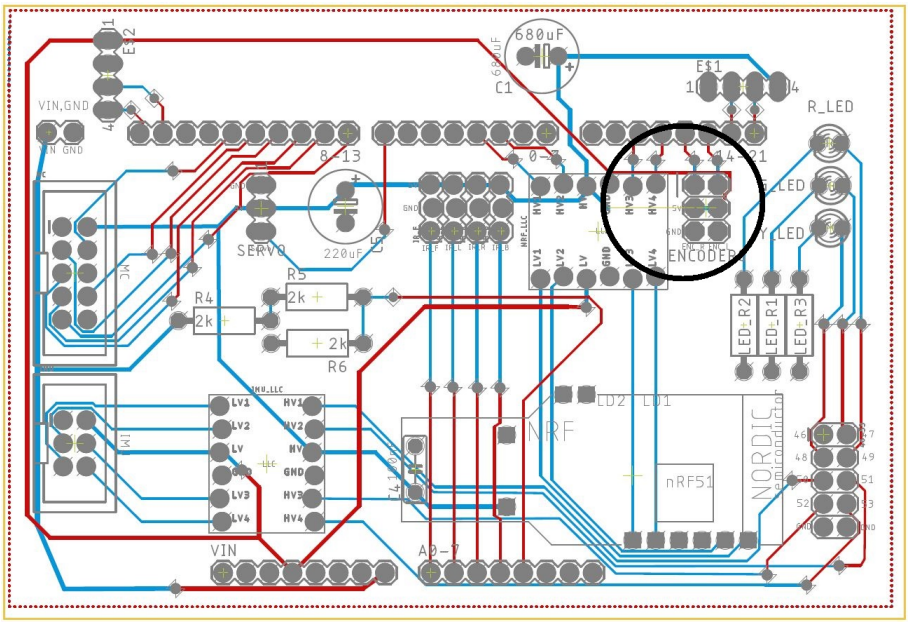


Figure 2.15: Modified design for the PCB, where the changes are made in the area indicated by the black circle

The new motors and encoders are expected to have a different wheel factor than the old setup. *Wheel factor* is defined here as a ratio between the wheel circumference and the number of encoder ticks per rotation. Previously, this has been calculated by using the number of encoder ticks and the motor's gearbox ratio. The new motor had no complete datasheet and therefore no such way to find the gear ratio. The encoders ticks per revolution will be determined by executing two tests. This method is described in chapter 3.3.3.

New servo

Another problem was detected when initially testing the robot. The sensor tower did not rotate. The issue was debugged by validating the level of input voltage and control signal, using an oscilloscope. It was found that the power signal was 5V, as it should be [28], and the control signal seemed plausible. It was concluded that the servo did not work. The servo was therefore replaced by a new servo of the same type [28], which fixed the problem.
This change also resulted in reconnecting the IR sensor. It was then discovered that the IR sensor's hardware connection did not match the connection in the robot application. This mistake was fixed and verified by reading one IR sensor at a time. The new mapping from hardware to software is described in the robot application and is found in table 2.1. Where the *Pin* is the connection to the Arduino board, *Analog In* are the connections to the sensor on the self-made PCB. It is worth noticing that the server receives the measured distance in an array of the following order [front, left, back, right].

	Pin	Analog In
Left	PINF0	A0
Front	PINF1	A1
Right	PINF2	A2
Rear	PINF3	A3

Table 2.1: New IR sensor mapping

FreeRTOS and Lidar

It is said that the most challenging thing in a relationship is communication. The same was initially true for the Arduino robot. When the robot ran with only IR sensors, no errors were detected. An issue appeared when the lidar was used. The error was provoked by a timeout between the server and the robot. The robot was tested with the lidar sensor in the original software structure, resulting in the robot timing-out after a period. It is believed that the robot sends more data over the BLE interface than the server can manage [19, chapter 5.2]. After troubleshooting, the conclusion was that the lidar and FreeRTOS might be the problem. Another possibility is what Jensen discussed in 2018 [18], that the lidar has to be a critical task to be handled in FreeRTOS. After discussing the problem with Professor Tor Onshus, it was decided not to use the lidar, but instead focus on the setup of the IR-sensors. This desertion did not compromise objective of the thesis, as the robot should be able to detect obstacles using only IR sensors.

2.4.2 Software application changes

During the start of this project, it was discovered that the distributed software application and the master project from Dypbukt had some differences. It was decided to change the robot application to match the work documented in Dypbukt's master thesis.

Changes that were made in the robot application:

- The gyro weight was set from one to zero, in the *if statement*, that checks if the robot turns less then 10 dps, in the pose estimation. This causes only the gyro measurement to be used to calculate the robot heading, during turning, as described in [19].
- The error factor described in Dypbukt 2020 [19], chapter 6.1 and 6.2, was not implemented in the source application for the IR setup.
- At the end of the pose estimation, *predicted theta* was converted from radians to degrees.

The problem with converting the predicted theta into degrees, was that the measured data was extracted in radians. This mismatch of units meant that adding them together, in the integration, resulted in the value of predicted theta being wrong. The value was saturated to be in the range $\pm 2\pi$, which made the error harder to detect. An example with the gyro measurement is found below. Note that only the relevant code is shown.

```
float gyrZ = 0;
float predictedTheta_gyro = 0;

while(1){
    // Import gyro measurement
    gyrZ = (fIMU_readFloatGyroZ() - gyroOffset); // [dps]

    // Find the angle measured by the gyroscope since previous iteration
    dTheta_gyro = gyrZ*period_in_S* DEG2RAD; // [rad]

    // Add the new gyroscope angle to previous angle
    predictedTheta_gyro += dTheta_gyro; // Mismatch between deg and rad!

    // Saturate measurement values to be +- 2 pi
    vFunc_Inf2pi(&predictedTheta_gyro); // [rad]

    // Calculate the predicted heading of the gyroscope in degree
    predictedTheta_gyro *= RAD2DEG; // [deg]
}
```

2.4.3 Merging IR and lidar application

In the code base submitted by Dypbukt it was found that there existed two software applications, one for the IR setup and another for the lidar setup. From experience, it is not easy to maintain two different versions of the same software project. It was considered beneficial for the robot project to have the robot applications merged together. It would be easier for future students to work with one project version, instead of keeping track of the two similar, but different, versions. The differences in the project can be summarised as follows:

1. The robot names and robot name lengths were different.
2. The calibration matrix, which transforms IR measurement to centimetres was different.
3. In server_communication.c the modes have different lengths of the *arq* message.
4. In *SensorTower.c* the robot has different wait times.
5. In *SensorTower.c* at the lidar application, the forward IR measurement is replaced by the measurement from the lidar.
6. In *main.c* file in the lidar application, the lidar is initialised.
7. In the lidar application, *PoseController.c* the global variable gTheta_target is integrated and calculated from radians to degrees.

These small changes in the two robot applications were easy to merge. If the robot only uses the IR sensor, the right robot application will be used by commenting in ***ROBOT_IR*** in *define.h.* By using ***ROBOT_LIDAR***, the robot accesses the lidar application.

2.4.4 Folder structure

When first opening the project file in Atmel studio, it was noted that all 63 files were in the same folder. At the same time, the *main.c* file had 1220 lines of code.

Since this is a student project, the software development process and the resulting structure should be of high priority, to avoid unnecessary bugs and excessive work. A folder structure would make it easier to maintain the robot application. Therefore, the robot application was structured into three folders, and the main file was shortened. The folders got the names *Drivers*, *FreeRTOS* and *Tasks*. In *Drivers* all files that are written by students to operate the robot, and interface with other systems are found. In *FreeRTOS* all files provided by FreeRTOS are saved. In *Tasks*, the four main tasks (described in 2.1.3) that the robot executes, are kept. By having a folder structure, it is easier to get a quick overview of the robot application. As a result of the refactoring, the *main.c* file has now 250 lines of code. The robot was tested at the floor of the office to validate that the folders did not affect the robot's functionalities. It was only tested with the IR sensor setup because of the problems with the lidar, described in 2.4.1. No error was detected with the IR setup.

2.4.5 Testing autonomous docking

Former students have worked on how to dock the robot automatically. This task demands that the robot drives backwards into a docking station, after mapping an environment. Through this master thesis the autonomous docking has been tested as a side project, to see how far the development has come. The docking station can also double as charging bay. It is built up of three wooden planks, where there are two metal strips on the back wall. These metal plates can be connected to the robot charger, and the robot can be powered by having it's metal springs touch the strips on the back wall. The autonomous docking application was tested in the round court, with the docking station inside (figure 2.16).

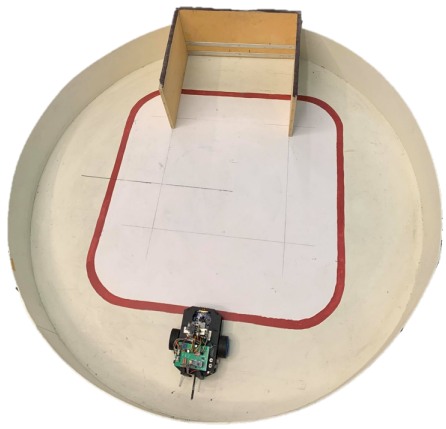
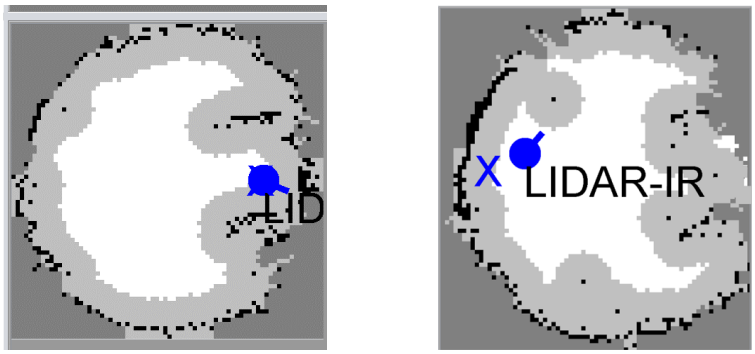


Figure 2.16: Test setup of the autonomous docking station

Some modification were done in the Java server application as well. In the robot task manager the variables *IsGoingHome* and *setDock* are set to true. A fixed coordinate for the docking station was also set in the server file *Robot*. The coordinates were set to x equal 95 and y equal to zero. These coordinates correspond to the docking station's placement in the round court, relative to the robot's initial position. The searching algorithm to find a path to the docking station was added back into the server. This searching algorithm is based on the A* algorithm and is found in the *Robot Task manager* file.

The system was debugged by using the server to print the state of the robot. If the robot had mapped the area and were going to drive to the docking station, the server would print "Robot is going home". After a lot of testing, it was seen that the server was not able to calculate a path for the robot to go home and that the navigation was not accurate enough. The only time the robot successfully docked is seen in figure 2.17a. In figure 2.17b, the robot was not able to dock autonomously. The server printed that the robot was going home, but no path was found. This problem has not been investigated further but shows that the robot can have an autonomous docking feature in the future.



(a) Successfully driving to the docking station **(b)** Unsuccessfully driving to the docking station

Figure 2.17: Different result from testing the autonomous docking application

When working on this side-project, some new ideas have surfaced. If the robot docks into the charger, to execute an autonomous charging cycle, the robot has to reverse into the docking station. The second idea that came to mind is to replace the manual switch on the robot with a diode. The robot can then automatically start charging when the metal springs touches the back of the docking station. Today a switch must be manually switched on to close the circuit from the charger to the battery (Switch 1 in figure 2.18). By replacing Switch 1 with a diode, shown in figure 2.19, the current will be directed from the charger to the battery, when the charger is connected. This solution removes the need for human intervention during docking, while still avoiding potential short circuits when the robot is driving.

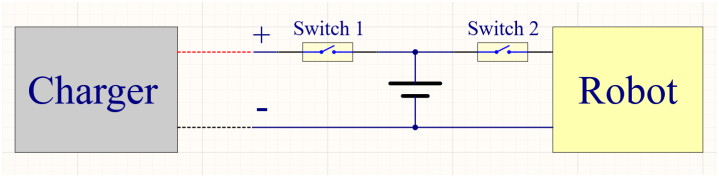


Figure 2.18: Simplified circuit diagram of the charging system for the robot

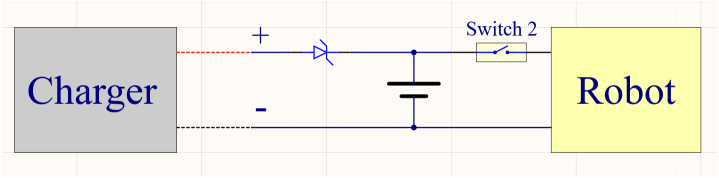


Figure 2.19: Simplified circuit diagram of the charging system for the robot with the switch replaced by a diode

Chapter 3

Method

Sources of error in the position estimate will be found by first looking at the system on a macro level (driving and navigation), before breaking down the system and ending on the system's micro-level (the sensors). First, the robots' driving performance will be looked at to determine how the position error develops over time, through use of the square test. The navigation performance will be tested in different testing courts. This test is described in section 3.1. An evaluation of the estimator module in the robot application is described in 3.2. After this, the robot's sensors are reviewed, in section 3.3, to see if there are inaccuracies in the sensors, that can propagate through the system. In the review of the sensors, a way to improve the sensor data is also described.

Limitations

Some constraints have been set on the evaluation of the robots position estimation and investigation into its sources of error. The task is limited to work on the robot's sensors and evaluating the position heading in the estimation. The other software modules, for example the *control module*, is left out. An analysis of the hardware construction of the robot, for example looking at the rubber of the wheels on different surfaces, is also left out. Suggestions to improve the position estimator will be given as theoretical ideas and an initial design.

3.1 Driving performance

3.1.1 Square test

The square test is a position test that is widely used in previous robot projects. The test is explained in detail in Jensen 2017. The point of the test is to objectively analyze the driving performance of the robot. This can be used to indirectly validate the internal po-

sition estimate. The test is done by having the robot drive one meter forward, then rotate 90 degrees and repeat this action, until the robot has driven in a square. The square will be driven in the clockwise (CW) and counterclockwise (CCW) directions. The robot will be monitored by the Optitrack Motion system. Using the post analysis script made by Halvorsen [1], the data will be processed and visualized. The robot is set to manual mode and will get commands from the operator. The commands are found in table 3.1.

	CCW		CW	
Turn	x	y	x	y
1	100	0	100	0
2	100	100	100	-100
3	0	100	0	-100
4	0	0	0	0

Table 3.1: Commands to execute the square tests in the CCW and CW directions

In Dypbukt 2020 [19], an error compensation factor for the gyroscope and the encoder was implemented. Dypbukt argued for implementing the error factors, the reader can find this in Dypbukt 2020 [19, p. 36-40]. The square test method aims to find the setup of the robot application, that gives the most accurate position. The square test method will be executed two times with each setup, once with encoder error factor, gyro error factor, and no error factor. Each setup will also be tested in both the CCW and CW direction. The setup with the lowest absolute value of the error distance will be tested further in the continuous square test.

To evaluate the position performance, the error distance is found through post analysis in Matlab. Every square run will be plotted. The distance error will be calculated by using equation 3.1, where x_{tar} and y_{tar} are the corner targets shown in table 3.1. These coordinates are subtracted from the position coordinates logged by Optitrack, x_{real}, y_{real}. The error distance is defined as the Pythagorean distance between the target and actual point.

$$d = \sqrt{(x_{real} - x_{tar})^2 + (y_{real} - y_{tar})^2} \tag{3.1}$$

3.1.2 Continuous square test

Another version of the square test will be the continuous square test. The reason for executing this test is to find a tendency in the position estimate over time. The execution of this test is similar to the square test, with the exception that this test will consist of the robot driving multiple squares consecutively. The target is to drive four to five rounds. The amount is chosen to gather enough data for predicting a trend in the accuracy of the position. The continuous square tests will be done in the CW and CCW directions, the same as for the square test. The test will be executed five times per direction.
To quantitatively find the position error, the distance error for every turn will be found. A

comparison of the precision for each round can then be discussed. The average distance error to each corner will be calculated to find the tendency over time.

3.1.3 The round court

To be able to navigate in any environment, the robot must be able to detect different shapes. This means that the robot can not detect only straight surfaces. A circular court will be used to test the navigation performance of the robot with sloped walls. The round court is 150 centimetres in diameter. It has a wooden floor and walls, all painted white. The robot will be placed in the court approximately 15 centimetres from the wall. The robot will then be connected to the server with the initial position at the origin. The server navigation will be used by clicking on the start button in the server GUI. A test is seen as successful if the robot navigates the court without crashes, while producing a map of the court. If the robot is successful, the docking station will also be placed in the round court (see figure 3.1) to see the result of the navigation.

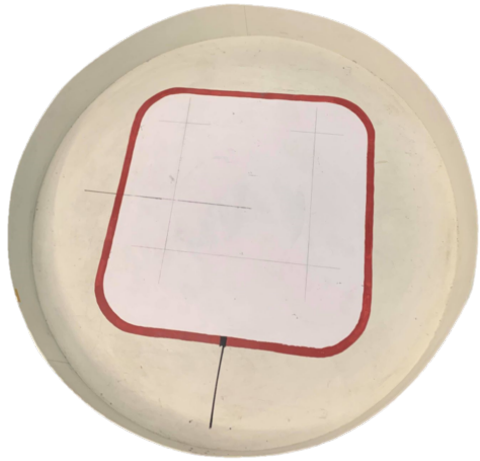


Figure 3.1: The round court

3.1.4 Larger tracking court

This method has the objective of determining the precision of the whole system. If the robot has an accurate position estimate and properly detects the objects, the server's GUI map should be a clear birds-eye view of the environment.

The course will be built up of hobby boards. The boards are 22 centimetres wide and will work as a 22-centimetre tall wall. The walls are tall enough for the IR sensors to detect them.

The test track will consist of both straight and sloped walls. No incisions can be less than 30 centimetres wide. The reason for this is restrictions from the server, marking areas of 15 cm from the wall as slightly restricted, meaning the robot will not explore this area. A sketch of the testing court is found in figure 3.2. This test will be executed two times to see the robot's driving performance. The red **X** in the figure marks the starting position. The initial position will be set as the origin, and the robot will operate in navigation mode with the server.

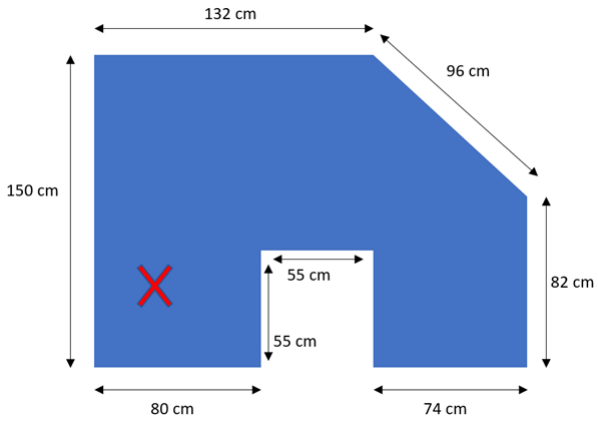


Figure 3.2: Plan for a larger court for the robot to navigate

3.2 Position estimation

After validating the sensors on the robot, the processing of sensor data can lead to errors. Therefore, it has been chosen to look closer at the position estimator implementation in the robot application. The following is a pseudo-code representation of the algorithm found in the *PositionEstimation.c* file.

```
void positionEstimation{
    if !Handshook{
        Calculate the gyroOffset
    }
    if Handshook{
        Get the global left and right wheel ticks
        Calculate the number of encoder ticks since last sample
        Find the distance the robot has traveled since last sample
        Find the angle from the encoders
        Extracted gyro data and subtract the offset
        if gyrZ < 10 deg per sec {
            gyro weight is 0
        }
        if gyrZ > 10 deg per sec{
            gyro weight is one
        }
```

```
17          Scale the gyro measurement from deg to rad
18          Integrate gyrZ
19          Fuse the sensor data to the predicted heading in this sample:
20              dTheta = (1-gyroweight)*dTheta_enc + gyroweight*gyrZ
21          Calculate the predicted x and y coordinates
22          Integrate the predicted heading of the robot
23          Update pose
24      }
25  }
```

In this chapter, a description of the method for evaluating the position estimation software is found. Firstly, a closer look at *dTheta*, the change in heading. Evaluation of the robot heading and a theoretical design, fusing all the sensors with an Extended Kalman Filter (EKF), is found at the end of the chapter.

3.2.1 dTheta

The robot runs the position estimation as a task in *FreeRTOS*. Each time a task is executed, there is a fixed time since the start of the previous iteration. This difference is used to integrate the angular velocity of the robot. In this method, the robot can use gyroscope data and encoder data to estimate it's position and rotation. *dTheta* is the name of the variable in the robot application, which is a product of the fusion of encoder and gyroscope data. It is the rate of change in the heading of the robot. By integrating *dTheta*, an estimate of the robot heading can be found. In previous master theses, gyroscope data and encoder data are weighted to calculate *dTheta* using the following equation:

$$\text{dTheta} = (1 - \text{gyroWeight}) \cdot \text{dTheta\_encoder} + \text{gyroWeight} \cdot \text{dTheta\_gyro} \tag{3.2}$$

The variable *gyroWeight* has previously been set dynamically to 1 or 0, depending on whether the robot is believed to be rotating or standing still, respectively. Since predicted robot heading is directly calculated from *dTheta*, it is desirable to see how *dTheta* develops over time. As well as what impact gyroscope data and encoder data has on *dTheta*. Experimental tests are planned to map out the development of *dTheta*.

The method is based on selecting different *gyroWeight* numbers so that the two sensors are weighted differently, adjusting their influence. The gyro weight number should vary from zero to one to accommodate the full range of combinations. First, it was natural to pick zero and one, which means only testing the impact from one sensor at the time. Zero means only testing the encoder, and one only using the gyroscope. After this the 50 – 50 combination of the gyroscope and encoder value is a logical next step. Where the sensors evenly contributes to the calculation of *dTheta*. To get more data points and a lower step size, values evenly distributed between zero and one are also selected. Based on the preceding arguments the following *gyroWeight* values were selected: 0, 0.2, 0.4, 0.5, 0.6, 0.8 and 1.

dTheta during driving and standing still

The sensors can be affected by the robot standing still, or by it moving. Therefore the method will include both the robot standing still and moving. To answer what *dTheta* is when the robot is not moving, the robot will be powered but standing still for five minutes. The *dTheta* value will then be printed and saved for post-processing. The test will be executed two times for each selection of the gyro weight. The choice of two tests was considered a trade-off between time and certainty. Two tests allow a repeat of the test if it seems that either result is an outlier. The same procedure is followed when the *dTheta* is evaluated when the robot is driving. The robot will be driving forward in manual mode, with the distance set to one meter. The number of tests and the choice of gyro value will be the same as for the test where the robot is standing still.

3.2.2 Robot heading

In further investigation of the internal position estimation, the orientation of the robot needs to be evaluated. The robot heading value is called *predicted_theta*. The heading is calculated from *dTheta*, as explained previously, and is sent to the server. The robot will be tested when standing still and when driving.The gyro weight has a direct influence on the calculation of the robot heading. The choice of gyro weight is selected to give a variation from the whole spectrum from 0 to 1. The choice fell on the following gyro weights: 0, 0.2, 0.4, 0.5, 0.6, 0.8 and 1, based on the same argument as in 3.2.1.

Standing still

The first test will examine the robot's heading for five minutes, when it is stationary, and observe the development. The goal is to see how much the gyro weight influences the heading and how it develops over time. The number of tests and the duration is a trade-off to get efficiency in the testing. It is also not likely that the robot will stand completely still for more than five minutes during navigation of an area. The method is similar to what is described in 3.2.1. The *gyroWeight* value is identical and selected on the same basis as for previous tests. The robot will be standing still, and the heading will be extracted. The test will be executed two times for each gyro weight and plotted in Matlab.

Heading in straight driving

The second test evaluates the robot's ability to drive longer distances in a straight line. The ideal test case, will be to have the robot driving for 10 to 15 meters, or more, to be able to validate the robots ability more accurately. With this distance, it is impossible to execute the test with the objective oversight of Optitrack at B333, because the room is too small. This puts a limit on how far the robot can travel in a straight line, during the test. Due to this limitation, the distance will be three meters. With the data from Optitrack, and the extracted robot heading, post-processing can be executed to analyze the data in Matlab.

The robot will have different gyro weight values to evaluate the driving performance, chosen identically to the previous tests. The test will be executed according to the following method:
The robot will be placed on a marked starting line. It is connected to the server and set in manual mode with the initial position in origin and no rotation (x: 0, y: 0, rotation: 0). The robot will then be commanded to drive three meters straight ahead, by entering x = 300 and y = 0 in the server.

3.2.3 New position estimator design

In the robot application today, only the gyro measurement and the encoder value are used to calculate the robot position and orientation. Utilising all sensors, that can be used to estimate the POSE, can in theory improve the position estimation. A suggestion is to implement an extended Kalman filter (EKF). The reason for selecting the extended Kalman filter over an ordinary Kalman filter is that the ordinary Kalman filter must have zero mean Gaussian noise and assumes a linear model [50]. The robot motion will be non-linear, due to sine and cosine inputs. Therefore the extended Kalman filter can be used, with a linearization of the system. The EKF deign is based on the approach from [4].

Firstly the state must be defined. The robot's wheel do not allow for sideways movement to reach a target destination. It is constrained to a set of paths to reach the target configuration and is therefore a nonholonomic system [51] [52]. To model the robots movement, the state can be defined as shown in equation 3.3. Where x and y are coordinates and θ is the robot heading, while V is the velocity in the heading direction, and ω is the angular velocity around the z-axis.

$$\beta = \begin{bmatrix} x \\ y \\ \theta \\ V \\ \omega \end{bmatrix} \tag{3.3}$$

To determine the robot's position and orientation, a reference from the theoretical world frame to the robot has to be established, where the robot position and orientation is represented by x, y and θ in robot frame. A rotation matrix with angle θ, seen in equation 3.4, can describe the relation.

$$R(\theta) = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.4}$$

The robot can be controlled by linear velocity, V, and angular velocity ω. The robot's

following motion can be calculated with respect to the world frame p_{world}.

$$\dot{p}_{world} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} \tag{3.5}$$

$$\dot{p}_{world} = R(\theta)^{-1} \begin{bmatrix} V \\ 0 \\ \omega \end{bmatrix} \tag{3.6}$$

$$\dot{p}_{world} = \begin{bmatrix} V\cos\theta \\ V\sin\theta \\ \omega \end{bmatrix} \tag{3.7}$$

This can be used to redefine the state equation 3.3 into what is shown in 3.8.

$$\dot{\beta} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \\ \dot{V} \\ \dot{\omega} \end{bmatrix} = \begin{bmatrix} V\cos\theta \\ V\sin\theta \\ \omega \\ 0 \\ 0 \end{bmatrix} \tag{3.8}$$

The system has to be discretized to be used in an EKF. The model is based on the sampling time Δt. The notation for the sample at time t is subscript k, while the previous sample uses subscript $k-1$. The model is discretized, and the result is as follows.

$$\beta_k = \begin{bmatrix} x_k \\ y_k \\ \theta_k \\ V_k \\ \omega_k \end{bmatrix} = \begin{bmatrix} x_{k-1} + V_{k-1}\Delta t\cos\theta_{k-1} \\ y_{k-1} + V_{k-1}\Delta t\sin\theta_{k-1} \\ \theta_{k-1} + \omega_{k-1}\Delta t \\ V_{k-1} \\ \omega_{k-1} \end{bmatrix} \tag{3.9}$$

The nonlinear function is called f for the system model and h for the measurement model. w_{k-1} is the system noise and v_k is the measuring noise [53].

$$\beta_k = f(\beta_{k-1}) + w_{k-1} \tag{3.10}$$

$$z_k = h(\beta_k) + v_k \tag{3.11}$$

$$F_{k-1} = \frac{\partial f(\beta)}{\partial(\beta)}|_{\beta^*_{k-1}} \tag{3.12}$$

To use the EKF, both the measurement and system models have to be designed and linearized, when the model is non-linear. The linearization can be done with the calculation of the Jacobian around β_k. Using this, the following linearization is found.

$$F_k = \begin{bmatrix} 1 & 0 & -\Delta t\sin\theta_k & \Delta t\cos\theta_k & 0 \\ 0 & 1 & \Delta t\cos\theta_k & \Delta t\sin\theta_k & 0 \\ 0 & 0 & 1 & 0 & \Delta t \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.13}$$

The measurement model is based on the measurements from each sensor on the robot. Individually they are found as:

$$H_{gyroscope} = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 \end{bmatrix}^T \tag{3.14}$$

by integrating the gyroscope measurement:

$$H_{heading} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \end{bmatrix}^T \tag{3.15}$$

The two encoders can measure linear acceleration by counting the encoder ticks between subsequent iterations of the FreeRTOS task.

$$H_{encoder} = \begin{bmatrix} 0 & 0 & 0 & 1 & l/2 \\ 0 & 0 & 0 & 1 & -l/2 \end{bmatrix}^T \tag{3.16}$$

Where l is the distance between the wheels.
By integrating the acceleration measurement, it is possible to find the velocity.

$$H_{accelerometer} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 \end{bmatrix}^T \tag{3.17}$$

Using the compass, the global heading can be measured.

$$H_{compass} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \end{bmatrix}^T \tag{3.18}$$

With this, the following algorithm can be used to predict and update the extended Kalman filter.

$$\beta*_k^- = f(\beta*_{t-1}^+, u_k) \tag{3.19}$$
$$P_k^- = F_k P_{t-1} F_k^t + Q_k \tag{3.20}$$
$$K_k = P_k^- H_k^T (H_k P_k^- H_k^T + R_k)^{-1} \tag{3.21}$$
$$\beta*_k^+ = \beta_k^- + K_k(z_k - h(\beta*_k^-)) \tag{3.22}$$
$$P_k^+ = P_k^- - K_k H_k P_k^- \tag{3.23}$$

Where H is equal to $[H_{heading}\ H_{encoder}\ H_{accelerometer}\ H_{compass}]$. Q and R are the covariance matrices for the process and measurement noise, respectively [54].

3.3 Sensors

Today the Arduino robot uses only encoders and a gyroscope to calculate the internal estimate of its position and orientation. Earlier theses disagree on whether the compass improved the position estimate, or if it actually made it worse. Therefore, the compass will be tested here. The accelerometer will be tested as well. Finally, methods for testing and calibrating the IR sensors are described, this is done so that the robot can find the precise location of obstacles in it's environment.

3.3.1 Gyroscope

The gyroscope in the IMU is used to measure angular velocity, in other words, speed of rotation. The sensor is mounted under the front of the robot, it gives its output in degrees per second (dps). An offset value is subtracted from the gyroscope measurement before it is used in the *positionEstimation.c* file. The offset value is found before the robot connects to the server. The offset is an average of 300 measurements when the robot is standing still. This is the method recommended by Sparkfun, the distributor of the IMU [55].

The gyro data is stored in the variable called *gyrZ*, in *positionEstimation.c*. This value is multiplied by a fixed time step, which is the time since the last iteration of the position estimation task. Resulting in an integration of the velocity, which gives the robot heading.

Drift and noise

Most, if not all, sensors can be subject to bias and noise. A gyroscope can also drift over time. This will be tested by leaving the robot on for 30 minutes, connected to the server over BLE or by using the USB cable. The robot must stand still, i.e., velocity equaling zero. By extracting the *gyrZ* variable and plotting it in Matlab, it would be possible to see a potential drift. This test could be performed five times to ensure that the data is valid. The argumentation for testing the robot five times is based on the following: By executing only one test, there is nothing to compare the data to, so it is not possible to see if the data is valid or only a bad measurement. By only executing two tests, it is impossible to determine which test is valid or a bad sample. Having five tests will allow for more valid data, and any faulty tests will more likely stand out. By having more runs, a higher validity is possible. However, due to the length of the test, it was decided that having more than five tests would take too long.

The robot can generate noise from the different hardware components, or the noise can originate in the sensor itself. To quantify the noise in the gyroscope, the measurement will be extracted for a couple of minutes, while the robot is standing still. A successful test would be that *gyrZ* is steady around zero, when the robot is at rest. The variance of the noise can be calculated to quantify the amount of noise. The standard deviation will so be calculated to find how far the signal fluctuates from the mean [56, chapter 2].

Bias

The offset that is calculated before the robot is connected to the server serves the purpose of counteracting any bias in the sensor. This initial calibration ensures that the sensor has the right starting value. This offset is always subtracted from the gyro measurement before it is used in the position estimator. The offset is constant after the robot has made contact with the server, and has a direct impact on the calculation of the position estimate. To validate that the offset is calculated correctly every time, the robot will be powered, and the offset value will be printed. This test will be done 20 times to ensure that the robot is tested thoroughly. If the offset is a value that is more or less the same in every test, it can be concluded that the calculation is correct. Since the test is executed in the same environment and with no physical changes, the offset is expected to be the same. If the results vary a lot, this could be an indication that the sensor is broken or the algorithm is

wrong. If only some results can be characterized as outliers, it is likely that something wrong has happened during the initial calculation of the offset.

90 degree test

In this test, the aim is to see if the gyroscope correctly measures a rotation of 90 degrees. This test will extract the estimated robot heading *predicted_theta*, using only the data from the gyroscope. *predicted_theta* is the same variable that is sent over BLE to the server. The reason why the test uses 90 degrees is the simplicity of creating a 90-degree corner, by using the property that a straight sides normal to another straight side gives 90 degrees. Creating a 90 degree corner is easier than creating a corner with angles such as 110 degrees or 45 degrees.

To inspect the gyroscopes ability to measure 90 degrees, the robot is placed on a book. This will enable a smooth rotation of the robot. The book and the robot are placed with the front against a wall. The book is then rotated around a fixed point, marked on the floor, until the book and the robot are parallel with the wall. This test will be executed both in the clockwise and counter-clockwise directions. The test will be executed five times in each direction to ensure the validity of the data.

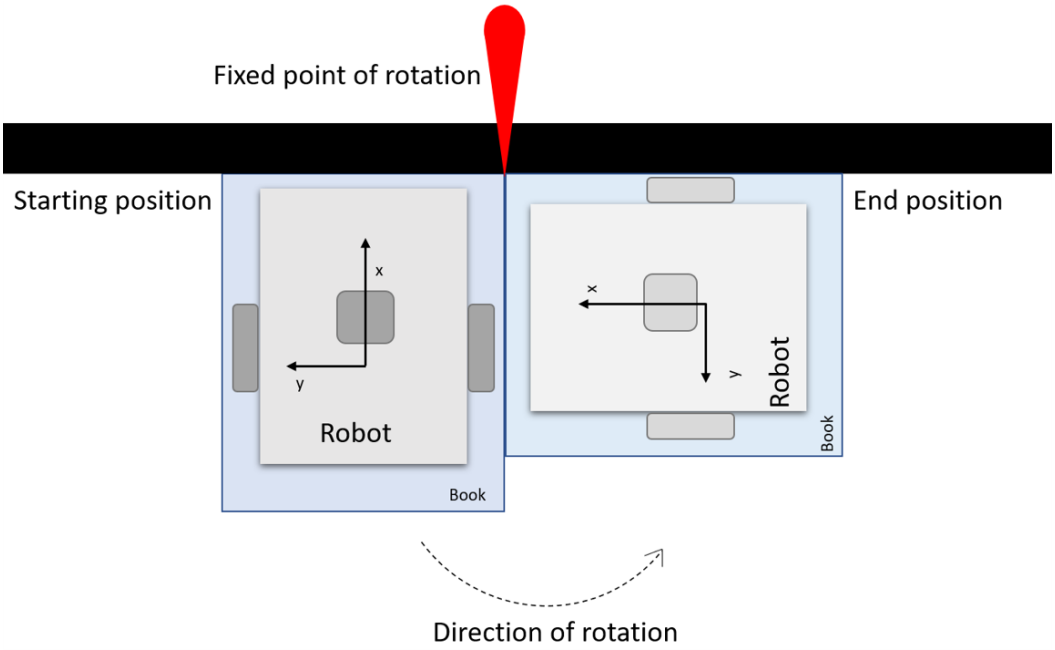


Figure 3.3: Visual description of the 90 degree test as described in 3.3.1

Improving the gyroscope data

As explained previously, the measurement signal can be corrupted by noise and drift. Mitigating drift, by only looking at the gyroscope data, is not possible. A more complex solution with multiple sensors can be used in the robot-application. An example is that the gyroscope value is kept the same if the robot is standing still (i.e., motors not activated). For reducing noise, different filters can be used. A filter is an operation on the measured data signal, which reduces the noise [57].

Noise can have different characteristics, to combat this there exists a wide range of filters. Digital filters are usually groped into Inifinte Impulse Response (IIR) and Finite Impulse Response (FIR) filters. IRR filters generally has lower sidelobes compared to FIR filters of the same order [57, chapter 10]. The choice of filter is often based on the complexity and properties of the filter. Halvorsen 2020, implemented an exponential moving average (EMA) filter [1, chapter 3.3.1], because there was not enough memory on that robot to handle a more complex filter. In figure 3.4 the frequency response of five different filters are shown. Any filters have pros and cons, but it is clear that the alternatives all have sharper roll-off[1] than the EMA-filter used by Halvorsen. The decision on exactly which filter to use should take into account whether notches are accepted, as well as the permissible amount of ripple in the passband. If avoiding notches in the response is desired, and some passband ripple is accepted, a Chebyshev I filter is suggested as a good option. Otherwise, the Butterworth filter is proposed, as it is flatter in the passband than Chebyshev I filters [58]. Tools for designing these filters are available and should be considered to be used in this project, one such tool is the Matlab DSP toolbox. If Matlab is used, C-code for the filter can be generated using *Matlab Embedded Coder* [59].

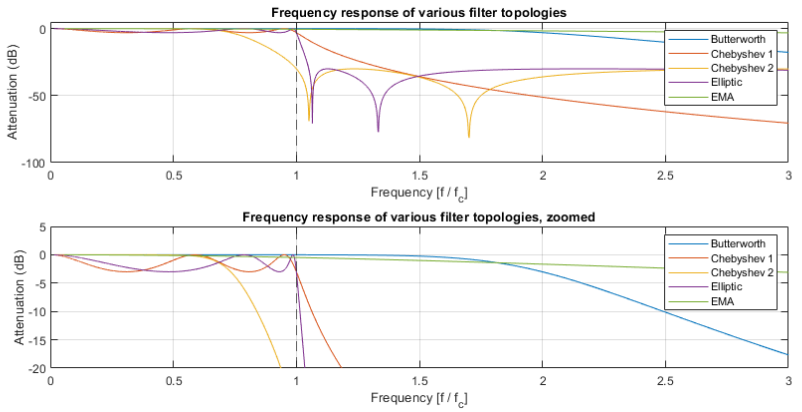


Figure 3.4: Frequency response of different filters

[1] Decrease of amplitude at the cut-off frequency

3.3.2 Accelerometer

The IMU also has a 3-axis accelerometer. The accelerometer has not been used in the robot project for many years. In previous master theses and specialization projects (e.g. Dypbukt 2020 [19]) the accelerometer is used to resolve the problem of wheel slip.

Accelerometers measure changes in velocity i.e., linear acceleration. To ensure that the sensor is working properly, the accelerometer data will be extracted. By extracting the measurement data in x-, y-, z-directions, when the robot is standing still on a horizontal surface, the following result should be seen. The x and y data should be zero, while the z-axis should have a value of one. The accelerometer gives data in the unit of g ($1\ g = 9.81\ \frac{m}{s^2}$), meaning the z-component should show 1 g. This is because the gravitational field will be parallel to the z-axis, but orthogonal to the x- and y-axes. This method verifies that the z-axis gives the correct measurement. The same test will be executed in all three positive axes, to determine if they all give a valid result. If all axes show 1 g, when pointing in the positive direction of the gravitational force, the accelerometer can be assumed to be operational. An image of the coordinate systems in the test is shown in figure 3.5, this is shown relative to a world coordinate frame.

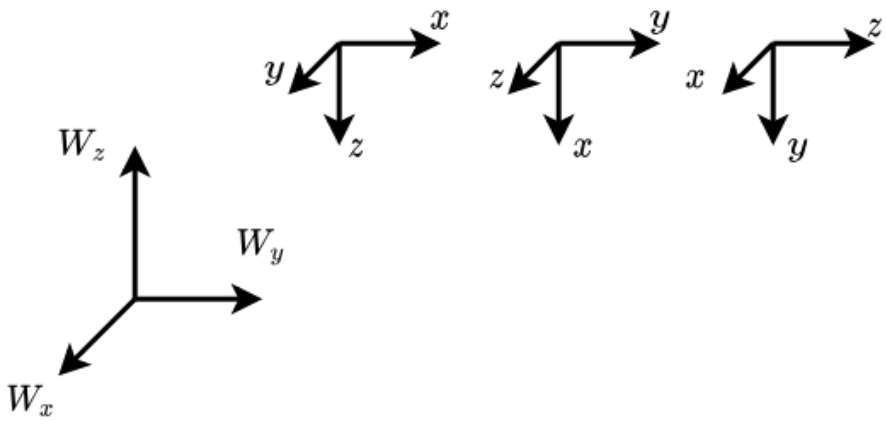


Figure 3.5: Coordinate system of the three tests of the positive gravitational force

Another test is when the robot is moving forward. During this operation, it is expected that the acceleration in the x-direction will have a spike from driving begins until the velocity is constant, or the robot stops. This will be tested by extracting the accelerometer data when the robot drives forward.

A USB cable and the Putty software will be used to extract and log the acceleration measurement. Putty is a desktop program that allows reading serial communication from a COM-port [60]. Using serial communication over USB ensures that the test will not

be disturbed by an unstable BLE connection, and it is possible to log all the data from the x-, y- and z-axes at the same time. The drivers for communicating from the robot to Putty is written by Torgeir Myrvang, and the steps to use Putty is described in appendix A.5

Improving the accelerometer data

The IMU is not placed in the centre of rotation, in the robot frame. The placement has no impact on the gyroscope's angular velocity, but it has a significant impact on the accelerometer. The accelerometer will pick up the rotational movement of the robot, not only linear movement. To get valid acceleration measurements, the IMU must then be localized in the centre of rotation. Due to the complexity of moving the sensor, it is desirable to design a mathematical description of the accelerometer position. A suggested solution is a mathematical transformation of the accelerometer data from the IMU frame to the robot frame.

First, some definitions and constraints have to be set. The robot frame, also called body frame, has the notation x^b, y^b and z^b. The measurement from the IMU has the notation x^m, y^m and z^m. The parameters p_b^w and p_m^w are the origin of the body frame and the measurement frames respectively, measured in a fixed world frame. The vector r_{bm}^b is the IMU displacement, relative to the body, expressed in body frame.

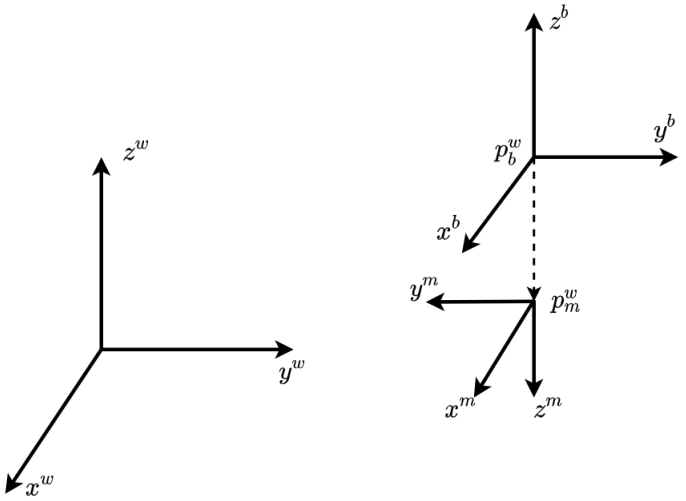


Figure 3.6: Coordinate frame of the robot and the IMU, b is the body frame, m is the measurement frame and w is the fixed world frame

From the accelerometer the linear measurement has the notation $\ddot{x}$, $\ddot{y}$ and $\ddot{z}$. The gyroscope will give the angular velocity measurements p, q and r. When the IMU is not placed in the center of rotation, the accelerometer will measure linear acceleration, centripetal acceleration and traversal acceleration [61]. The goal is to transform the accelerometer data, to only contain the linear acceleration. Equation 3.24 shows the extracted measurements from the IMU, i.e., the acceleration (a) and gyroscope (ω) measurements.

$$a_m^w = \begin{bmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{z} \end{bmatrix}, \quad \omega_{wb}^b = \begin{bmatrix} p \\ q \\ r \end{bmatrix} \tag{3.24}$$

The relationship between the body's origin and the IMU position can be expressed as in equation 3.25. Where R_b^w is the rotation matrix from the body frame to the world frame.

$$p_b^w = p_m^w - R_b^w r_{bm}^b \tag{3.25}$$

The velocity is then found by taking the time derivative of equation 3.25. The result of this derivation can be seen in equation 3.26.

$$\dot{p}_b^w = \dot{p}_m^w - \dot{R}_b^w r_{bm}^b - R_b^w \dot{r}_{bm}^b \tag{3.26}$$

$$v_b^w = \dot{p}_m^w - R_b^w S(\omega_{wb}^b) r_{bm}^b \tag{3.27}$$

The following properties have been used to find equation 3.27, derived in [51].

$$\dot{p}_m^w = v_m^w = R_b^w v_m^b \tag{3.28}$$

$$\dot{R}_b^w = R_b^w S(\omega_{wb}^b) \tag{3.29}$$

Where $S(a)$ is the skew-symmetric matrix and $\omega_{wb}^b = [p, q, r]^T$ is the angular velocities expressed in body frame. $\dot{r}_{bm}^b$ is equal to zero, as the IMU is assumed rigidly attached to the robot and is therefore constant in the body frame.

The acceleration is found by taking the time derivative of equation 3.27. Resulting in the expressions seen in equations 3.30 and 3.31.

$$\dot{v}_b^w = \dot{R}_b^w v_m^w + R_b^w \dot{v}_m^w - \dot{R}_b^w S(\omega_{wb}^b) r_{bm}^b - R_b^w \dot{S}(\omega_{wb}^b) r_{bm}^b - R_b^w S(\omega_{wb}^b) \dot{r}_{bm}^b \tag{3.30}$$

$$a_b^w = R_b^w S(\omega_{wb}^b) v_m^b + R_b^w a_m^b - R_b^w S^2(\omega_{wb}^b) r_{bm}^b - R_b^w S(\dot{\omega}_{wb}^b) r_{bm}^b \tag{3.31}$$

Where $\dot{\omega}_{wb}^b = [\dot{p}, \dot{q}, \dot{r}]^T$ is the angular acceleration expressed in body. This will be found by time derivation of the measurement from the gyroscope. The centripetal acceleration is found as the third expression in equation 3.31, and is $R_b^w S^2(\omega_{wb}^b) r_{bm}^b$. The traversal acceleration is the expression $R_b^w S(\dot{\omega}_{wb}^b) r_{bm}^b$. By subtracting both, the linear acceleration is left. The measurement from the encoders is expressed as $R_b^w S(\omega_{wb}^b) v_m^b + R_b^w a_m^b = a_m^w$. To realise this theory, the distance of r_{bm}^b and the rotation matrix R_b^w has to be found. The distance from the origin between the robot frame and the IMU frame can be measured and become the r_{bm}^b. The rotation matrix from the robot frame to the world frame can be seen as a rotation around the z-axis, as drawn in figure 3.7.

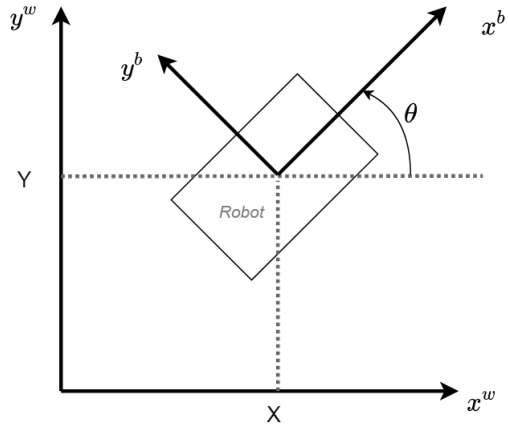


Figure 3.7: Reference between fixed world frame and robot frame

With the notation of an Euler angle, the rotation matrix can be expressed as seen in equation 3.32. This assumes no translation between the world frame and robot frame. In other words that the two reference frames have the same origin, at the robots center of rotation.

$$R(\theta)_z = R_w^b = \begin{bmatrix} \cos(\theta) & -sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.32}$$

3.3.3 Encoders

The problem of determining how far a wheel has rotated, can be solved by using an encoder. A quadrature encoder, like the ones mounted to the Uxcell motors, consists of two sensors that react to the pattern of an incrementally coded disc. The encoder sensors are fastened with a 90 degree offset. This will give a 90 degree phase shift between the encoder signals. A visual representation of this is seen in figure 3.8. To improve the software for the system, the phase shift can be used to determine the direction the wheel is rotating. This is done by checking whether phase B is leading or lagging compared to phase A. Today, determining the direction of rotation is done by assuming that the wheel rotates in the requested direction. This decision couples the motor controller software and encoder software modules more than necessary.

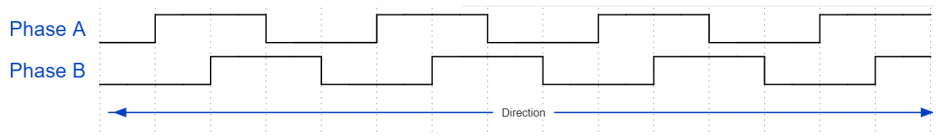


Figure 3.8: Quadrature encoder concept

The initial plan was to validate the number of encoder ticks from the DAUG motors. Since the motors were broken and new ones had to be used instead, a new wheel factor had to be calculated. The wheel factor is a constant in the robot application and describes the relationship between the wheel circumference and the motor's rotation. The wheel factor is defined in the robot application as seen in equation 3.33.

$$\text{Wheel factor} = \frac{\text{circumference}}{\text{encoder ticks per wheel rotation}} \tag{3.33}$$

It is customary to have the number of encoder ticks and gear ratio listed in the datasheet. Because this information is not listed in the technical specification [49], experimental methods must be executed to determine this. Since the wheel factor only uses encoder tics per wheel rotation, the goal is to find this number. Two tests will be executed. In both methods, the counting of encoder ticks is the key. For registering the encoder ticks, the existing function *vMotorEncoderRightTicsFromISR(...)* and *vMotorEncoderLeftTicsFromISR(...)*, in the robot application will be used.

The first method will find the average number of encoder ticks, by manually rotating the wheel. It is chosen to rotate the wheel 20 times, to get a large data set and reduce the impact of human error. With a mark on the wheel, it is possible to visually determine when the wheel has rotated one turn. The encoder ticks are printed continuously. The last encoder value will represent the accumulated number of encoder ticks for the 20 rounds. The total number of encoder ticks will be divided by 20 to find the average number of ticks.

The code to get this data is written in list 3.1. *sei()* switches interrupts on. The while loop will run as long the robot is powered. The interrupt service routine (ISR) registers the motor ticks, which are retrieved using *vMotorEncoderRightTickFromISR(...)* and *vMotorEncoderLeftTickFromISR(...)*. The total counted wheel ticks is finally printed to Putty.

```
sei();
int16_t leftWheelTicks = 0;
int16_t rightWheelTicks = 0;
uint8_t leftEncoderVal = 0;
uint8_t rightEncoderVal = 0;
uint8_t gLeftWheelDirection = 0;
uint8_t gRightWheelDirection = 0;

  while(1){
    ATOMIC_BLOCK(ATOMIC_FORCEON){
      leftEncoderVal = gISR_leftWheelTicks;
      gISR_leftWheelTicks = 0;
      rightEncoderVal = gISR_rightWheelTicks;
      gISR_rightWheelTicks = 0;
    }
    vMotorEncoderRightTickFromISR(gRightWheelDirection, &rightWheelTicks,
    rightEncoderVal);
    vMotorEncoderLeftTickFromISR(gLeftWheelDirection, &leftWheelTicks,
    leftEncoderVal);

    printf("Encoder left Wheel: \t %d \t \n",leftWheelTicks );
  }
```

Listing 3.1: SW to extract the encoder ticks from let wheel

In the second method, the robot will drive forward. By connecting the robot to the server, controlling it with manual navigation, it is possible to command the robot to drive one meter straight forward. Then, by measuring how far the robot has driven, calculating how many times the wheel has rotated, is easily done. During this method, the encoder ticks will also be extracted. The encoder ticks per wheel rotation can be found by dividing the total number of ticks by the total number of wheel rotations.

Improving the encoders

The number of encoder ticks per rotation is a fixed value. The exact amount is the result of the hardware in the encoder. Therefore it is not possible to improve the resolution of the encoder itself. On the other hand, the current design only uses one of the two encoder phases, meaning the total resolution can be doubled by using both phases. XOR'ing the signal from each phase results in a new signal with a frequency of two times the original signal, see figure 3.9. The combined signal can be used where the single-phase is used in the software now, allowing the current framework for counting ticks to be used. If one also counts both positive and negative edges of this combined signal, the resolution can be improved by a factor of four, compared to the original design.

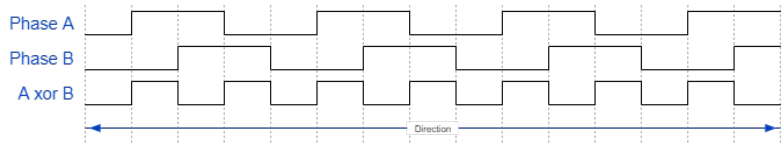


Figure 3.9: XOR phase A and B to improve the resolution

3.3.4 Compass

The compass has been discussed in previous reports. The disagreement between earlier results raises an interesting question on whether the compass could improve the position estimation or not. As the robot only operates in the xy-plane, compass data from the z-axis will not be used. By rotating the robot 360 degrees, and log the x- and y-data from the compass, it is possible to visualise the data after the test. The robot will rotate by itself, by requesting the motors to drive in opposite directions of each other, meaning the robot will rotate around its own z-axis.

An ideal compass will have the measurement orientated as a circle around the origin. Matlab will be used to plot the x and y values from the compass, to get this visualisation. A perfect compass with no distortion is shown in figure 3.10. The image is taken from the Vectornav page about compasses and calibration [62]. The test will be executed two times to look for a trend in the accuracy of the compass. It is not seen as beneficial to execute the test more times.

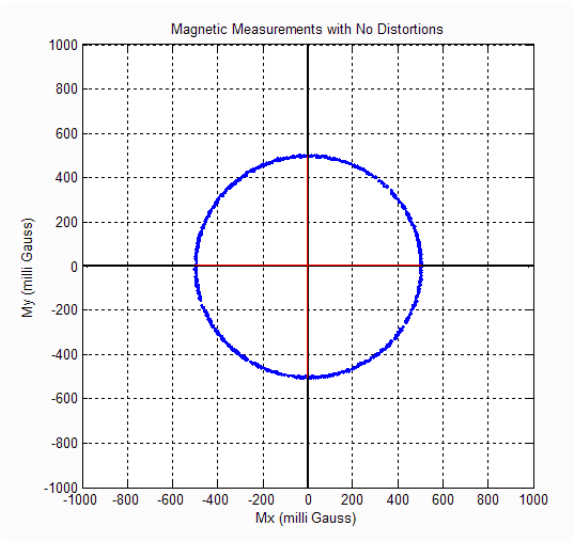


Figure 3.10: Compass measurement with no distortion, image taken from [62]

Compass Calibration

In 2016, Ese [15] did the initial work on the compass, including calibration. There is no evidence that the compass has been calibrated since then. Therefore calibration of the sensor should lead to an improvement of the compass. The calibration method is similar to the calibration done by Ese in 2016, [15, p. 43].

All measurements, in general, are expected to have errors and distortions, this is also true for compasses. The distortion is categorised into hard and soft iron. An object that produces a magnetic field is categorised as a hard iron [62]. The object could cause a constant bias if the object is fastened in the same reference frame as the compass. Soft irons are changes in the existing magnetic field, like direction. Compared to the hard iron distortion, the distortion from soft irons is more challenging to measure and counteract.

The method for calibrating the compass, to counteract the hard iron distortion, is similar to the compass testing. While the robot turns 360 degrees, the x- and y-data from the compass is extracted. The lowest and highest value from the x and y variable is found. The computation for finding the x and y offset i shown in equation 3.34 and 3.35. To ensure good communication and that no data is lost, the robot will be connected via USB, and the data value is printed to Putty.

$$xComOff = \left(\frac{xComMax - xComMin}{2}\right) - xComMax \tag{3.34}$$

$$yComOff = \left(\frac{yComMax - yComMin}{2}\right) - yComMax \tag{3.35}$$

3.3.5 IR sensors

Even if the IR sensors are not used directly for finding the internal estimate of the robots position, it is important for the mapping. It is used so that the robot can determine the true distance to obstacles. For the goal of creating a correct map, it is desirable to check the precision in the distance measurements, and potentially calibrate the sensors.

The IR sensors, as described previously in 2.1.2, uses the reflection of light to calculate the distance, and outputs a voltage based on the distance [63]. Objects with different colours will reflect different frequencies of light [64], and can influence the IR sensor measurements. As seen in figure 3.11, the distance found for the grey and white paper is slightly different. This deviation only emerges when the distance is over 45 centimetres. Due to the small deviation, the light beige walls used to test larger mapping (see section 3.1.4), will be used to test and calibrate the IR sensors.

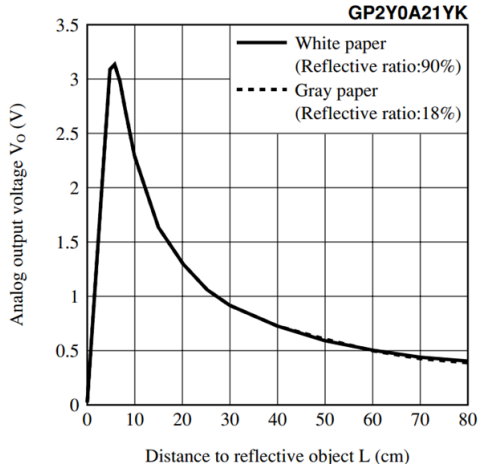


Figure 3.11: Voltage to distance curve from IR-sensor datasheet [30]

The current IR calibration will be tested to see how accurate it is. This will be done by printing the distance found from the four IR sensors. The data is transmitted over a serial link and is showed in Putty to ensure all measurements are logged. The servo will not be initialized in *main.c* to ensure that the sensor tower does not rotate, making it easier to get a correct and consistent set of measurements. One of the light beige walls, used in section 2.1.2, will be placed at a known distance from the robot, and then the IR-sensor measurements are extracted. The setup is visualized in figure 3.12. Each sensor will then be tested to measure different fixed distances between 10 and 80 centimetres. This range is where the IR measurements are valid. The accuracy of the IR sensors will be done at 10 cm intervals. A smaller step size of five centimetres is used in the range of 10 to 30 centimetres. This is because the expected curve has a higher gradient at shorter distances, as seen in figure 3.11. Another reason for reducing the step size is the importance of the robot having precise measurements at short distances, to prevent collisions.

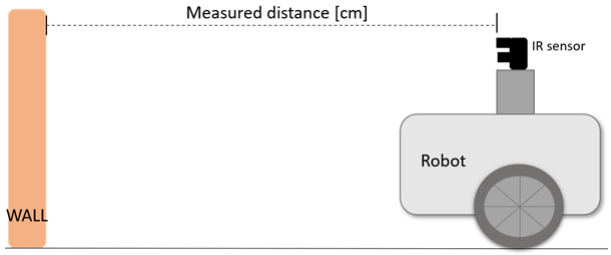


Figure 3.12: Test setup for distance measurement

New calibration

Calibration of the IR sensor will be done with a method called *curve fitting* [65]. This method is used to find a mathematical function based on a set of data points. For the curve fitting method, a data set of distances with corresponding voltages will be saved, for each IR sensor. This will be used to find the power function that best fits the individual sensors.

The goal is to have only a few lines of code that gives the distance from an IR measurement, instead of the current implementation that uses a large array as a look-up table. This modification will reduce the amount of program memory used. It will also be a more flexible solution, where the result is less coupled with the specifics of the ADC used (e.g. reference voltage and resolution). This can hopefully provide more accurate distance measurements, which will give a more accurate mapping.

A visualization of the data flow that the new calibration will lead to is shown in figure 3.13. The IR measurement has IR light as an input and will output a voltage. The ADC reads this, and it transforms the signal into a digital value. The bit value then has to be calculated back to determine the input voltage. This voltage is used in the calculation of the distance.

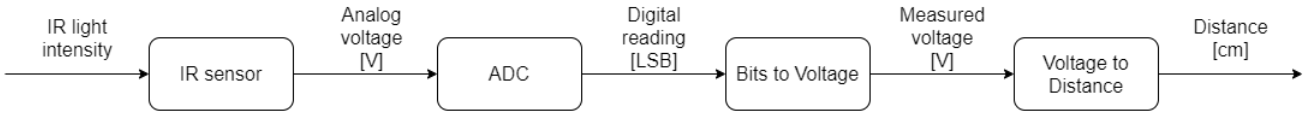


Figure 3.13: Data flow for IR measurements in the robot

When calibrating the IR sensor the full resolution of the ADC will be used (compared to the old 8-bit resolution). By using 10 bits the resolution is four times higher than with 8 bits, see calculation in equation 3.36. A better resolution will improve the result of the new IR calibration. In summary, it is seen as beneficial to use 10 bits instead of 8. The software code to read the ADC is found in listing 3.2.

$$\frac{2^{10}}{2^8} = 2^2 = 4 \tag{3.36}$$

```
uint16_t ui16DistSens_readAnalog(uint8_t distSensDir){
  uint16_t ui16_analogValue;

  /* Choose channel */
  ADMUX = distSensDir;

  /* Enable internal 2,54V AREF */
  ADMUX |= (1<<REFS1) | (1<<REFS0);

  /* Start conversion */
  ADCSRA |= (1<<ADSC);
  loop_until_bit_is_clear(ADCSRA, ADSC); // Macro from avr/io.h: Wait
    until bit bit in IO register sfr is clear.

  /* Join the data from the two ADC registers, creating a 10-bit value */
  ui16_analogValue = ((ADCH & 0x03)<<8) | (ADCL);
  return ui16_analogValue;
}
```

Listing 3.2: Software design of a new IR calibration

As shown in the data flow, the ADC bits have to be converted into a number representing the voltage. This is done by multiplying the ADC signal with a coefficient, defined in this thesis as C. From the datasheet and documentation in the robot application, it is found that V_{ref} equals 2.56 V and that the ADC has 10 bits of resolution [20]. With this information it is possible to calculate the coefficient C, using equations 3.37 to 3.40. It can be noted that the number of possible digital values used are $2^{10} - 1$, this is done to ensure that the largest readable input (V_{ref}) will be 11 1111 1111 instead of wrapping around to 10 zeros. The result of equation 3.40 gives the voltage for one bit (denoted LSB). In the code, this coefficient C is multiplied with the output of the ADC, converting it to a value representing the voltage.

$$LSB = \frac{C}{V_{ref}} \cdot (2^{10} - 1) \tag{3.37}$$

$$C = \frac{LSB \cdot V_{ref}}{2^{10} - 1} \tag{3.38}$$

$$C = \frac{1 \cdot 2.56}{1023} \tag{3.39}$$

$$C = 0.00250244379 \tag{3.40}$$

After the voltage has been found, the distance can be calculated with the new voltage-to-distance method. The type of function for this conversion is a power function, as found in the datasheet [30]. This can be used in the calculation of the distance, by applying equation 3.41, where d is the distance from the object, and v is the measured voltage, while a and b are the coefficients found through curve fitting.

$$d = a \cdot v^b \tag{3.41}$$

The curve fitting method uses knowledge about the shape of the measurement curve, shown in figure 3.11. From 10 centimetres to 80 centimetres, the curve is a power function. A modified version of the setup used to validate the old calibration (see figure 3.12), will be used to obtain data for the curve fitting method. The main addition is that the analogue voltage is measured via an oscilloscope, instead of extracting the measured distance from the IR sensor. This data gathering for the calibration is executed by taking three samples of the voltage, at each distance. It will ensure a better data set for the calculation of the coefficients. Because of the large gradient in the IR sensors' theoretical measurement output, the distance has to have a low step size in the lower range. Therefore the distances found in table 3.2 are chosen.

cm	10	11	12	13	14	15	20	25	30	40	50	60	70	80

Table 3.2: Distance to calibrate the IR sensors

After all the measurement have been executed, the curve fitting method will be applied. The plan is to do this using the Matlab *curve fitting* toolbox. This software will calculate the coefficients based on a data set and a desired type of function.

To evaluate the new method, the new calibration will be implemented in the robot application. More exactly in the function *ui8DistSens_calib_readCM(...)* in *distSens_g2d12.c*. With a simple switch statement, it is easy to have the conversion for all the sensors in the same function, seen in list 3.3. Finally, the new calibration will be tested with the same method and setup as for the old calibration, as described previously.

```
1  uint8_t ui8DistSens_calib_readCM(uint8_t sensorDirection){
2      uint16_t ui16_analogValue;
3      uint8_t ui8_cmValue;
4
5      /* Choose channel */
6      ADMUX = sensorDirection;
7
8      /* Enable internal 2,54V AREF */
9      ADMUX |= (1<<REFS1) | (1<<REFS0);
10
11     /* Start conversion */
12     ADCSRA |= (1<<ADSC);
13     // Macro from <avr/io.h>, wait until bit bit in IO register is set.
14     loop_until_bit_is_clear(ADCSRA, ADSC);
15
16     /* Join the data from the two ADC registers, creating a 10-bit value
       */
17     ui16_analogValue = ((ADCH & 0x03)<<8) | (ADCL);
18
19     // ADC-coefficient
20     float volt = 0.0025024437927664 * ui16_analogValue;
21     switch (sensorDirection)
22     {
23       case distSensLeft:
24         ui8_cmValue = a1 * pow(volt, b1);
25         break;
```

```
26
27      case distSensFwd:
28        ui8_cmValue = a2 * pow(volt, b2);
29        break;
30
31      case distSensRight:
32        ui8_cmValue = a3 * pow(volt, b3);
33        break;
34
35      case distSensRear:
36        ui8_cmValue = a4 * pow(volt, b4);
37        break;
38
39      default:
40        ui8_cmValue = 0;
41    }
42 }
```

Listing 3.3: Software design of a new IR calibration

Chapter 4

Result

In this chapter, the result from executing the methods described in chapter 3 are shown and reviewed. The rotation of the acceleration data is not implemented and is therefore not tested. The same is true for the extended Kalman filter, due to the limited time available for such a project.

4.1 Driving preformance

When executing some of these tests, the Optitrack system was used. The calibration of the system was done according to what is described in A.6. The *wanding* was done until the software showed the status: *"Wanding quality: very high"*. After the calibration was complete, the calibration quality was shown as *Very High*, which indicates a mean error of less than 0.5 millimetres.

4.1.1 Square test

The test was executed in B333 at NTNU. The error-factors implemented by Dybpukt for the gyroscope and encoder measurement were tested to see how it affected the position estimate. The *gyro weight* was set according to what was initially described in section 3.2, i.e., gyro weight equal to zero if the robot is not turning, and one if it is turning.

Some of the visualised results of the square test are shown in appendix B.1. During post-processing the error distance was calculated. It shows the error from where the robot was, compared to the target corner. This result is found in figure 4.1. It can be seen that the overall error is lower when **not** using the error factor.

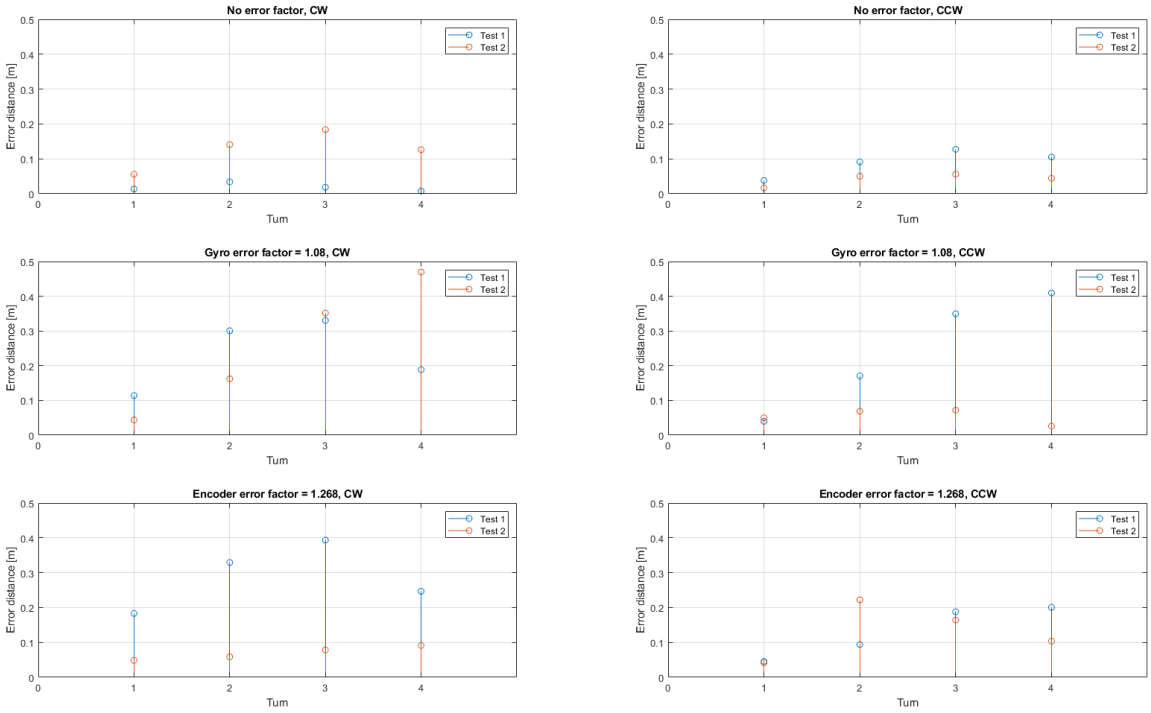
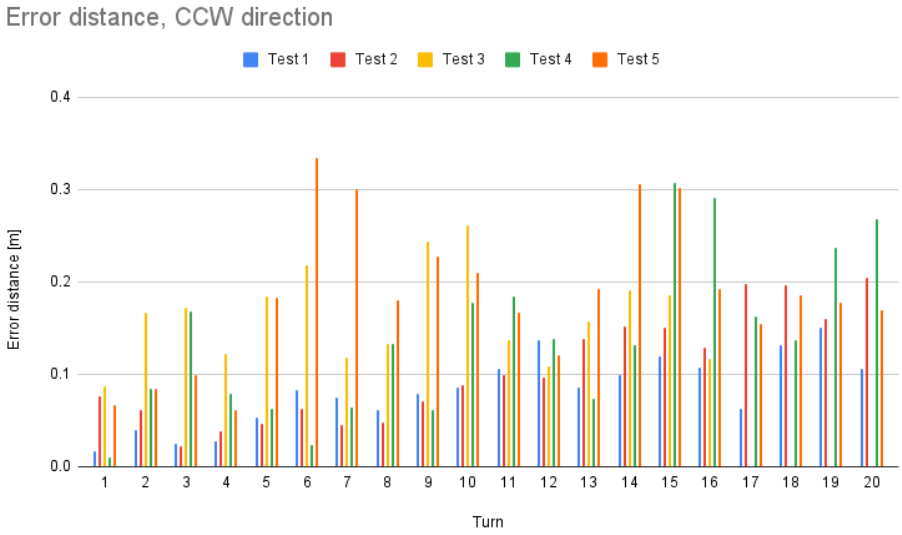


Figure 4.1: Error distance in square test

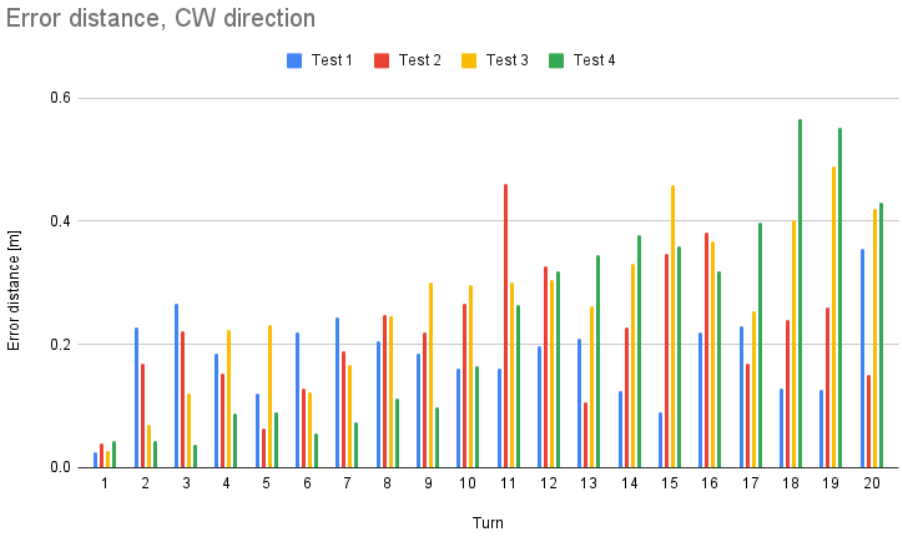
4.1.2 Continuous square test

The robot will drive the *continuous square tests* to find the precision in position over longer intervals. The error factors where disabled for this test, as this is what gave the best result in the single square test.

The data was post-processed, and each run was plotted (shown in appendix chapter B.2). The actual position at which the robot turned, was compared to the point intended. The distance from the real position to target was then calculated. The result is shown in figure 4.2. The average distance error is calculated to find the error distance over time, this is found in figure 4.3. Here it can be seen that the average error distance varies from corner to corner.



(a) CCW direction



(b) CW direction

Figure 4.2: Result of the continuous square test

The continuous square test was executed in room B333 at NTNU. The calibration of the Optitrack system went without trouble. The robot got the commands for the square test, with the server in manual mode. Some unforeseen problems were discovered, in

some cases the robot would not move forward, and a buzzing sound from the motors was heard. Another challenge was that the robot lost connection to the server. The problem occurred when other Bluetooth devices, such as Bluetooth headphones, were active in the room. After the other devices were turned off, the robot had no problem connecting to the server.

The robot was tested when driving CCW and CW. All the tests were executed with five continuous squares, with the exception of test 3 in the CCW direction, which had just four rounds, caused by a mistake by the operator. This mistake was not detected until the data was post-processed, and a new test has not been conducted. In the CW direction, one test resulted in a substantial error, seen in figure 4.4. This test has been excluded from the results, because of the large and unlikely position error. In the post-processing of the continuous square test, the data was plotted with a colour gradient indicating time. This was done to more easily separate each round in the plots.

From figure 4.2a and 4.2b it can be seen that the distance to each corner varies. A clear tendency is that the position accuracy gets worse from round one to round five in both directions. The average error has a tendency to increase over time, seen in figure 4.3, the data is available in appendix B.2.

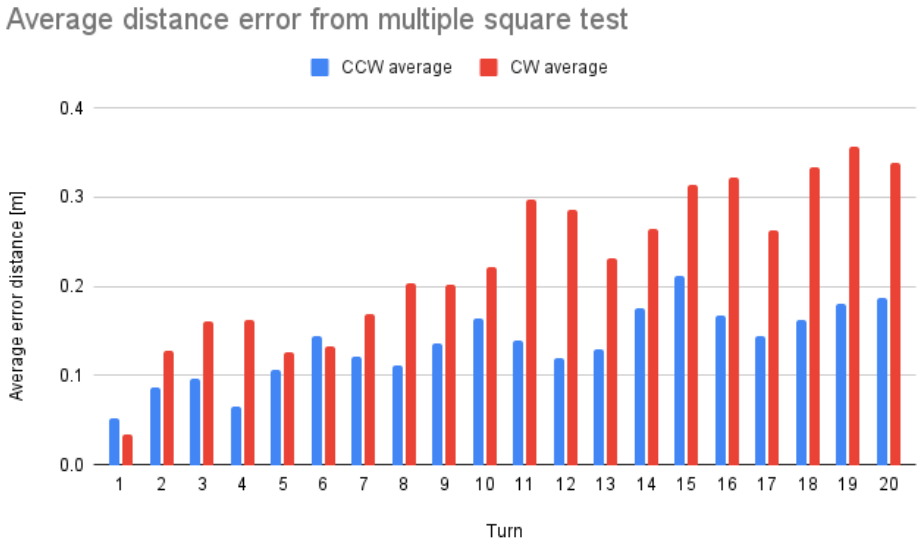


Figure 4.3: Average distance error from the multiple square test run

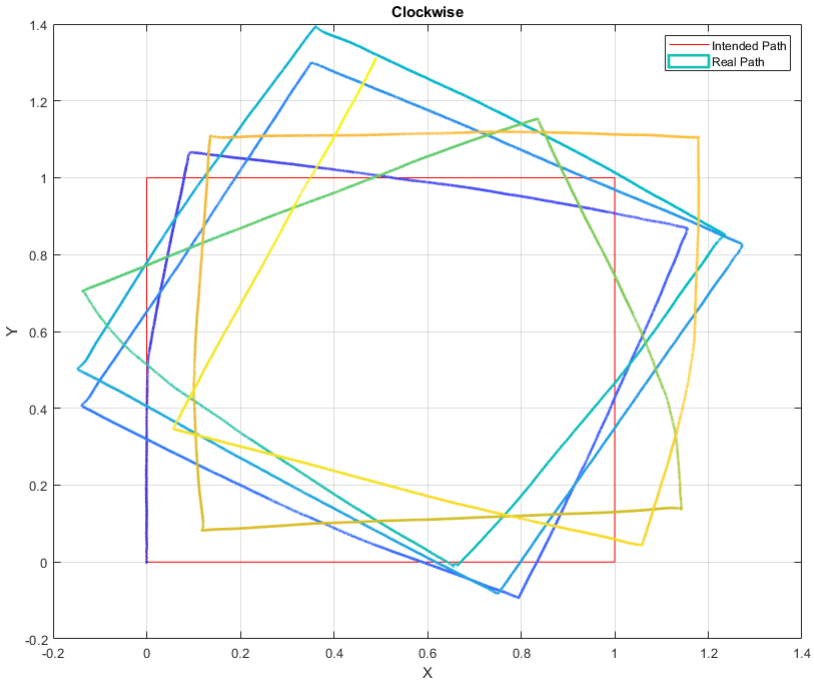
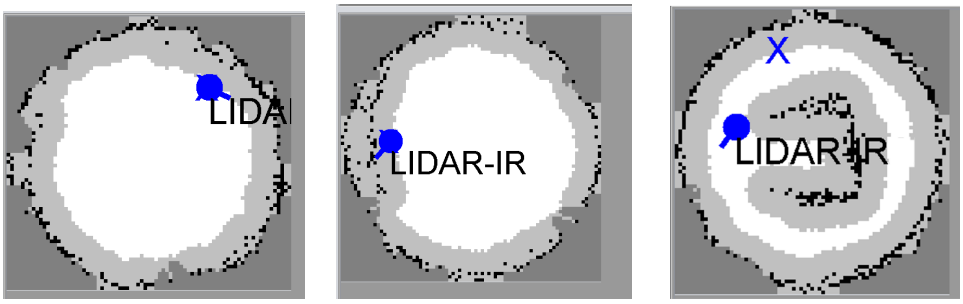


Figure 4.4: Outlier from the multiple square run in the CW direction

4.1.3 The round court

The robot was tested in a round court, to test the detection of non-straight walls. The navigation and mapping were done in server mode. After the first, successful, mapping of the round court, an object was placed there, to see if the robot would have problems navigating the area without crashing. It is evident from the graphical map in the server, shown in figure 4.5, that the robot can detect obstacles with sloped edges.



(a) Successful mapping of the round court

(b) Drift of the wall, non-successful mapping

(c) Successful mapping with obstacle

Figure 4.5: Different outcomes of mapping the round court on the server, the name of the robot was LIDAR-IR, even though the lidar was not used

4.1.4 Larger tracking court

A larger environment was set up to test the robots ability to navigate over a more extended time period. The method used to test the robot is described in section 3.1.4. The larger testing court was built according to the drawings in figure 3.2. A picture of the actual court is shown in figure B.9 in the appendix. As seen in the result, figure 4.6, the robot could not map the environment clearly.

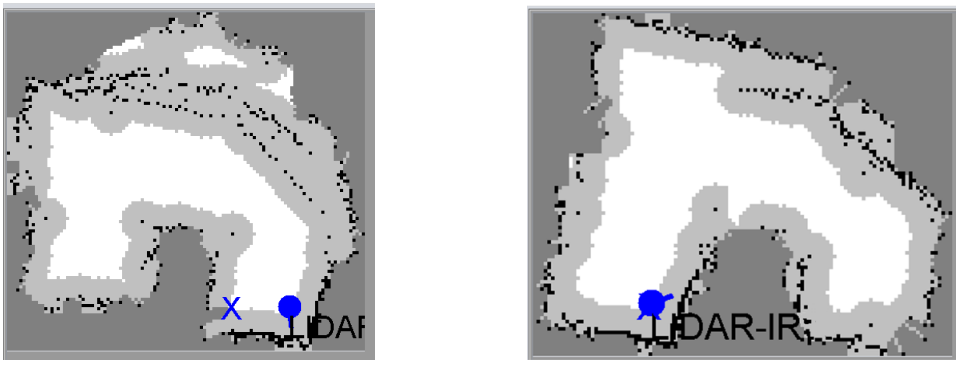


Figure 4.6: Result of mapping the larger court

4.2 Position Estimation

In this section, the results from testing the robot heading is presented. The method used is described in chapter 3.2.

4.2.1 dTheta

The different gyro-weights were set, and both tests executed as planned. In the first test, where the robot was standing still, dTheta was seen to be noisy when the gyroscope was used. This is seen in figure 4.7. When using only the encoder while the robot was at rest, gave results without outliers or noise, seen in figure 4.8. In the second test where the robot was driving one meter forward, noise is observed in every test. This is seen in figure 4.9 and in figure 4.10.

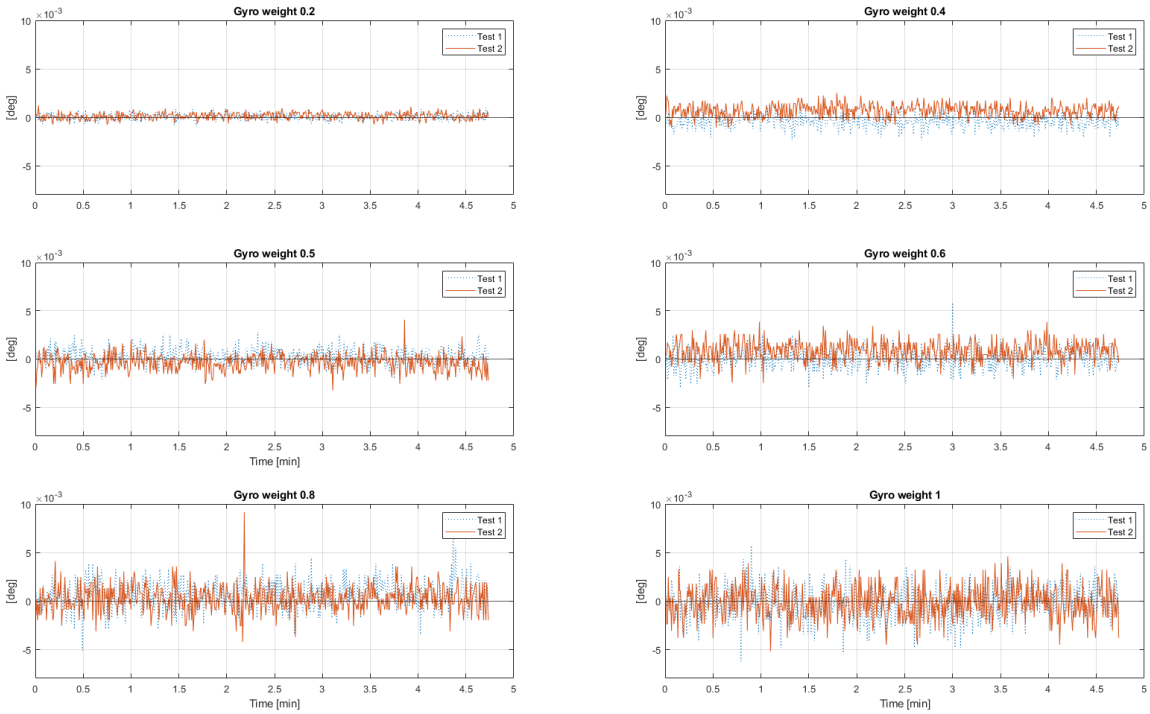


Figure 4.7: dTheta value when the robot is standing still, for different gyro weights

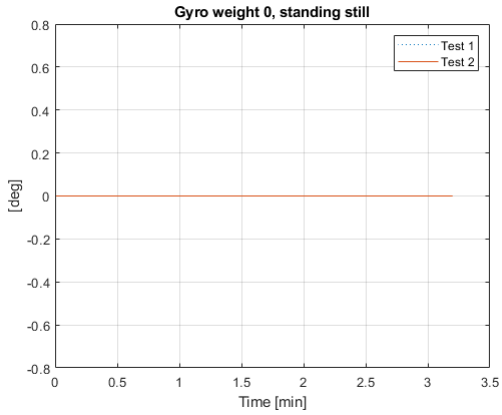


Figure 4.8: dTheta value using only encoders, robot is standing still

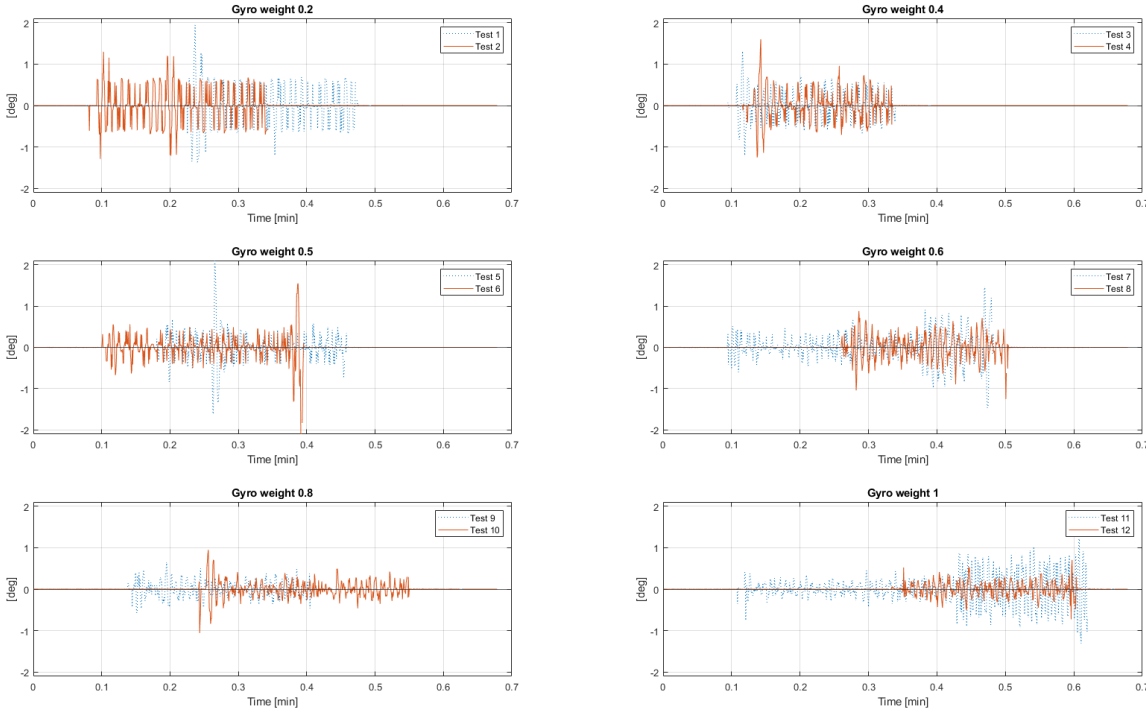


Figure 4.9: dTheta value when the robot is driving, for different gyro weights

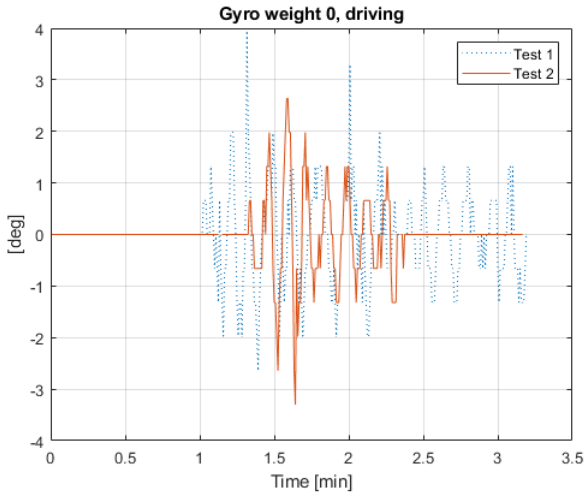


Figure 4.10: dTheta value using only encoders, robot is driving

4.2.2 Robot heading

Drift in heading

The code used to compute and print the robot heading is found in listing B.1 in the appendix. The robot heading results when the gyro weight was zero, is found in figure 4.11. Here it is observed that the heading is zero the entire time. The results of using other gyro weights can be seen in figure 4.12.

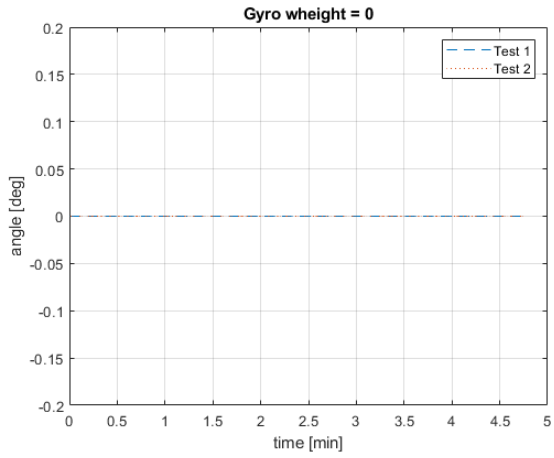


Figure 4.11: Robot heading during five minutes of standing still with gyro weight = 0

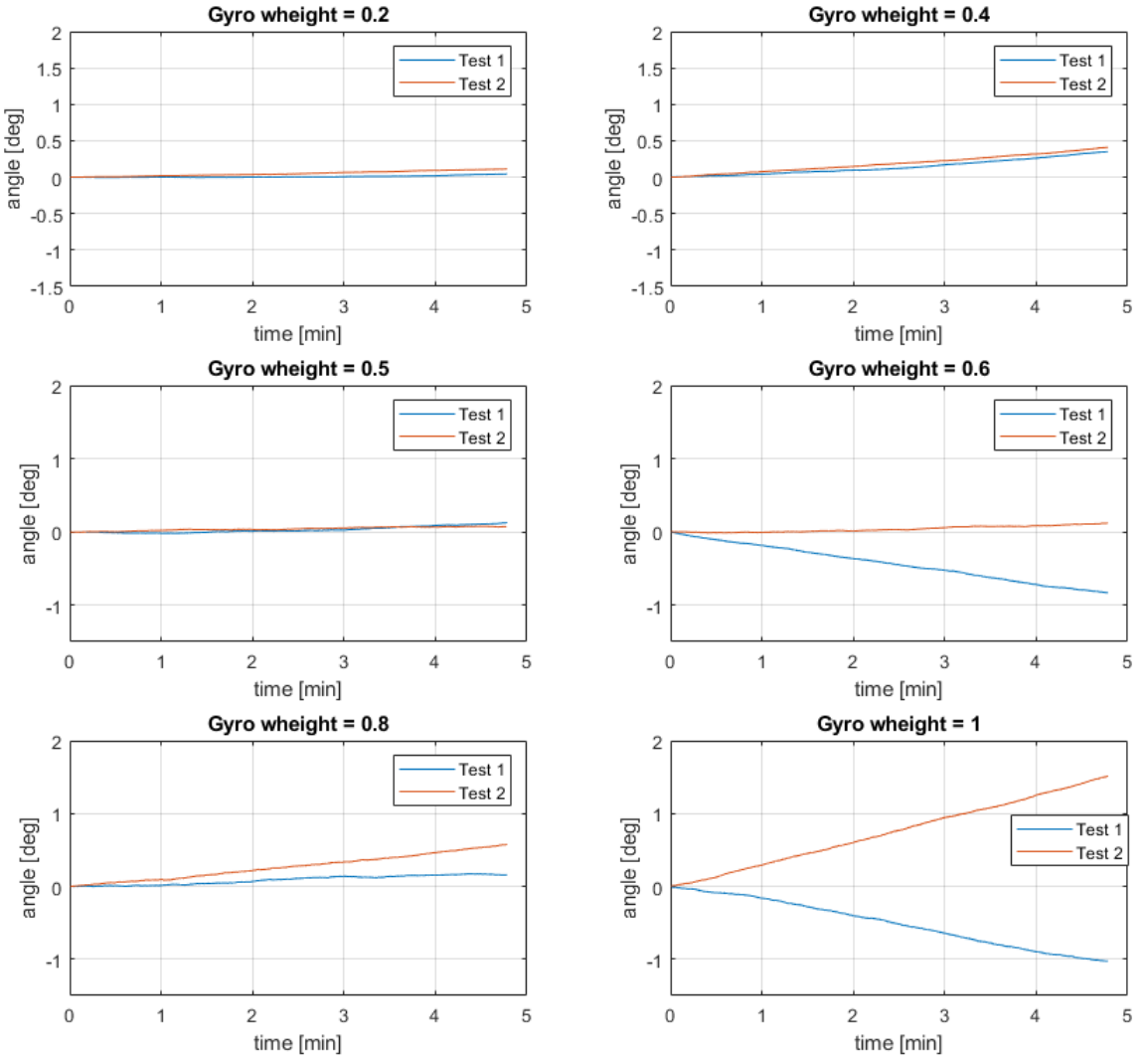


Figure 4.12: Logged robot heading during five minutes standing still

The final orientation from the five minute test is found in table 4.1. Here it is seen that the heading varies but that the best result is when the gyro weight equals zero. The largest error in the robot heading is found when the gyro weight is one.

	Robot heading	
Gyro weight	Test 1	Test 2
0	0.0	0.0
0.2	0.044	0.111
0.4	0.35	0.40
0.5	0.12	0.072
0.6	0.110	-0.833
0.8	0.57	0.11
1.0	1.51	-1.026

Table 4.1: Robot heading [deg] after five minutes

Straight driving and heading

The straight driving test was executed in the camera-room, B333, at NTNU. The robot was able to drive 3 meters on the floor. The test was executed two times for each gyro weight. Data from Optitrack is plotted, showing the driving distance in the x and y directions, the results are presented in figures 4.13 and 4.14. The driving distance, error distance and average error distance are summarized in table 4.2.

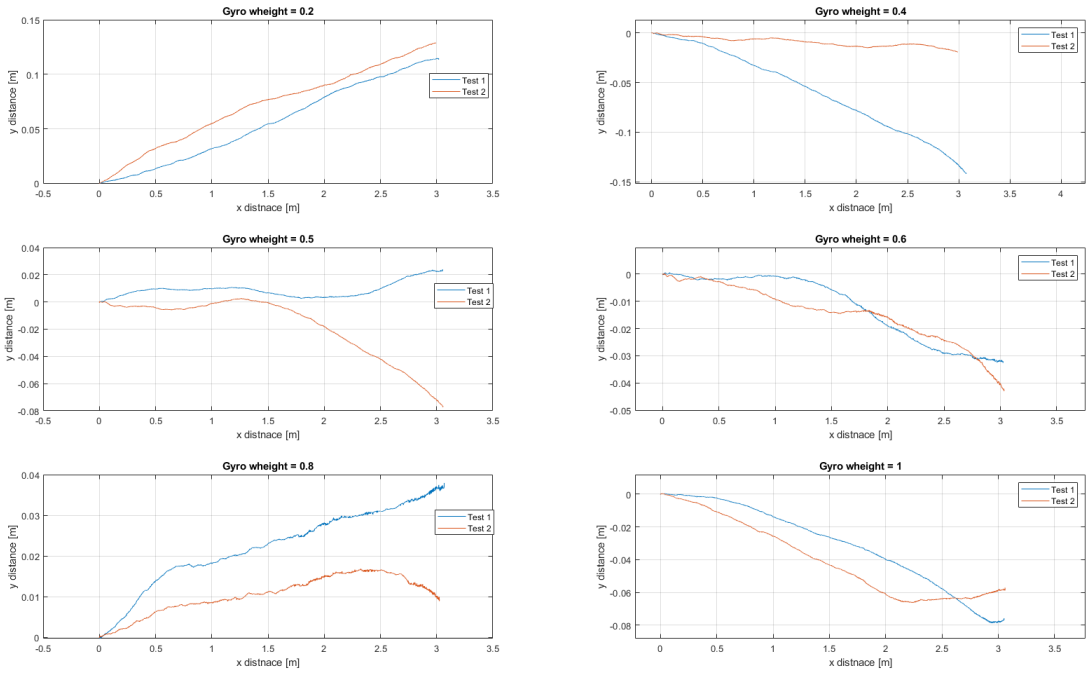


Figure 4.13: Result from the robot driving three meters in a straight line

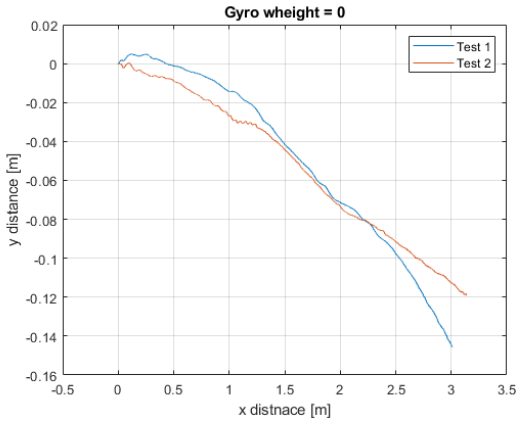


Figure 4.14: Result from the robot driving three meters in a straight line, with gyro weight = 0

From table 4.2, the distance error varies between 2.2 and 17.8 centimetres. On average, the tests where low gyro weight values are used gives the least accurate position. The most accurate position estimates, on average, is when the *gyro Weight* equals 0.6. The best position accuracy in a single run, is found when the gyro weight is equal to 0.4 (test 2), with an error distance of 2.2 centimetres. The greatest contribution to the error distance comes from the error in the y-direction, rather from the x-direction. In the x-direction, x_{error} varies from -13 to 1.2 centimetres, compared to the y-direction, where y_{error} varies from -12.8 to 14.5 centimetre. It can also be seen that the error in y-position is greater than, or equal to, the error in x-direction in 10 of the 14 tests. The robot heading was extracted during the driving test. This data is seen in figures 4.15 and 4.16. A summary of the results are written in table 4.3.

Gyro weight	runn	x	y	x_{error}	y_{error}	Distance error	Average
0	test 1	3.007	-0.145	-0.007	0.145	0.145	0.1615
	test 2	3.133	-0.118	-0.13	0.118	0.178	
0.2	test 1	3.017	0.113	-0.017	-0.113	0.114	0.1210
	test 2	2.991	0.128	0.009	-0.128	0.128	
0.4	test 1	3.072	-0.142	-0.072	0.142	0.159	0.0905
	test 2	2.988	-0.019	0.012	0.019	0.022	
0.5	test 1	3.052	0.023	-0.052	-0.023	0.057	0.0760
	test 2	3.056	-0.077	-0.056	0.077	0.095	
0.6	test 1	3.023	-0.032	-0.023	0.032	0.039	0.0470
	test 2	3.035	-0.043	-0.035	0.043	0.055	
0.8	test 1	3.067	0.038	-0.0670	-0.038	0.077	0.0530
	test 2	3.025	0.01	-0.025	-0.01	0.029	
1	test 1	3.05	-0.076	-0.045	0.076	0.091	0.0880
	test 2	3.06	-0.06	-0.06	0.06	0.085	

Table 4.2: Data measurement from Optitrack during the three meter test, unit [m]

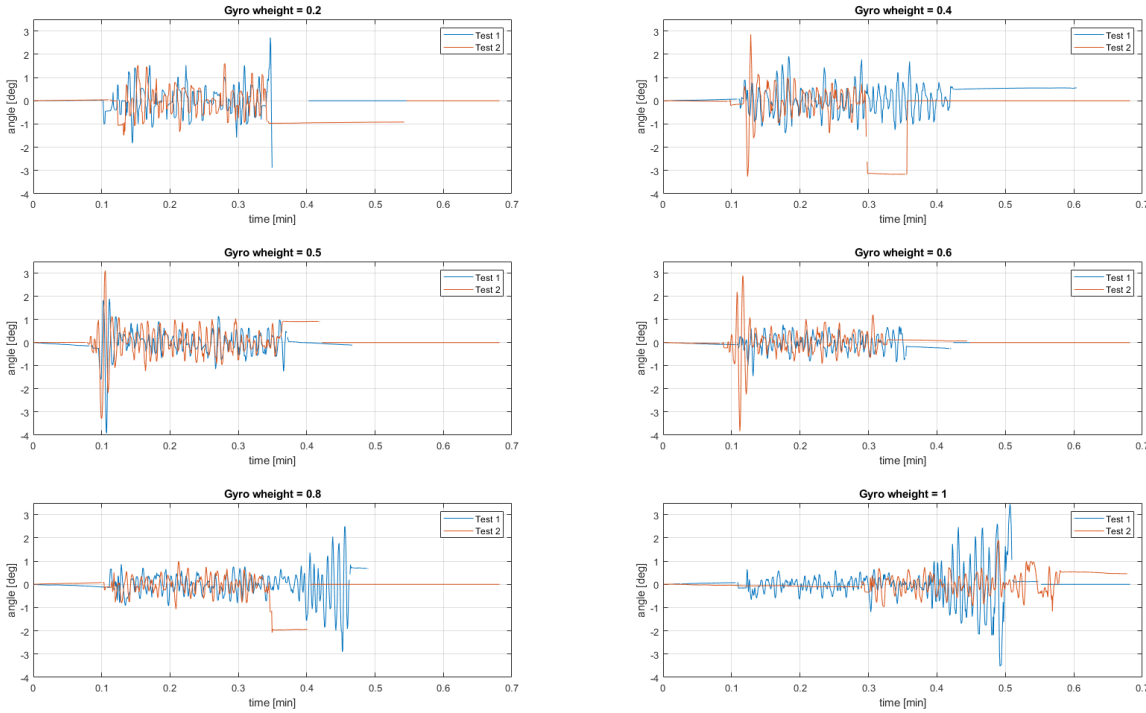


Figure 4.15: Predicted robot heading when the robot is driving

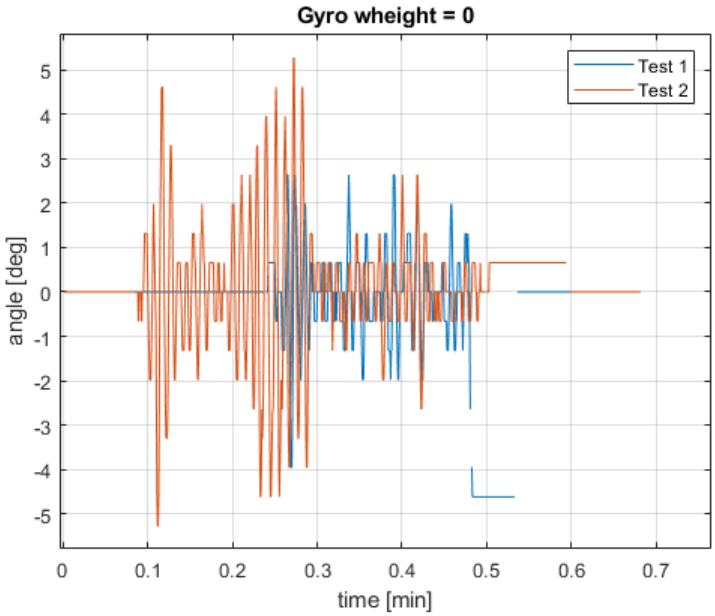


Figure 4.16: Predicted robot heading when the robot is driving, with gyro weight = 0

In table 4.3, it can be seen that the result varies more than the test where the robot was standing still. There is no clear correlation between increasing gyro weigh and final robot heading. On average, the best result was when the gyro weight was 0.6, this is the same result found for the position (shown in table 4.2).

	Robot heading	
Gyro weight	Test 1	Test 2
0	-4.62	0.66
0.2	-2.88	-0.97
0.4	0.52	-3.14
0.5	0.91	-0.03
0.6	0.39	-0.2
0.8	-1.93	0.70
1.0	0.54	0.49

Table 4.3: Final robot heading after driving 3 meters, given in degrees

4.3 Sensors

The sensors are tested to determine if the inputs used to calculate the position, makes it feasible to determine it accurately. If the incoming sensor data is corrupted, it is harder to calculate an accurate position, without additional operations to compensate for the error in the sensors. And in some cases, the error can not be compensated for in a satisfactory manner.

4.3.1 Gyroscope

As described in chapter 3.3.1, four tests will be executed to validate the accuracy of the gyroscope sensor data. The tests were executed as planned.

Drift in the gyroscope

The first test was to detect if there is drift in the gyro measurement. The execution of the test went according to the original plan. In the result, some drift and noise in the sensor measurements are observed, see figure 4.17. Three of the logs have relatively little drift, about 0.005 degrees, compared to the first test, which has a drift of 0.02 degrees after 30 minutes. Some individual spikes can also be seen in the measurements, but non are greater than 0.025 degrees. These spikes could be caused by external factors, such as sudden vibrations in the environment.

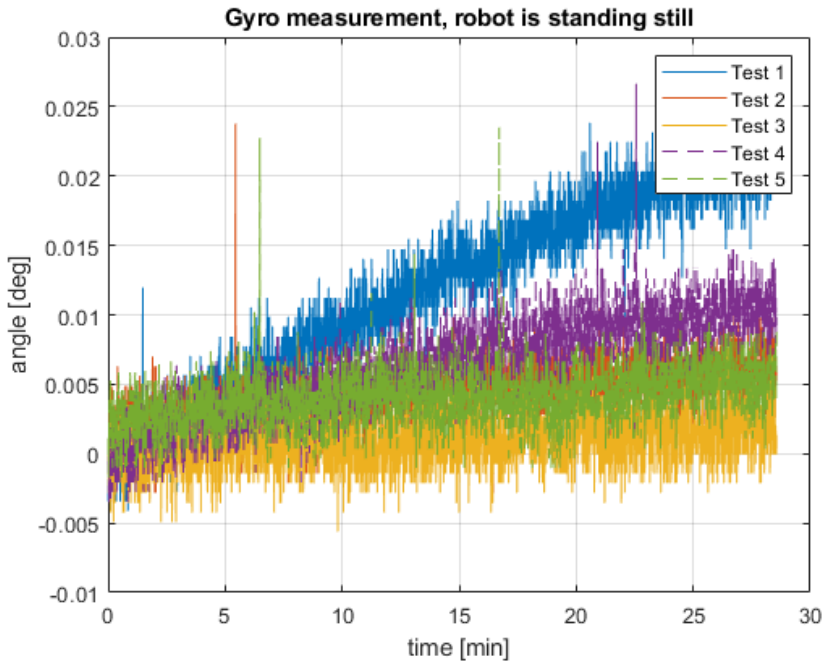


Figure 4.17: Data from the gyroscope, while the robot is standing still for 30 minutes

Noise in the gyroscope

A sensor can be affected by noise. Therefore, we can see from measurements how much the noise corrupts the data. One test can be seen in figure 4.18. The data from the other tests is found in appendix B.5, figure B.10. The variance of each test was calculated. The result is found in table 4.4.

	Test 1	Test 2	Test 3	Test 4	Test 5
Mean	0.0348	0.0477	0.0350	0.0405	0.0140
Variance	0.0021	0.0023	0.0022	0.0022	0.0023
Standard deviation	0.0458	0.0479	0.0471	0.0467	0.0475

Table 4.4: Calculated variance and standard deviation of the gyro measurement [deg]

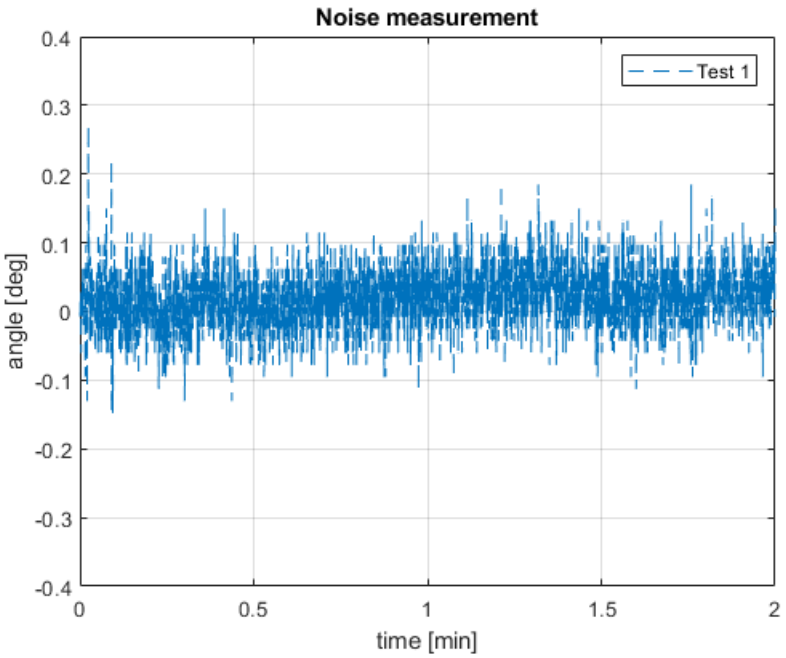


Figure 4.18: Result of gyroscope noise

Bias

The bias from the gyroscope was extracted with the robot connected to the server. This test was executed 20 times, shown in figure 4.19. In total 18 out of 20 tests lie between -5 and -6 dps. Two times the bias was measured with an increase of over 300%.

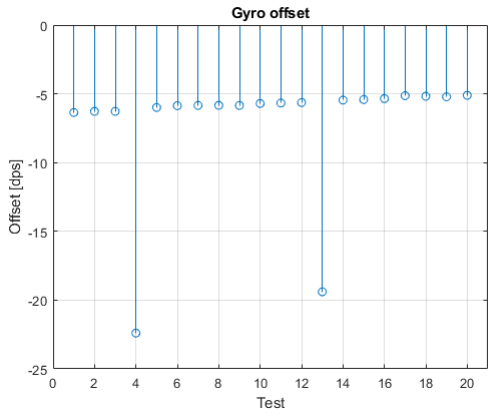
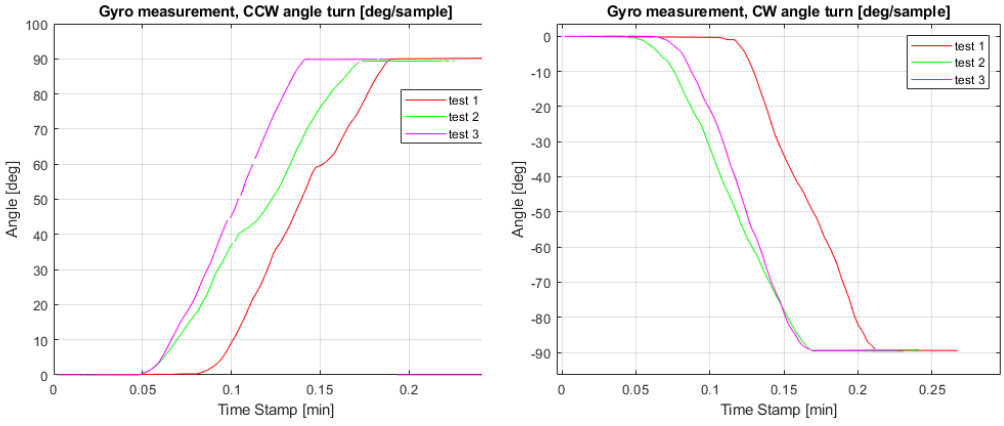


Figure 4.19: Bias test: Offset values from 20 separate tests

Accuracy

For testing if the gyroscope can detect a 90-degree rotation, the test was executed as described in section 3.3.1. The result from the 90-degree test is found in figure 4.20. The result shows the robot heading to be 90 degrees ± 1 degree. The exact values from the test are shown in table 4.5.



(a) Counter clockwise

(b) Clockwise

Figure 4.20: Result from the gyroscope test with 90 degrees turn

	CCW		CW	
number	start	end	start	end
1	0.008	89.95	-0.004	-89.40
2	-0.00006	89.35	0.010	-89.32
3	-0.001	89.86	-0.0124	-89.59

Table 4.5: Detailed result from the 90 degree turn test

4.3.2 Accelerometer

The accelerometer has not been used in the robot project previously. Extracting the sensor data when the robot was standing on a table, gave the result found in figure 4.21.

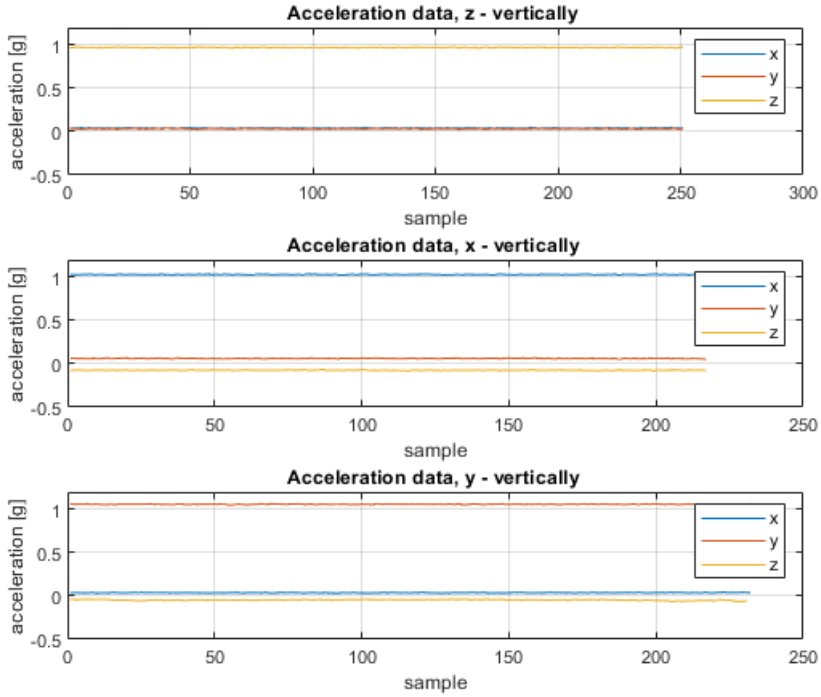


Figure 4.21: Acceleration data when the robot is at rest, with various rotations

Table 4.6 shows the mean value from the data in 4.21. As seen, the acceleration data in the z-direction is not 1, and the values in x and y directions are not equal to 0. This shows that the sensor is not mounted horizontally on the robot. The results are similar for all positive axes.

Vertical axis	X	Y	Z
x - measurement	**1.032**	0.029	0.029
y - measurement	0.062	**1.049**	0.038
z - measurement	-0.089	-0.041	**0.968**

Table 4.6: Measurement from acceleration test, with the different orientations of the accelerometer

From these results, it became desirable to determine what position the robot should have to get the sensor data to be correct. By carefully lifting the support wheel slightly, the robot was tilted forward and the sensor data became as expected. This is shown in figure 4.22. The figure shows one test where the robot is standing still on a horizontal surface and one test where the robot was tilted. The result is an improvement when tilted, which may indicate that either the IMU is attached incorrectly or the robot is not horizontal when resting on the support wheel.

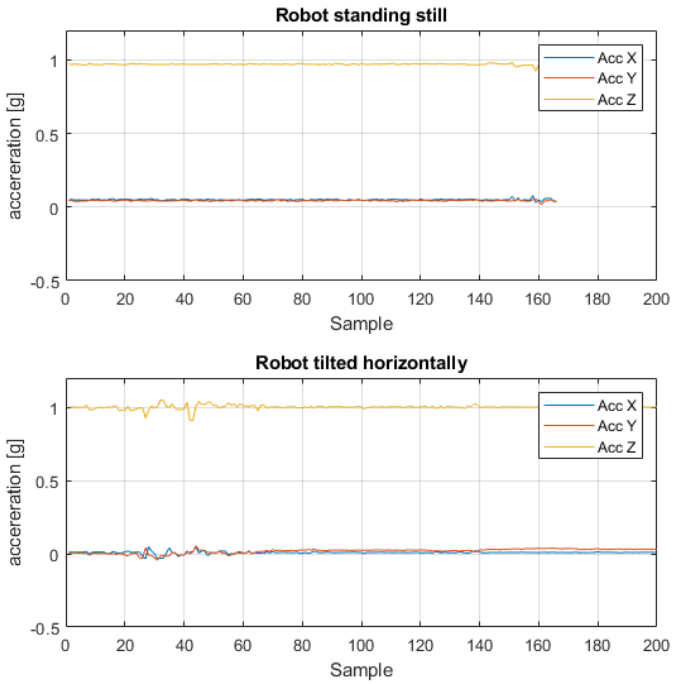


Figure 4.22: The improvement of the accelerometer data when the robot was tilted to be horizontal

Also, the acceleration has been tested when the robot drove forward. The result is seen in figure 4.23. The execution of this test, ensured logging of data had started before the robot accelerated. It started accelerating approximately three seconds into the log, in figure 4.23 it is indicated by a red line. The acceleration is noticeable as a small spike in

the measurement. The spikes have different amplitude for the three tests logged in figure 4.23. When the robot is driving forward, ripples can be seen in the measurements. The results show that it is difficult to measure when the robot start to move forward. This is expected, as the result from when the robot is standing still indicated that the robot, or the IMU, is not horizontal. Finally, it can be seen that the magnitude of the noise in the signal is close to that of the quantity measured. Finding the position through integration would therefore most likely give a significant integration error.

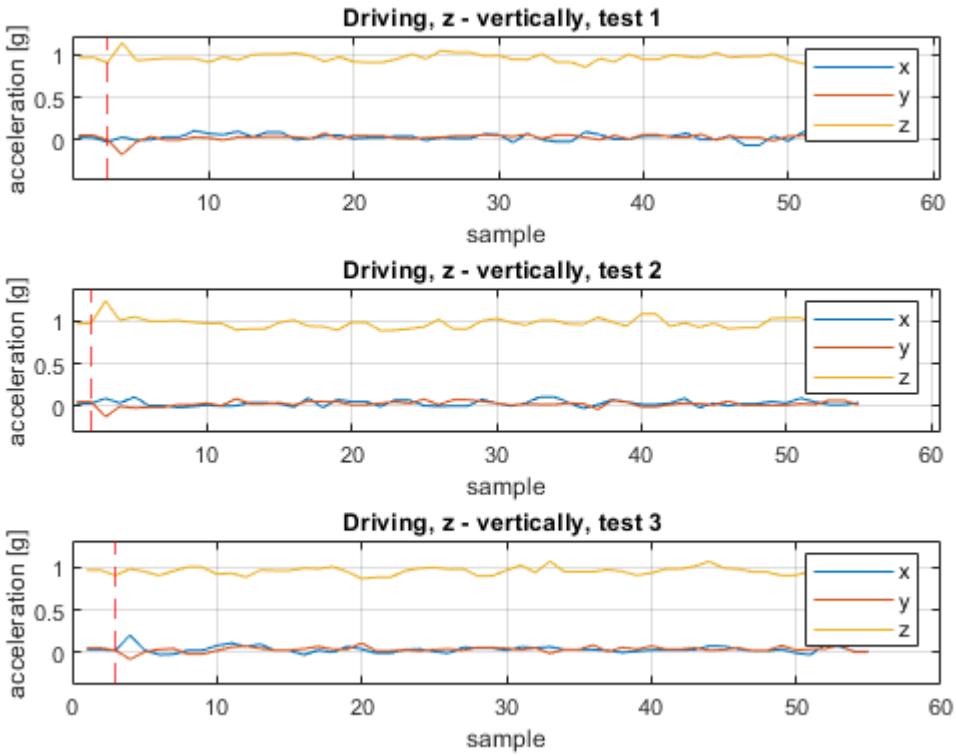


Figure 4.23: Acceleration results when the robot moves forward

4.3.3 Encoder

The result was extracted with the robot application functions *vMotorEncoderLeftTick-FromISR(...)* and *vMotorEncoderRightTickFromISR(...)*. The results from when the wheel was manually rotated can be seen in figure 4.24 and table 4.7. The encoder ticks were also found by calculations with the robot driving a known distance. This, as well as the resulting wheel factor, is found in table 4.8. The calculation of the wheel factor was done according to equation 3.33.

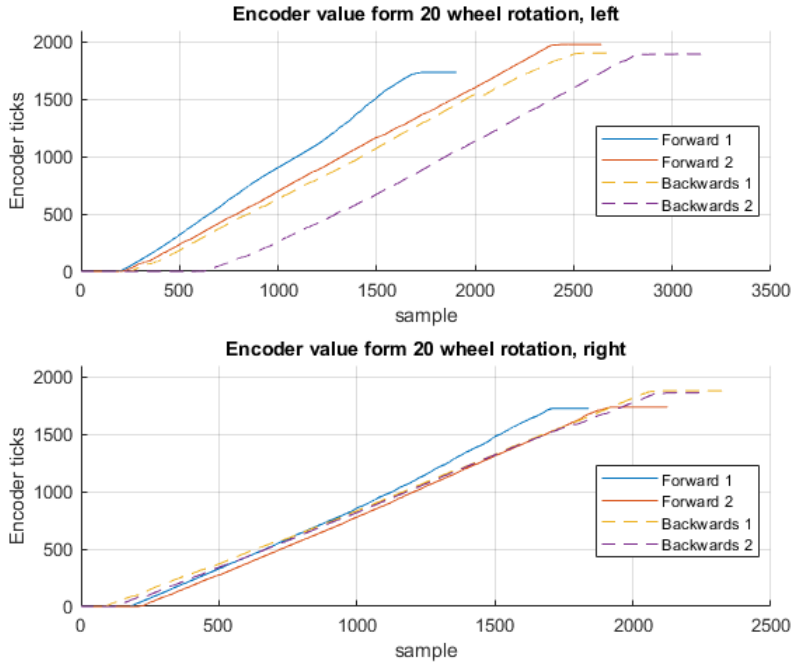


Figure 4.24: Encoder ticks from rotating the wheel 20 times

The total encoder value from each test is presented in table 4.7. The amount of encoder ticks per revolution, found in the manual wheel rotation test, varies by twelve. Resulting in 86 to 98 encoder ticks per round, meaning 3.6 to 4 degrees of angular resolution. When the robot drives one meter straight forward, the result shows that the encoder tics vary between 534 to 542, seen in table 4.8. The resulting value then varies from 103 to 109 ticks per wheel rotation, giving a resolution of 3.3 degrees per tick.

	Total ticks		Ticks		Wheel factor	
Direction	test 1	test 2	per round			
Left Forward	1762	1977	88	98	2.3182	2.0816
Left Backwards	1901	1892	95	95	2.1474	2.1474
Right Forwards	1724	1736	86	87	2.3721	2.3448
Right Backwards	1878	1860	93	93	2.1935	2.1935

Table 4.7: Total encoder tics found from 20 manual wheel rotations

	Total ticks	Distance [cm]	Tick per round	Wheel factor
Left test 1	534	102	103.94	1.9065
Left test 2	539	101.5	108.3	1.8831
Left test 3	539	102	107.8	1.8924
Right test 1	535	101	108.059	1.8879
Right test 2	539	102.5	107.2741	1.9017
Right test 3	542	106	104.3	1.955

Table 4.8: Result from finding encoder values from driving

After discussing the results for *encoder ticks per wheel rotation*, with fellow students working on the robot project, the result of around 100 ticks per rotation seemed wrong compared to what others had found. As others had found a value closer to 200, it was decided to investigate further. The result was that the already existing function *vMotorEncoderLeftTickFromISR(...)* and *vMotorEncoderRightTickFromISR(...)*, which divided the ticks by two before returning the result, were modified to no longer perform that division. The *driving test* was then executed again, as this is the test that was considered most reliable, and least susceptible to human error. The result was that a higher value for encoder ticks per revolution was found, and a new calculation of the wheel factor was done. All the new results were between 224 and 225, all the numbers are shown in table 4.9. Extensive testing was not possible for the new wheel factor, as this error was found late in the project.

	Test	Total encoder ticks	Encoder ticks per round	Wheel factor
Left	1	4491	224.55	0.9085
Left	2	4499	224.95	0.9075
Right	1	4499	224.95	0.9075
Right	2	4496	224.80	0.9075

Table 4.9: New encoder ticks and calculation of wheel factor

4.3.4 Compass

The compass was tested and then calibrated as described in section 3.3.4, except that the plan was to use the motors to rotate the robot. An unexpected problem was that the robot rotated too fast to log a good dataset. The speed of the motors was then lowered, which resulted in the robot not being able to rotate smoothly. It was therefore decided to rotate the robot by hand. This was done by placing the robot on a book, then rotating it 360 degrees. After the logging the compass data the offset was calculated using equations 3.34 and 3.35. The compass offset used was $x\ offset$ = -456 milligauss and $y\ offset$ = 246 milligauss.

Figure 4.25 shows the measured compass output and the result of the theoretical calibration on the same test. In the last test, the new calibration was tested on the robot. The offset was set in the robot application, and the robot was rotated 360 degrees as before. The result is shown in figure 4.26. Here it is seen that the new calibration drastically improved the compass measurement.

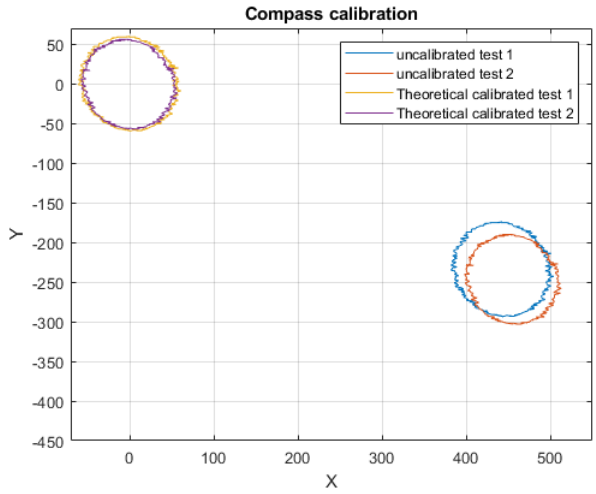


Figure 4.25: Compass measurement and theoretical improvement with the calibration

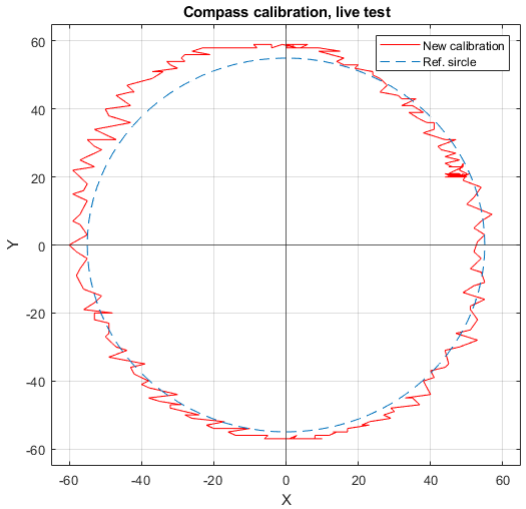


Figure 4.26: Compass measurement with the implemented calibration

4.3.5 IR measurement

The curve fitting method for the IR sensors was executed as described in section 3.3.5. The robot was put at a fixed distance from the object, and the voltage was read. On the oscilloscope, three data points were logged. The data is shown in appendix B.6. This result was loaded into Matlab and the Matlab *curve fitting* toolbox was used to calculate the coefficients a and b. The curve fitting was done by inserting the distance (as Y data) and voltage (as X data) from the data set and selecting the general mathematical model of a power function, $f(x) = a \cdot x^b$. The result of the curve fitting is found in table 4.10.

The result of the curve-fitting is shown in figure 4.27. Here, the new calibration is overlaid onto the original datasheet, the curve is similar to what is shown in the datasheet [30, figure 5]. The theoretical error of the new IR calibration is also calculated. This is found in figure 4.28.

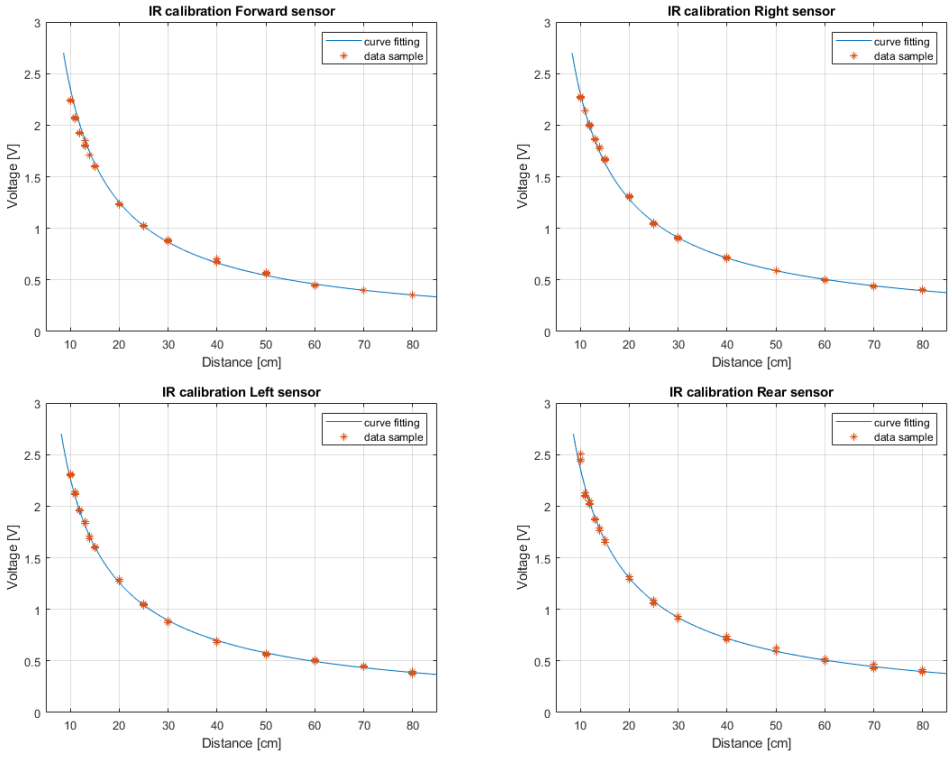


Figure 4.27: Curve-fitting from IR sensors, with coefficient from table 4.10

Coefficient	a	b
Rear sensor	27.26	-1.164
Right sensor	26.82	-1.179
Left sensor	26.23	-1.177
Forward sensor	25.61	-1.097

Table 4.10: Result from curve fitting using the Matlab toolbox

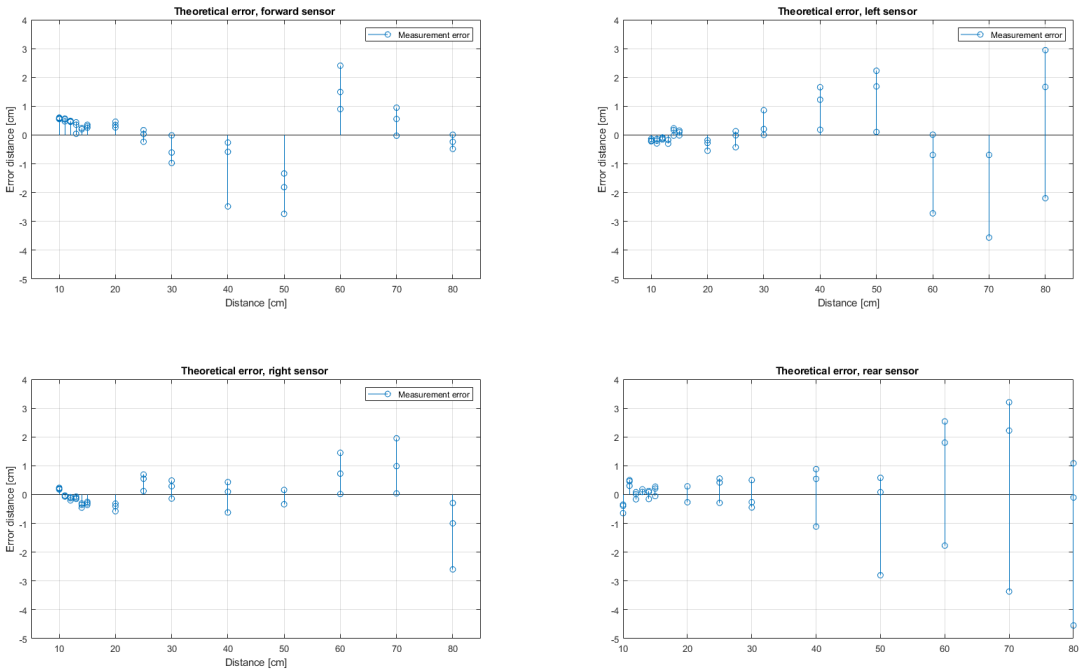


Figure 4.28: Theoretical error from the new calibration

The method described in section 3.3.5 was executed again to compare the new and old calibrations. This time printing the results over USB. By doing this, it was possible to have data from both calibration methods at each distance. The measured distance is found in table 4.11.

To compare the results, and evaluate which calibration is the most accurate, the calculated distances from the IR sensors are subtracted from the actual distance. By doing this, the error from the IR measurement is found, and a comparisons between the calibration methods done. The result is seen in figure 4.29.

	10	15	20	25	30	40	50	60	70	80	cm
New FrW	10	15	19	25	29	40	48	57	64	70	
New Right	10	14	19	24	29	39	47	55	62	72	
New Left	9	13	18	23	28	38	47	57	62	70	
New Rear	9	15	19	25	29	38	47	53	61	69	
Old FrW	11	16	22	28	33	46	55	65	71	79	
Old Right	11	15	21	25	30	41	51	56	64	71	
Old Left	10	15	21	25	31	41	51	60	65	71	
Old Rear	10	15	21	26	31	40	49	56	62	68	

Table 4.11: Result of IR calibration method

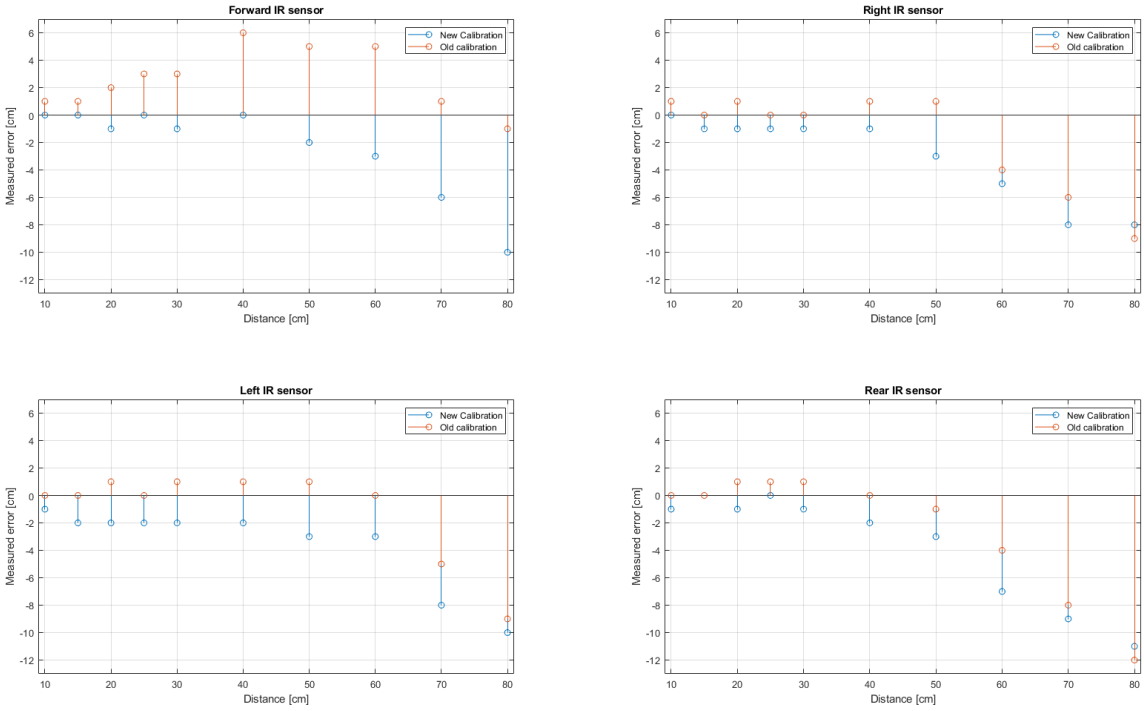


Figure 4.29: Error from measured distance and actual distance from the IR sensors

The new calibration measures the distance a bit shorter than it actually is, while the old calibration outputs a distance that is too large. For both calibration methods the error increased, as the distance became larger. This matches what was predicted, looking at the theoretical error (see Fig. 4.28).

Chapter 5

Discussion

The objective of this master thesis has been to investigate the Arduino robot's estimation error. In this chapter, the method, execution and result of the tests will be discussed. Firstly, the driving accuracy in the square test, and navigation precision when the robot maps different tracks, has been evaluated. Further, the discussion goes more in-depth into the system by looking at the data processing in the robot application. In the end, the sensors, the fundamental building blocks for the POSE estimation and object detection, are evaluated.

5.1 Driving performance

5.1.1 Square test

The square test is a method that is often used in the Robot project to examine driving performance. An advantage of this test is the manual control of the robot and the control over the position target. In addition, the method can be supervised using the Optitrack system. The measurement of driving distance with Optitrack is more accurate than manual measurements. When using the Optitrack system, some post-processing of the data must be performed. For example, the robot might be placed with a heading that is not parallel to the x-axis of the Optitrack system. This is accounted for in the post-processing by making a vector to compensate for the heading direction. All data in the set is then calculated with respect to the direction of this vector.

In this master thesis, the square test was used to find which setup gave the best position estimate. The test could have been executed more than two times to get a larger data set. The test results showed that the position estimate of the robot is more accurate without the use of the error factor on the gyroscope or encoder.

During the execution of the square test, the robot would, on a few occasions, not drive forwards. At the same time, the motors made a buzzing sound. It could be due to the bat-

tery being discharged. The robot was then connected to the charger, but would not charge, a sign that the battery is close to fully charged. After this, the robot was turned on, and the driving performance was back to normal. This happened four times during the square tests. This bug has not been possible to reproduce, and it is not clear why it happened. A hypothesis is that the robot's initialisation had not been successful. Another possibility is that the applied voltage to the motors was too low or that there are one or more unstable hardware connections in the robot.

5.1.2 Continuous square test

This method aims to evaluate eventual changes in the position estimate over a period of time. In the method's description, the target is described as the robot driving four to five rounds. This gives a total driving distance of 16 to 20 meters and an operation time of three to four minutes. A larger data set could be generated by executing the continuous square test more times, and would get a clearer trend in the development of the position error. The average error in each corner varied between both rounds and tests. The tendency for the CW error was increasing more rapidly, compared to the CCW direction. Since the average distance error changes over time, a good prediction for how the position error develops has not been found.

To further investigate how the error develops over time, it is desirable to look at the net error. This net error describes the increase in error from round to round, between each time the robot passes the same target corner. For each round in the continuous square test the equation 5.1 is used to find the net error. Where x_p and y_p are the coordinates of the robot in the previous round. The coordinates x and y are the position of where the robot was in the current round. The i is in range 1-4 and indicates which corner that is under investigation.

$$e_{net} = \sqrt{(x(i) - x_p(i))^2 + (y(i) - y_p(i))^2} \tag{5.1}$$

The data is stored in table B.1 in the appendix and is visualized in figure 5.1 and 5.2. From the net calculation, it is seen that the net error from each round is not constant, seen in both the CCW and CW driving directions. In the CW direction, the net error will probably increase every round. In the CCW direction, the net error is more steady, but the total error will grow.

The reasons for this increasing error can be noise or faulty sensor measurements from the gyroscope and the encoders. Alternatively, the software calculates the POSE wrongly or there is another fault in the software design. All of these can be reasons for the error in position varying over time.

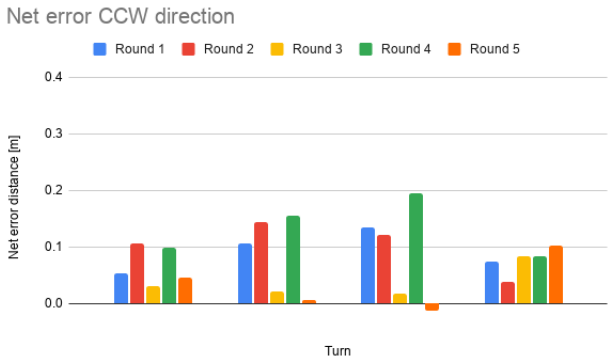


Figure 5.1: Net error for every round in the continuous square test, CCW direction

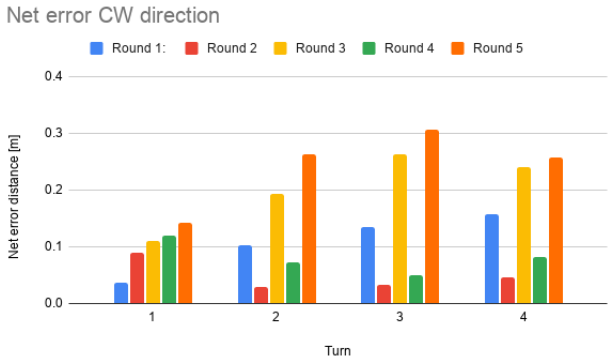


Figure 5.2: Net error for every round in the continuous square test, CW direction

5.1.3 Round court

The robot was tested in the round court, to confirm the robot's ability to detect, and map, slopes. Both this and the larger track method provide challenges for getting good quantitative results. The objective measurements from Optitrack cannot be directly compared with the measurements from the robot. The Optitrack system and the robot do not have the same coordinate systems, or the same time stamping of data. All this results in a challenging and complex post-processing, for the test to yield more quantifiable results. Contrasted against the square test, where quantitative data can easily be processed and evaluated.

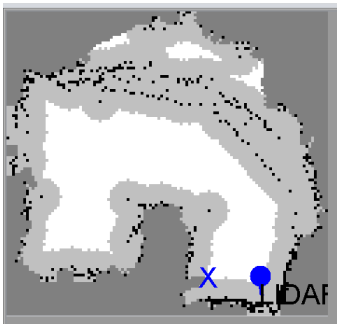
The round track was navigated both with and without the docking station inside. As the test was carried out only once, the data set was small and the same outcome can not be guaranteed in future tests. From previous tests during the master thesis, it is seen that the robot can behave somewhat stochastic. The round track test concludes that the robot can

navigate the court without crashing, and the mapping of sloped obstacles works, which was the point of this test.

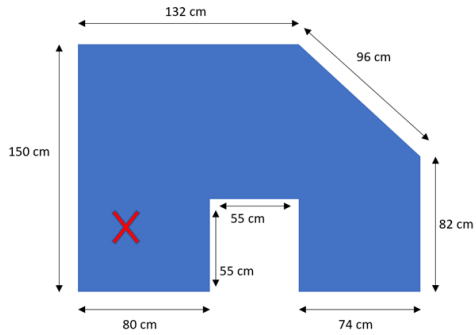
5.1.4 Larger track

The main goal of the robot project is to map an unknown environment, using an autonomous robot. Therefore, the robot has been tested in a larger court. This benchmarks the robot's driving and navigation performance with the initial robot application. The method described in section 3.1.4 describes a court with multiple corners and one sloped wall. The track has different elements that tests the robot's driving performance. An improvement could be varying the design of the court between runs, to see if some courts are easier to navigate than others.

In the execution of this test, the robot was placed on the same starting position each time. The robot could have been initially placed at different starting positions, to see if this affected the mapping. The result of the tests are deviating, as the mapped result from the GUI did not have the same shape as the testing court (see figure 5.3). Note that the red 'X' in figure 5.3b is the starting position of the robot. The blue 'X' in figure 5.3a is the next position target for the robot, given by the server. A *drifted wall* can clearly be seen (figure 5.3a), while not all walls are clearly mapped with a black line. The final result is an unsatisfactory mapping of the environment.



(a) Server GUI map



(b) Model of the testing track

Figure 5.3: Comparison of the larger tracking test

5.2 Position estimation

Even with good sensor data, bad software application design can worsen the accuracy of the final position estimate. Today the robot uses the gyroscope and the encoder data, whereas at the start of the project, the robot only emphasized the gyroscope data when turning.

5.2.1 dTheta

Testing dTheta, the fusion of encoder and gyroscope measurements, with different weighting of the gyroscope and encoder, has allowed a visual result of the robot heading. The collected data set could be improved by running five or more tests with every gyro weight. The method could be improved by selecting the gyro weights with smaller step size, e.g., a step size of 0.05, giving a set of 0.05, 0.10, 0.15 etc. up to 1.

The execution of the test went as planned. An improvement would have been to extract both dTheta and predicted robot heading from the robot application. The advantage of extracting both variables would have been comparing both sets of data from the same test run. The data could then have been used to conclude how the error in the dTheta affects the predicted robot heading. With the extraction of dTheta and robot heading separated into different runs, it is not accurate to compare these results directly. Nevertheless, it can still be used to indicate a connection.

The results from when the robot is standing still, show that the gyroscope introduces noise in dTheta, seen in figure 4.7 in 4.2.1. In an ideal case, when the robot is standing still, the value should have been zero. The only test that showed this result was when only the encoder ticks were used in the calculation of dTheta. No voltage will be sent to the motors when the robot is standing still, resulting in no encoder ticks being generated. On the other hand, when driving forward, the fluctuations of dTheta are largest when the robot uses only the encoders. Regarding the precision in the robot's orientation, it can be concluded that it is best to use the encoder when the robot is at rest. When the robot is driving forward, a combination where the gyro is weighted higher than the encoders seems to work best.

5.2.2 Robot heading

In the tests for observing the heading, where the robot is driving forward, an important criticism is that the control system is not taken into account. The control system does not use the heading as a control variable for the target point. The robot only concerns itself with the x and y position, rather than which heading it has. This means that the robot's heading at the end of the test is not a factor for the control module. At the same time, the robot should, in theory, have the same heading in the ending, as in the beginning, since the robot is only driving straight forward. In this method, the control module's impact has not been considered, which could have affected the result of the heading. The control module cannot be neglected and can be the reason for some of the spikes in the results. The execution of the test matched the planned method. The results show that the robot heading drifts more using the gyroscope than using the encoders. This can be caused by integration of the noise introduced into dTheta by the gyroscope. The heading oscillates in all the tests where the robot moves forward. The largest spikes are found when only the encoders are used. This matches what is observed and concluded regarding dTheta, that a setup where the gyroscope is combined with the encoders significantly improves the heading estimate, when the robot is driving.

5.3 Precision in sensors

All tests in this section were executed at the office. The first tests were to investigate each sensor. Some sensors are calibrated to improve accuracy. A general problem was an unsteady connection between the robot and the server. The connection ended when the other students at the office tested their robots over BLE. To bypass this problem, the robot was connected by USB to print over Putty, rather than using the server.

5.3.1 Gyroscope

Drift

The first test was done to look for tendencies in the drift. The method is described in section 3.3.1, where the robot is standing still for 30 minutes. An improvement of the method would have been to calculate the trend line for the drift. An estimate of the drift without the effect of noise and bias, can be found by looking at the slope of the trend line.
It can not be concluded from this data alone, why one test-run gives a larger drift than the rest. One possibility is that the offset calculated in the beginning is incorrect and that this error accumulates throughout the 30 minutes of the test. This can be investigated by repeating the test, with the addition of extracting the offset, and looking for a correlation between the drift and the offset.

Noise

The gyroscope was also tested to look at noise on the signal. The five tests show the corruption of the sensor data. To compare the tests, the mean, variance and standard deviation were calculated. The variance, and standard deviation, is almost identical in every test. This can be interpreted as the noise having the same effect on the sensor throughout all the tests. The tests show the noise in the measurement when the robot is stationary.

The question of where the noise is introduced into the measurements arises. Many different sources of error can have an impact. If the sensor is not rigidly fastened, vibrations can affect the signal. When a sensor is attached, the sensor can be exposed to mechanical stress, as explained in the datasheet [34]. The sensors offset must be found to compensate for the mechanical stress. In the gyroscope, this is done by finding the bias offset. Inside the MEMS gyroscope the measurement itself can also be affected by noise. The analogue signal will be quantized by the ADC (inside the gyro). This operation will introduce a quantization error (and other errors such as INL) [57, chapter 4.3]. After the signal is quantized, the digital signal is represented by bits. Bit-flips that will corrupt the data is less prevalent than noise on analogue signals. It has not been possible to quantify each of these factors, but the primary effects are assumed to be in mounting of the sensor, and inside the gyroscope. A visualization of how, and where, the different errors can enter the signal chain is shown in figure 5.4.

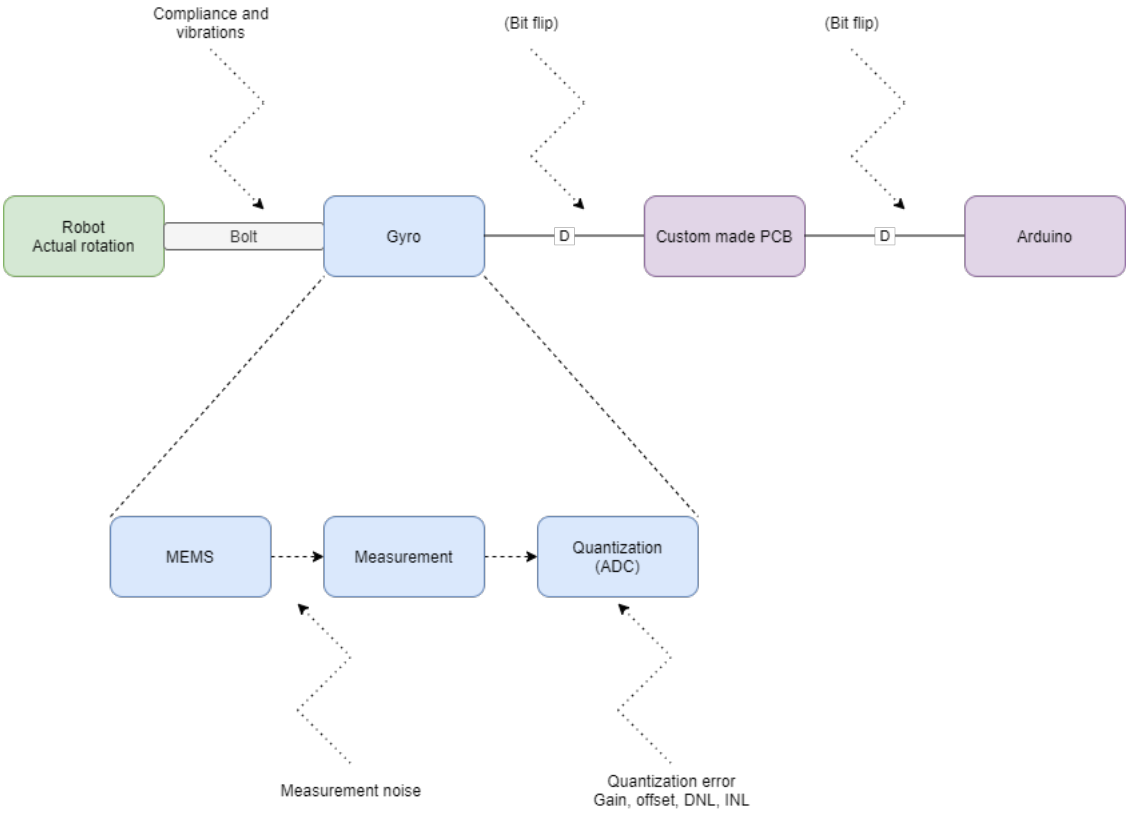


Figure 5.4: Conceptual diagram of potential sources of error in the gyroscope

An improvement to this method is to test the gyroscope for noise when the robot is driving. When the robot is running forward, noise can be introduced from the robot's motors or other hardware, such as rotation of the sensor tower. This can easily be tested with the same test setup as described, with the addition of activating the sensor tower.

Bias

From the bias test, it is seen that the gyroscope offset can vary. The bias test results show that the offset lies between -5 and -6 dps. In two cases, the offset was found to be -19 and -22. This is an increase in the gyro offset by over 300%. How large impact this offset has on the position estimation can be investigated further by evaluating the offset value with the corresponding square test of the robot. By doing this, the offset's impact can be directly seen in the performance of the square test. In the bias test execution, the robot was moved, rotated, and tilted before a test was executed. This was done to simulate the robot being moved to the testing site. This has had no impact on the gyroscope measurement. It can be concluded that the gyroscope is not broken, but the calculation of the offset can, in some cases, be wrong.

Accuracy
In the fourth test, the accuracy of detecting a 90-degree turn was executed. One potential improvement in the method would have been securing the robot to something that rotated only 90-degrees. This could remove the inaccuracy in the manual rotation. A challenge in the execution was rotating the robot around a point, that was only marked on the table. This may have introduced translation into the intended rotation. A solution to this is to have a mechanically fixed point of rotation. The robot was also placed on a notebook, instead of a book with a hard cover, which might have reduced the precision of the rotation.

The turning of the robot was executed three times in each direction. It can be argued that the test should have been executed more than three times to get a better average of the results. The test results show that the gyroscope can detect a 90-degree turn with an inaccuracy of less than one degree, in both the CCW and CW directions. There are some uncertainty connected to this result, but all the tests show the same result and is therefore believed.

5.3.2 Accelerometer

The method described in section 3.3.2 was executed according to plan. Based on the results of this, it can be concluded that the accelerometer is working well. When extracting the sensor data, during driving, the support wheels in the rear of the robot should have been larger. If the robot had a larger supporting wheel, the IMU would have been tilted to be horizontal, removing some error from the measurement. When the robot was standing still and manually tilted horizontally, the accelerometer data was as expected. The value for acceleration in the x- and y-directions equaled zero g, while it in the z-direction equaled one g.

As the IMU is not mounted horizontally, a small error can be seen in the data when standing still or moving. The offset is measured to have an average of $0.04\ g$, and is listed in the datasheet [34, p. 20]. This offset equals $0.4\ m/s^2$. One could naively try to integrate the accelerometer data twice to get an estimate for the travelled distance. In the 10 seconds (see figure 5.5) the robot uses to drive approximately one meter, the small offset in the accelerometer data would result in an estimated distance of 40 m. This shows how important an adjusted and calibrated accelerometer is. Because of this, it is hard to evaluate whether the sensor values can be used in the position navigation without further work.

The estimated time of 10 seconds, for the robot to drive one meter is found in data from the square test. In figure 5.5, the time versus distance in the x-direction can be seen.

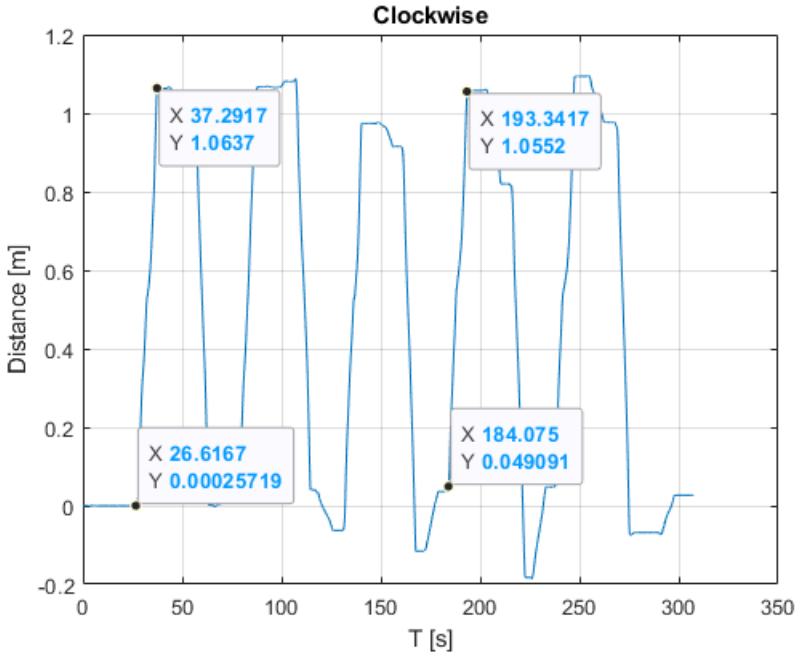


Figure 5.5: Time the robot used to drive approximately one meter

An improvement to the method of finding out how the accelerometer could have been used, would be gathering data when the robot drives with different speeds. This could have resulted in an acceleration that was easier to detect in the accelerometer. If the IMU was moved into the rotation centre, the accelerometer could have been tested again, with an expected result of zero in the x-and y-directions while turning around the z-axis (as it would not experience any linear acceleration). Alternately, the implementation of the mathematical rotation and translation of the acceleration data, described in section 3.3.2, could improve the measurement. An easier task is to calibrate the sensor. As explained in the datasheet for the IMU, it is recommended to calibrate the sensor after it is mounted, [34, chap. 4.6.2]. This can be done by extracting the offset when the robot is placed on a horizontal surface. The measurement has to be the average from a longer period of time.

In conclusion, the accelerometer itself worked well and the results were as expected. The low acceleration of the robot could pose problems for estimating the position. To use the accelerometer data in calculation of the position, the data has to be integrated twice, yielding an estimate of the distance. Today the data is too noisy to be used in such a method. As an example the integration of an imperfect signal is shown in figure 5.6. Here it is seen how the calculation will introduce a large integration error for even a small bias.

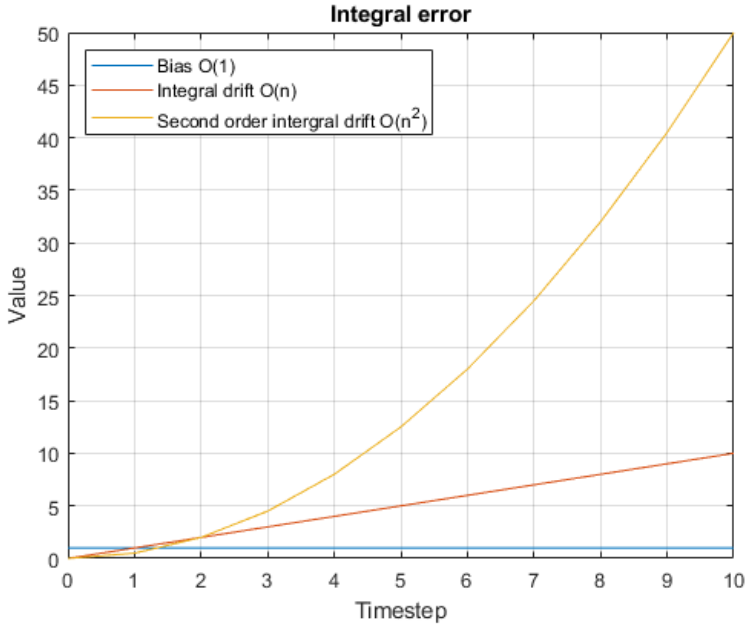


Figure 5.6: Illustration of a signal with a small bias, and the resulting single and double integral errors

5.3.3 Encoder

To find the precision of the encoder, the number of encoder ticks produced by the sensor has to be compared to the information in the datasheet. With the new motors, the number of encoder ticks were not listed. Because of this, it was determined to estimate the number of encoder ticks, and then calculate the new wheel factor.

The method in this thesis does not find the gear ratio of the motors, only the encoder ticks used in the position estimation. The method did not consider the frequency of the task, or the runtime, as these should have no impact on the results. This is because the encoder value is registered in an interrupt handler. This means that every encoder tick is registered regardless of task frequency. The two tests to find the wheel factor were executed as true as possible to the planed method. To improve the method, where the robot drives a fixed distance, one could execute the test in the camera room B333. In this way, a more accurate measurement of how far the robot has driven would have been found by Optitrack. Using Optitrack would be instead of using a measuring tape, with the lower resolution, an higher uncertainty, arising from using a coarse manual measuring tool. The other test, where the wheel is rotated by hand, looks at the average of 20 rotations. It can be argued that ten rotations could be enough and that the test should be executed more times.

In the robot application, the wheel factor had to be set. And the conclusion was to use the result from the test where the robot drives one meter. The driving test result was emphasized more because the robot behaves the same way when it navigates an environment. A conclusion on why the two tests were different has not been found. A hypothesis is that the driving test is more accurate and less prone to human error.

To get maximum resolution from the encoder, the robot application software has to be adjusted. The functions *vMotorEncoderLeftTickFromISR(...)* and *vMotorEncoderRightTickFromISR(...)*, has to be changed so that the encoder-ticks are not divided by two. A new test was done with these changes, which yielded 225 encoder ticks per wheel rotation. This comes out to a resolution of 1.6 degrees. If this solution is to be used, the wheel factor has to be updated to 0.9075.

5.3.4 Compass

During the testing of the compass, a few weaknesses in the method was discovered. An improvement to the execution can be found in the accuracy of manually rotating the robot. It is not easy to rotate the robot around a fixed centre. This may have had an impact on the result and weakened the conclusion. It is possible to have a board rotating around a fixed point and then place the robot on top. Such a solution could remove some outliers in the measurements. An greater improvement could be found by having the robot rotate slowly around is own z-axis with the motors turning in the opposite direction of each other. The movement of the robot would then be more similar to the movement during normal operation.

The result from the compass calibration shows an improvement. The new calibration centres the data around the origin, as expected for an ideal compass (shown in figure 3.10). Because of plausible errors in the manual rotation of the robot, it is not possible to see if the outliers in the data are from the test or from the sensor itself. A filter is not applied to the data, but as the signal is potentially affected by noise, this could be beneficial. Before the compass is used in the robot application, tests of noise would be preferable. Results of such a test can be used to dimension the filter. Such a test can be performed by letting the robot have a constant heading over some time, extracting the compass data, and then plotting, and analyzing it. Looking at the signal in the frequency domain, one can potentially identify the frequencies of noise in the data.

5.3.5 IR sensor

The method used for calibrating the IR sensors was curve fitting. The execution of the IR-calibration went as described in 3.3.5. An improvement to the calibration method would have been to increase the number of samples (from three) at each distance. During testing of the new calibration, a larger fluctuation of the measured distance, compared to the old calibration was also detected. This can be the result of *integer casting* in the robot application. For example, if the IR sensor is placed 12 centimetres away from an obstacle and the IR sensor's measurement is oscillating between 11.9 and 12.1 centimetres, the output of casting from a float to an integer will oscillate between 11 and 12 centimetres. Casting

from float to integers in C, happens by discarding the fractional part, behind the scene, causing this issue. In summary this is perceived as the distance measurement from the sensor fluctuating. A solution for this could be to either extract the distance in millimetres, by multiplying the data by 10, or to round the distance before the cast from a float to an unsigned integer.

The fluctuating distance measurement in the new calibration was decisive in the choice of using the old calibration. The result from the sensors are not perfect with either calibration. It is observed that the sensors have some uncertainty in the measurement, although the value for this is not given in the datasheet for the sensor [30]. The other main source of error is in the internal ADC of the ATmega. The datasheet for the ATmega 2560 gives an *Absolute Accuracy* of $\pm 3\ LSB$[1], defined by the datasheet [22] as "[Absolute Accuracy] is the compound effect of offset, gain error, differential error, non-linearity, and quantization error.". With the 2.56 V internal reference used, ± 3 LSB comes out to ± 7.68 mV. As an example using the new calibration for the left sensor, a measured voltage of 0.6987 V gives a distance of 40 cm, while a voltage of 0.6987 V + 7.68 mV = 0.7064 V gives a distance of 39.5 cm. As shown by this example, the accuracy of the ADC is not negligible when discussing precision in the IR-sensors. It should be noted that this will be worse for larger distances, as the slope of the conversion function is steeper for lower distances. The result of this calculation, and the trend explained, are shown in figures 5.7 and 5.8.

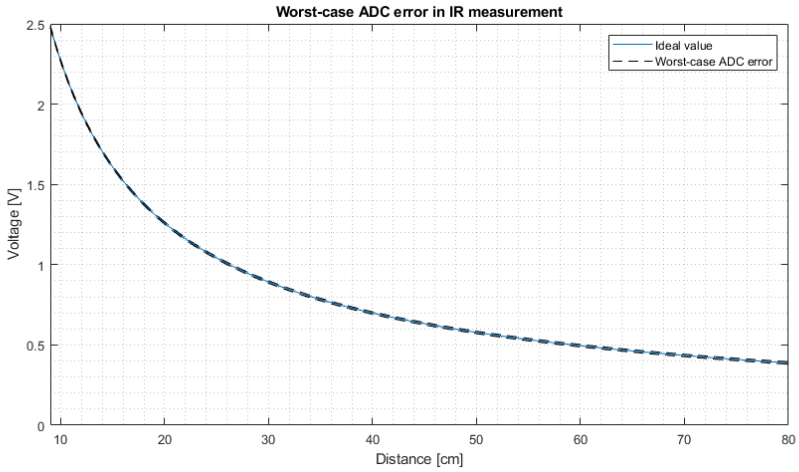


Figure 5.7: Theoretical worst-case of ADC error in the left IR sensor

[1] Found using Table 31-9 in [22], knowing that the Arduino has a 16 MHz crystal (seen in the schematic), and that the ADC is initialized with a prescaler of 16 in the application

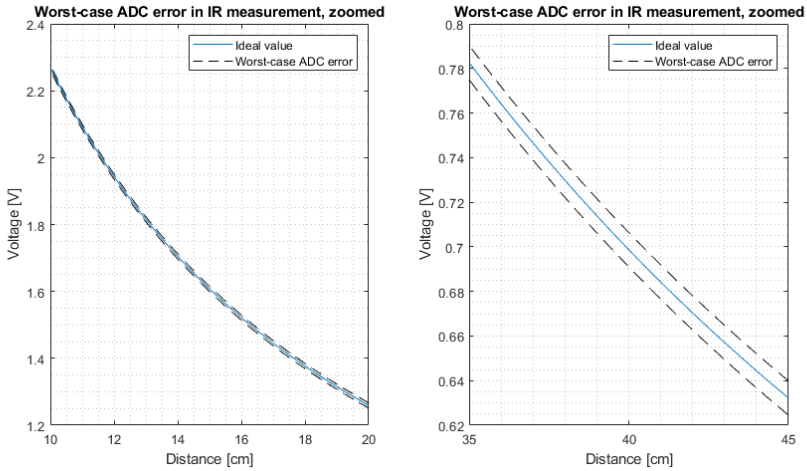


Figure 5.8: Zoomed image of theoretical worst-case of ADC error in the left IR sensor

This imprecise measurement has no impact on the position estimation today, as the IR-sensors are only used in object detection and not for estimating the robot's position. It should therefore be prioritized lower in the work to improve the position estimate. In the future, the IR-data could be used to detect the docking station, and from knowing the docking position, re-calibrate the internal position of the robot. This is saved as future work for improving the robot autonomy in the project.

It could be argued that inaccuracy in the servo, controlling the sensor tower, can impact the measured distance and position to an obstacle. The servo is seen as relatively accurate, and the sensor tower can perhaps have one to two degrees error in the angle. If the angle error in the servo were 10 degrees, the measured distance would be one centimetre wrong at a distance of 80 centimetres. Compared to the already measured error from the IR sensor, the servo is not a significant part of the distance measurement error. However, even if the measured distance is only one or two centimetres wrong, the error in angle impacts the x-y coordinates. Example, measure a distance in the heading direction to be 80 centimetres, but the sensor towers angle is two degrees wrong. The x-y coordinates are then x: 80.049 centimetres and y: 2.792 centimetres. Ergo their position is almost three centimetres wrong. If the measured distance is 40 centimetres, the error in the y-direction is 1.39 centimetre. This decrease in the position coordinates' error supports the already implemented distance restriction in the server application. Here, the server disregards measurements from the robot that are over 40 centimetres, which reduces the mapping error.

Conclusion

The conclusions from this chapter are found in the beginning of the thesis, at page v.

Chapter 6

Further work

Through the work on this master thesis, several thoughts on future assignments for the project have emerged. The tasks are grouped into the improvement of the robot's position estimate, and making the robot able to behave more autonomously.

To improve the position estimation the following tasks can be looked at:

- Implement and test different filters to minimize the impact of noise in the sensors
- Move the IMU, or use calculations to transform the accelerometer data to the center of rotation, see 3.3.2
- Implement a new position estimator based on the extended Kalman filter (EKF), see 3.2.3
- Investigate the possibility of using sensor models to improve the sensor data, see [66]
- Improve the seemingly stochastic behavior of the robot, by investigating the initialisation of the robot application and its use of FreeRTOS

Increasing the autonomous capabilities of the robot can be done through the following tasks:

- Implement a new communication system based on the nRF52 dongle.
- Complete the work on battery management. Control over the battery level would allow the robot to check for low battery, and for future projects to see if low battery impacts the driving performance
- Continue development on autonomous docking. The robot could then be able to have the docking marked as a fixed point, and when seeing it again, update it's own position estimate based on that.

References

[1] Karoline Halvorsen. *Investigation of sources of error and a way to minimize the error in position estimation of an NXT robot*. Tech. rep. Cybernetic institution, Norwegian University of Science and Technology, Norway, 2020.

[2] T. Hobbes & W. Molesworth & Homer & Thucydides. *The English works of Thomas Hobbes of Malmesbury*. London, J. Bohn, 1839.

[3] Epicor. *What is Industry 4.0—the Industrial Internet of Things (IIoT)?* Last accessed 04.01.2021. URL: https://www.epicor.com/en/resource-center/articles/what-is-industry-4-0/.

[4] E. I. Al Khatib et al. "Multiple sensor fusion for mobile robot localization and navigation using the Extended Kalman Filter". In: *2015 10th International Symposium on Mechatronics and its Applications (ISMA)*. 2015, pp. 1–5. DOI: 10.1109/ISMA.2015.7373480.

[5] *Definition Dead Reckogning*. Last accessed 20.09.2020. URL: https://www.dictionary.com/browse/dead-reckoning.

[6] Håkon Skjelten. *Fjernnavigasjon av LEGO-robot*. Tech. rep. Cybernetic institution, Norwegian University of Science and Technology, Norway, 2004.

[7] *Dictionary Autonomy*. Last accessed 02.09.2020. 2020. URL: https://www.dictionary.com/browse/autonomy.

[8] Megan Wallace. "What is GPS?" In: *NASA* (2019). URL: https://www.nasa.gov/directorates/heo/scan/communications/policy/what_is_gps.

[9] Rob Matheson. "Fleets of drones could aid searches for lost hikers". In: *MIT News* (2018). URL: https://news.mit.edu/2018/fleets-drones-help-searches-lost-hikers-1102.

[10] *Room location B333 at NTNU*. Last accessed 08.11.2020. 2020. URL: https://bit.ly/3n3h8nn.

[11] Microship. *Atmel Studio 7*. Last accessed 15.10.2020. 2020. URL: https://www.microchip.com/mplab/avr-support/atmel-studio-7.

[12] Arduino. *Arduino IDE 1.8.13, Downloads*. Last accessed 01.10.2020. 2020. URL: https://www.arduino.cc/en/software.

[13] *AVRDude - AVR Downloader*. Last accessed 01.10.2020. 2009. URL: https://www.arduino.cc/en/software.

[14] *Microsoft Project*. Last accessed 17.09.2020. 2019. URL: https://www.microsoft.com/microsoft-365/project/project-management-software.

[15] Ese. "Sanntidsprogrammering på samarbeidande mobil-robotar". MA thesis. Norwegian University of Science and Technology, Norway, 2016.

[16] Sondre Martin Jensen. *Arduino-Robot med LIDAR - sensor*. Tech. rep. Cybernetic institution, Norwegian University of Science and Technology, Norway, 2017.

[17] Tor Andersen & Mats Rødseth. "System for Self-Navigating Autonomus Robots". MA thesis. Norwegian University of Science and Technology, Norway, 2016.

[18] Sondre Martin Jensen. "Autom-robot med LIDAR-sensor". MA thesis. Norwegian University of Science and Technology, Norway, 2018.

[19] Erlend Velle Dypbuukt. "Position Estimation og Arduino SLAM - Robot". MA thesis. Norwegian University of Science and Technology, Norway, 2018.

[20] *Arduino MEGA 2560*. Last accessed 16.11.2020. 2020. URL: https://www.arduino.cc/en/Main/arduinoBoardMega2560/.

[21] *Arduino Mega rev 3*. URL: https://store.arduino.cc/arduino-mega-2560-rev3.

[22] *Data sheet ATmega 2560*. URL: https://ww1.microchip.com/downloads/en/devicedoc/atmel-2549-8-bit-avr-microcontroller-atmega640-1280-1281-2560-2561_datasheet.pdf.

[23] *UK 1122, L298 H. Brige Dual Bidirectional*. datasheet. Cana Kit. URL: https://gzhls.at/blob/ldb/2/5/9/3/68f832cf5a93c55a77c3577450b56e3e7f59.pdf.

[24] *Motor Driver 2A Dual L298 H-Bridge*. URL: https://www.kr4.us/motor-driver-2a-dual-l298-h-bridge.html.

[25] *DAGU 120:1 Motor*. Last accessed 21.08.2020. URL: http://dlnmh9ip6v2uc.cloudfront.net/datasheets/Robotics/DG01D.pdf.

[26] *Hobby Gearmotor DAGU*. Last accessed 21.08.2020. URL: https://www.sparkfun.com/products/13302.

[27] *simple Encoder Kit*. Last accessed 21.08.2020. URL: http://cdn.sparkfun.com/datasheets/Robotics/multi-chassis%20encoder001.pdf.

[28] *SO50 STD*. Last accessed 07.11.2020. URL: https://cdn.sparkfun.com/datasheets/Robotics/S05NF%20STD.pdf.

[29] *ROB-10333 - SO50 STD*. Last accessed 07.11.2020. URL: https://www.digikey.com/en/products/detail/sparkfun-electronics/ROB-10333/5766904.

[30] *IR sensor*. Last accessed 19.08.2020. 2017. URL: https://www.sparkfun.com/datasheets/Components/GP2Y0A21YK.pdf.

[31] *How to Convert the Analog Signal to Digital Signal by ADC Converter*. URL: https://www.elprocus.com/analog-to-digital-adc-converter/.

[32] *Lidar v3 Datasheet*. Last accessed 20.08.2020. 2016. URL: http://static.garmin.com/pumac/LIDAR_Lite_v3_Operation_Manual_and_Technical_Specifications.pdf.

[33] *Lidar v3 image*. Last accessed 20.08.2020. URL: https://www.sparkfun.com/products/14032.

[34] *IMU Datasheet*. Last accessed 09.11.2020. 2015. URL: https://cdn.sparkfun.com/assets/learn_tutorials/4/1/6/DM00133076.pdf.

[35] *3-Axis Digital Compass IC HMC5883L*. datasheet. Sparkfun. URL: https://cdn.sparkfun.com/datasheets/Sensors/Magneto/HMC5883L-FDS.pdf.

[36] Franco D. "Earth's magnetic field and its changes through time". In: 116 (2020). Accessed 22.11.2020. URL: https://researchoutreach.org/articles/earths-magnetic-field-changes-through-time/.

[37] S. R. Nilssen. "Autonom karlegging av labyrint med Lego robot". MA thesis. Norwegian University of Science and Technology, Norway, 2018.

[38] *Distance of BLE*. Last accessed 14.09.2020. 2015. URL: https://devzone.nordicsemi.com/f/nordic-q-a/5470/distance-of-ble%5C#post-id-29191.

[39] *nRF51-Dongle*. URL: https://www.nordicsemi.com/Software-and-tools/Development-Kits/nRF51-Dongle.

[40] *Li-Ion Battery H2B181*. Last accessed 14.09.2020.

[41] *FreeRTOS*. Last accessed 07.11.2020. URL: https://www.freertos.org/RTOS.html.

[42] Brian S. Dean. *AVRdude*. Last accessed 28.08.2020. 2006. URL: https://www.cs.ou.edu/~fagg/classes/general/atmel/avrdude.pdf.

[43] Eirik Thon. "Mapping and Navigation for collaborating mobile Robots". MA thesis. Norwegian University of Science and Technology, Norway, 2016.

[44] C. Cadena et al. "Past, Present, and Future of Simultaneous Localization and Mapping: Toward the Robust-Perception Age". In: *IEEE Transactions on Robotics* 32.6 (2016), pp. 1309–1332. DOI: 10.1109/TRO.2016.2624754.

[45] Richard Hartley & Andrew Zisserman. *Multiple View Geometry in computer vision*. Second Edition. Cambridge university press.

[46] OptiTrack. *Motive Documentation*. Last accessed 14.09.2020. 2020. URL: https://v22.wiki.optitrack.com/index.php?title=Motive_Documentation.

[47] Microchip. *Atmel Studio Installation*. Last accessed 14.09.2020. URL: http://atmel-studio-doc.s3-website-us-east-1.amazonaws.com/webhelp/GUID-ECD8A826-B1DA-44FC-BE0B-5A53418A47BD-en-US-5/index.html?GUID-68EF3FD3-CE37-4DBE-BADF-A1BE5D642220.

[48] Microchip. *Atmel Studio 7 crashes - Unable to lanch Atmel Studio - Debugging steps*. Last accessed 14.09.2020. 2020. URL: https://microchipsupport.force.com/s/article/Atmel-Studio-7-crashes---Unable-to-launch-Atmel-Studio---Debugging-steps.

[49] *uxcell DC 12V 220RPM Encoder Gear Motor with Mounting Bracket 65mm Wheel*. Last accessed 27.10.2020. Amazon. URL: https://www.amazon.com/gp/product/B078HYX7YH/ref=ox_sc_act_title_1?smid=A1THAZDOWP300U&psc=1.

[50] R. Kalman. "A new approach to linear filtering and prediction problems". In: *ASME. J. Basic Eng.* (1960).

[51] Seth Hutchinson Mark W. Spong and M. Vidyasagar. In: *Robot Modeling And Control*. WILEY, 2006.

[52] Robins Mathew and Somashekhar S. Hiremath. "Trajectory Tracking and Control of Differential Drive Robot for Predefined Regular Geometrical Path". In: *Procedia Technology* 25 (2016). 1st Global Colloquium on Recent Advancements and Effectual Researches in Engineering, Science and Technology - RAEREST 2016 on April 22nd & 23rd April 2016, pp. 1273–1280. DOI: https://doi.org/10.1016/j.protcy.2016.08.221. URL: http://www.sciencedirect.com/science/article/pii/S2212017316305758.

[53] *Extended Kalman Filters*. URL: https://se.mathworks.com/help/fusion/ug/extended-kalman-filters.html.

[54] Robert Grover Brown and Patrick Y C Hwang. *Introduction to random signals and applied kalman filtering: with MATLAB exercises and solutions; 3rd ed.* New York, NY: Wiley, 1997.

[55] *Gyroscope, tutorials*. Webpage host: Sparkfun. Last accessed 09.11.2020. URL: https://learn.sparkfun.com/tutorials/gyroscope.

[56] P. Scott. *Uncertainty in measurement: Noise and how to deal with it," in Intermediate lab manual*. 2000.

[57] Dimitris G. Manoakis Johan G. Proakis. *Digital Signal Prosessing, 4th edition*. Pearson Prentice Hall, 2006.

[58] Tushar Malica, Singdha Shekhar, and Zakir Ali. "Design and comparison of butterworth and chebyshev type-1 low pass filter using Matlab". In: (Sept. 2011).

[59] *Introduction to Filter Designer*. Webpage host: Mathworks. Last accessed 06.01.2021. URL: https://www.mathworks.com/help/signal/ug/introduction-to-filter-designer.html.

[60] *PuTTY.org*. Last accessed 06.01.2021. URL: https://www.putty.org/.

[61] Michael Stanley. “Accelerometer Placement – Where and Why”. In: *NXP* (2012). Last accessed: 15.12.2020. URL: https://www.nxp.com/company/blog/accelerometer-placement-where-and-why:BL-ACCELEROMETER-PLACEMENT.

[62] *Magnetometer errors and calibration*. Last accessed 19.11.2020, education material. URL: https://www.vectornav.com/resources/magnetometer-errors-calibration.

[63] *Light Sensors*. URL: https://www.electronics-tutorials.ws/io/io_4.html.

[64] Lucas Kittmer. *Which Colors Reflect More Light?* URL: https://sciencing.com/colors-reflect-light-8398645.html.

[65] Stephanie Glen. *”Curve Fitting”*. URL: https://www.statisticshowto.com/curve-fitting/.

[66] Ahmad Kamal Nasir and Hubert Roth. “Pose Estimation By Multisensor Data Fusion Of Wheel Encoders, Gyroscope, Accelerometer And Electronic Compass”. In: *IFAC Proceedings Volumes* 45.4 (2012). Last accessed 5.01.2021, pp. 49–54. DOI: https://doi.org/10.3182/20120403-3-DE-3010.00068. URL: http://www.sciencedirect.com/science/article/pii/S1474667015404410.

[67] *Tenergy Universal Smart Charger*. Last accessed 04.01.2021. URL: https://nuxx.net/gallery/v/stuffivemade/bicycle_video_recorder/IMG_5101.jpg.html.

[68] *Download PuTTY*. Last accessed 04.01.2021. URL: https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html.

Appendix A

Manuals to operate the system

A.1 Set up Netbeans IDE

This section describes what is done to install and make Netbeans work on a windows computer.

1. Downloaded Java 8 for windows x64, from this webside https://www.oracle.com/java/technologies/javase/javase-jdk8-downloads.html
2. Downloaded NetBeans IDE 8.0.2, and allowed every packet extention to be downloaded as well. Direct download link: https://netbeans.org/downloads/8.0.2/start.html?platform=windows&lang=en&option=all
3. Start NetBeans and open the SSNAR project
4. First time open the SSNAR project to the robot project an error will shown. This is resolved by finding the correct files and connect this to the project. This is found in SSNAR-cart → libraries *NB: it is worth noticing that no file is the same*

A.2 Use of the Netbeans server

The server runs in Netbeans. To use the server in the correct way, the following steps has to be followed.

1. Insert the server dongle in the host computer that will run the server application
   (a) In windows - *device handle*, see what COM port the dongle is connected to
2. Build the server application, press the hammer icon or press F11, or *Run* → *Build file* if an error occurs, cheek that all libraries are resolved under the installation or build one more time

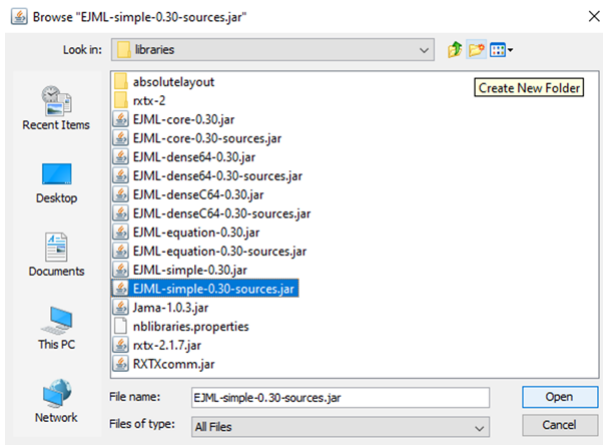


Figure A.1: Resolving error in Netbeans

3. Run the project, press the arrow button or *Run* → *Run Project*
4. Then a pop-up window shows, and you can choose which mode the server is going to run. Today you can choose Simulated World or Real World
5. In Real World: select setting and choose the COM port where the server dongle is
6. Turn on the robot and see the dongle on the robot and server connect by blinking and light red
7. The robots name will appear in the pop-up window
8. Click on connect
9. A new pop-up window will appear, insert the initial position
   (a) If you want the manual mode, check this in the check box
10. To start the navigation controller in the server: press *Start*

A.3 Flash script

This is to program the robot with a hex file, using AVRdude commands without Atmel Studio.

```
:: Requirements:
:: * First argument should be the relative path to the hex-file you want
     to flash. Example: 100920\lidarz to flash the file called lidarz.hex
     in the 100920 folder.
:: * The path to avrdude.exe must be added to your path
:: * The arduino installation must be on your
     C:\ drive, and under " C:\Program Files (x86)\ "

```

```
7     @echo off
8     avrdude -F -v -p atmega2560 -c wiring -P COM5 -b 115200 -D -U flash:w:
      "%CD%\%1.hex":i -C "C:\Program Files (x86)\Arduino\hardware\tools\avr\
      etc\avrdude.conf"
9
10 :: * %CD% returns current directory for the script
11 :: * %1% returns the first argument for the script
12 :: * Example:
13 :: * Running the command: C:\...\flash_lidar>flash 100920\lidar
14 :: * Converts "%CD%\%1.hex" into "C:\....\flash_lidar\100920\lidar.hex"
```

Listing A.1: SW script to flash the robot with AVR over terminal

A.4 How to charge the robot

The charger is a universal smart charger from Tenergy, and must be set to 11.1 V (nominal battery voltage), to charge the robot. The wiring for 0 V or ground (the black wire) is connected to the springs closest to the ground. The red wire is fastened to one of the springs positioned higher. The switch on the side of the robot must then be switched on. This closes the circuit so that the current can flow to the battery. While the robot is charging, the charger will signal this with a red light, which is explained on the back of the charger.

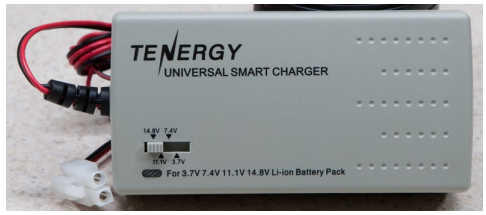


Figure A.2: The commercial charger, image from [67]

A.5 How to debug via Putty

In theory, the robot can be debugged through the server, but the unresolved problem of making the debug function restricts the logging. Therefore Torgeir Myrvang developed software to print values from the robot using Putty via USB.

Install Putty

To install Putty, the link in [68] was used. By selecting the software that fits the computer description, it is then only to follow the guidelines during installation.

Using Putty.exe

To use Putty with the robot, the following has to be set in the Putty configuration. Under *Session*, select *serial*, under connection type. The port number where the USB is connected to the robot is written under *serial line*. Set *speed* to 9600. Under *logging*, it is

possible to select the name and where the log file is to be saved. In *terminal*, check of the box *implicit CR in every LF*, this allows the data to be printed on the left of the terminal. To start putty, select *load*.

In the robot application

To print over Putty from the robot, some setting has to be in place. Include the files *"Serial_print.h"* and *"AVR_uart.h"* in the c-file that will print something. The function has to be initialized in main by writing:

```
USART0_init(USART_BAUD_PRESCALE(USART_BAUD_9600) );
Enable_print();
```

After this is done the *printf* function from c, can be used.

Limitation

A problem with the Putty-printing is the conflict with the USART, used by the Dongle communication. This limits the printing to be done without been connected to the server.

A.6 How to use the Optitrack system

Steps to use the Optitrack system and to save the data. The hardware and the camera and software is found at NTNU room B333 [10].

Steps to use the Optitrack system:

1. Calibration

   I After powering the SW, click on calibrate the system in the menu.

   II Click on *Block visible points*. This is to ensure that there are no false markers that can destroy the calibration.

   III Press Start *wanding*. Then take the wand (see figure A.3) and move it over the testing area, in a continuous movement. In the documentation is said to draw the figure eight or brush the dust in the air. It is important to cover the whole testing space and get sufficient samplings. The point of this is to let the cameras detect the wand.

   IV Continue to want until the quality is set to very high.

   V Press *Calculate*. Here the program trajectories the samples from each 2D image to a 3D image.

   VI Press *Apply result*, after the software is finishing with the calculation. A pop-up window with the calibration result is shown.
   During this project the following calibration result was achieved:
   Calibration result: Exceptional
   Calibration Quality: Very High

2. Set ground plane:

I To determine the ground and origin, the calibration figure *L*, must be placed in the testing area. In this project the L was placed in a marked corner, where the robot will start, like in figure A.4.

II In the software tool, mark the points from the calibration figure and click *Set ground plane*.

3. Tracking the robot

I Remove the calibration figure out of the testing area.

II Place the robot at the origin where the testing figure was.

III In the software application: mark the points, right click and select *make rigid body from selected marks*.

IV In *capture layout* (button in the top right window), it is easy to click record and stop record.

4. Exporting tracking data

I After the tracking, the file is shown with a time stamped on the left side of the screen.

II Right click on the file and select trajectories.

III Right click on the file again and select export tracking data.

IV Save the file as CSV file. The CSV file can be imported to Matlab using code written by Matlab.

Figure A.3: Wand used to calibrate the Optitrack camera, image from [46]

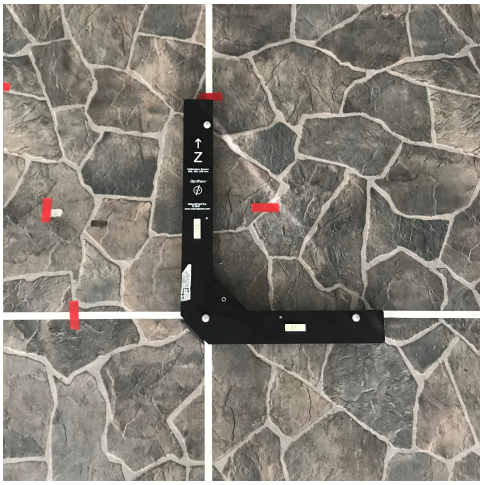


Figure A.4: Placement of the calibration L in the testing area

A.7 Lidar pinout

Since this master thesis did not use the lidar, the lidar was detached from the robot. For future work on the lidar, the method of attachment and connection of the wiring is documented in this section. The mounting of the lidar on the sensor tower is seen in figure A.5. The method for attaching the lidar is quite fragile, meaning tape has been used to additionally secure the sensor. For wiring the lidar to the self-made PCB the pin-out described in table A.1 must be used.

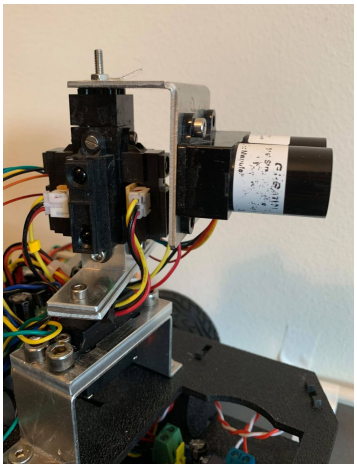


Figure A.5: Lidar mounted on the sensor tower

Wire	Pin
Black	1
Blue	2
Green	3
Red	4

Table A.1: Lidar pin-out

Appendix B

More results from testing

B.1 Square test

The square test has generated a large data set. Some of the results are viewed in this section. The reader is referred to the chapter 4.1.1 for the post-processed result.

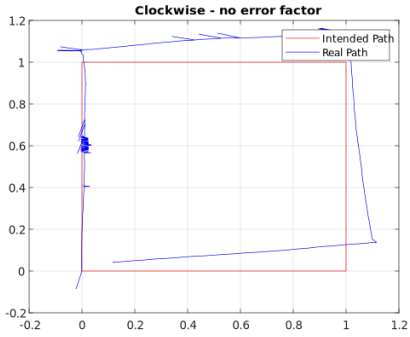


Figure B.1: Square test result with no error factor implemented

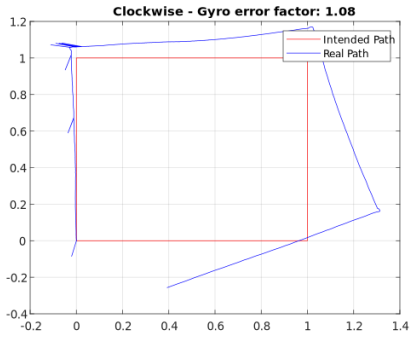


Figure B.2: Square test result with gyro error factor implemented

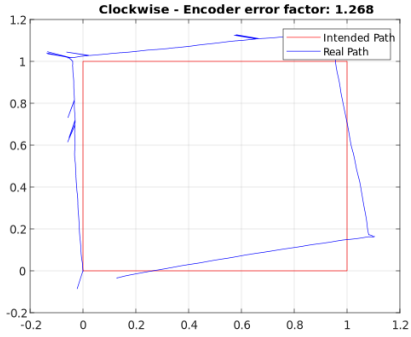
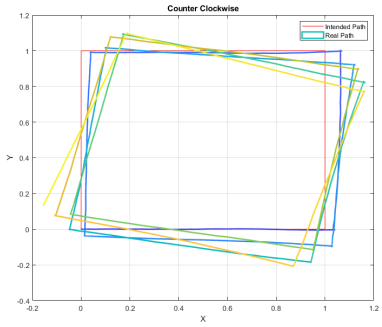


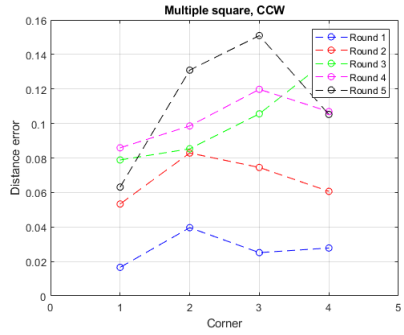
Figure B.3: Square test result with encoder error factor implemented

B.2 Continuous square test

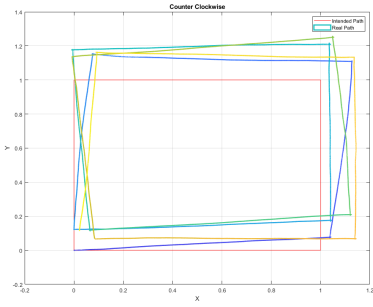
All plotted results from the continuous square test test is found in this section. Tables that view the error from the continuous test is found in table B.1 ans B.2.



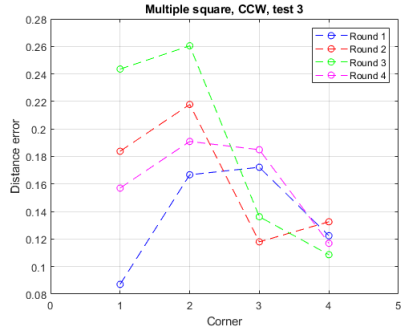
(a) Square test CCW test 1



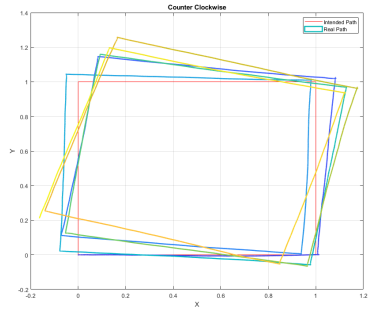
(b) Error distance from CCW test



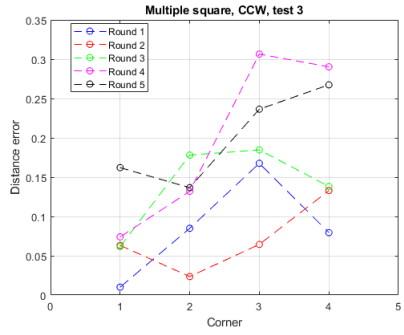
(c) Square test CCW test 2



(d) Error distance from CCW test

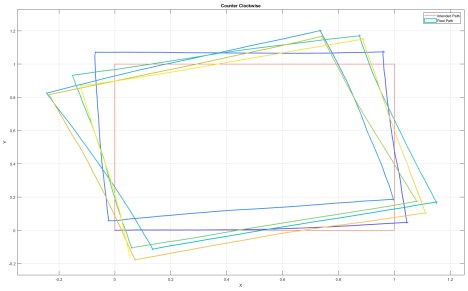


(e) Square test CCW test 3

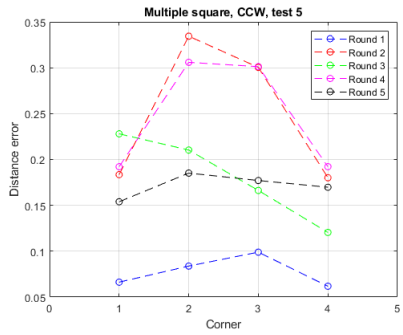


(f) Error distance from CCW test

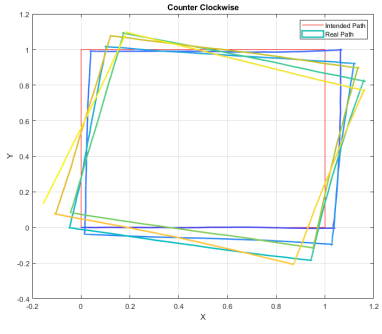
Figure B.4: Result of distance from continuous square test, CCW direction



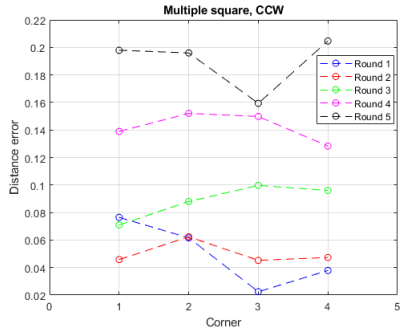
(a) Square test CCW test 4



(b) Error distance from CCW test

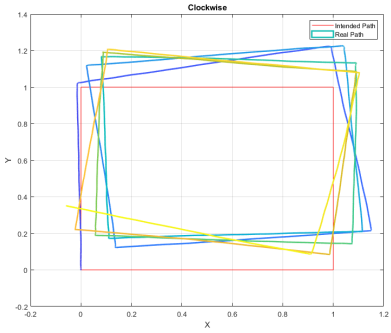


(a) Square test CCW test 5

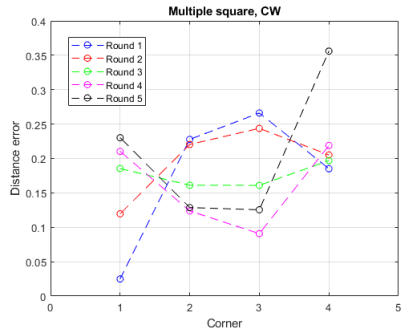


(b) Error distance from CCW test

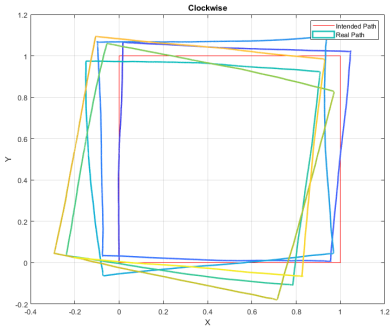
Figure B.6: Result of distance from continuous square test, CCW direction



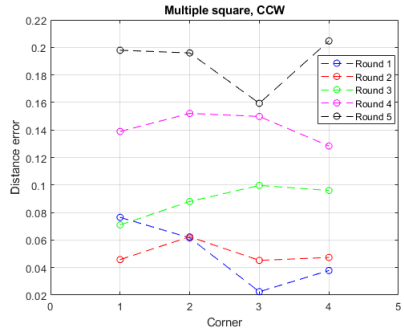
(a) Square test CW test 1



(b) Error distance from CW test

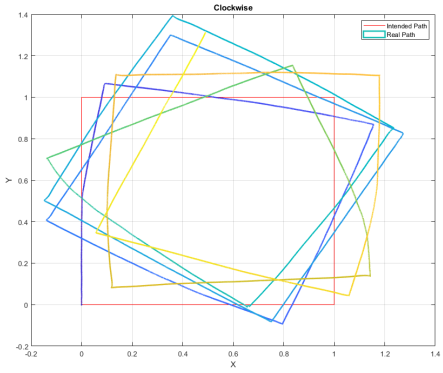


(c) Square test CW test 2

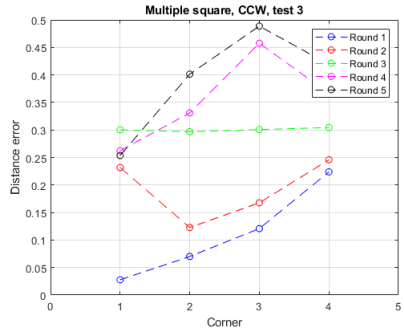


(d) Error distance from CW test

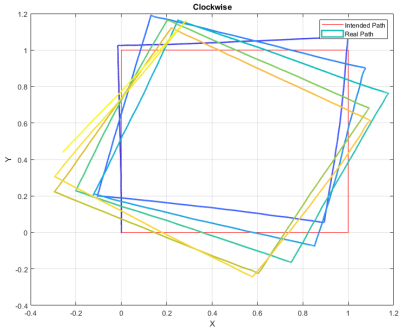
Figure B.7: Result of distance from continuous square test, CW direction



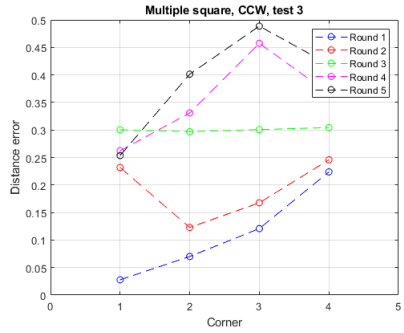
(a) Square test CW test 3



(b) Error distance from CW test



(c) Square test CW test 4



(d) Error distance from CW test

Figure B.8: Result of distance from continuous square test, CW direction

Net error

In table B.1 the result of the net error from each turn in the continuous square test are found. The result is rounded to the nearest thousandth. The visualization of the net error is found in figure 5.1 and 5.2.

			CCW	
Turn	1	2	3	4
Round 1	0.0537	0.1057	0.1340	0.0735
Round 2	0.1059	0.1442	0.1205	0.0373
Round 3	0.0307	0.0202	0.0180	0.0829
Round 4	0.0989	0.1557	0.1945	0.0840
Round 5	0.0453	0.0065	-0.0135	0.1027
			CW	
Turn	1	2	3	4
Round 1	0.0359	0.1030	0.1352	0.1566
Round 2	0.0902	0.0290	0.0334	0.0464
Round 3	0.1109	0.1930	0.2632	0.2400
Round 4	0.1198	0.0716	0.0505	0.0816
Round 5	0.1432	0.2623	0.3055	0.2576

Table B.1: Result of the net error calculation from the continuous square test

Average error

The average distance error is calculated and the result is viewed in table B.2. The result is viewed in chapter 5 in figure 4.3.

			CCW	
Turn	1	2	3	4
Round 1	0.1508	0.1391	0.1674	0.2033
Round 2	0.1590	0.1407	0.1484	0.1890
Round 3	0.1827	0.1427	0.1701	0.1918
Round 4	0.1533	0.1333	0.1556	0.1871
Round 5	0.1615	0.1389	0.1604	0.1928
			CW	
Turn	1	2	3	4
Round 1	0.1364	0.1872	0.1999	0.2276
Round 2	0.1260	0.1677	0.1664	0.2046
Round 3	0.1437	0.1860	0.1891	0.2181
Round 4	0.1354	0.1803	0.1852	0.2168
Round 5	0.1354	0.1803	0.1852	0.2168

Table B.2: Result of average distance error of the continuous square test

B.3 The larger track

Figure B.9 shows the larger track used to test the robot.

Figure B.9: The larger track that was tested

B.4 Robot heading

SW code to extract the robot heading from main. The data was extracted with Putty.

```
void main(){
    //snipped code for encoder ticks
    float gyroWeight = 0.8;
    uint16_t samples = 300;
    float gyro = 0;
    float dRobot = 0;
    float gyroOffset = 0.0;
    for (i = 0; i<=samples; i++){
      gyro+= fIMU_readFloatGyroZ();
    }
    gyroOffset = gyro / (float)i;

  while (1){
      _delay_ms(500);

    ATOMIC_BLOCK(ATOMIC_FORCEON){
      leftEncoderVal = gISR_leftWheelTicks;
      gISR_leftWheelTicks = 0;
      rightEncoderVal = gISR_rightWheelTicks;
      gISR_rightWheelTicks = 0;
    }

    vMotorEncoderLeftTickFromISR(gLeftWheelDirection,&leftWheelTicks,
    leftEncoderVal);
    vMotorEncoderRightTickFromISR(gRightWheelDirection,&rightWheelTicks,
    rightEncoderVal);
```

```
25
26      float dLeft = (float)(leftWheelTicks - previous_ticksLeft) *
        WHEEL_FACTOR_MM;
27      float dRight =(float)(rightWheelTicks - previous_ticksRight) *
        WHEEL_FACTOR_MM; /
28      previous_ticksLeft = leftWheelTicks;
29      previous_ticksRight = rightWheelTicks;
30      dTheta = (dRight - dLeft) / WHEELBASE_MM;
31
32      // for the gyroscop
33      gyrZ = fIMU_readFloatGyroZ()-gyroOffset;
34      gyrZ *= period_in_S * DEG2RAD;
35      dTheta =  (1 - gyroWeight) * dTheta + gyroWeight * gyrZ;
36      predicted_Robotheading += dTheta;
37      printf("Robot Heading: %f \n", predicted_Robotheading);
38    }
39  }
```

Listing B.1: SW to extract the encoder ticks from let wheel

B.5 Gyroscope data

Image B.10 shows the result of the noise test of the gyroscope.

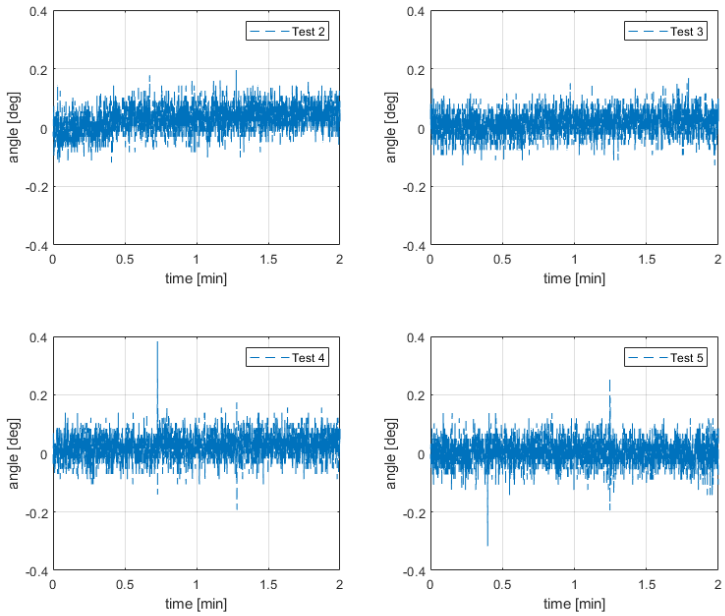


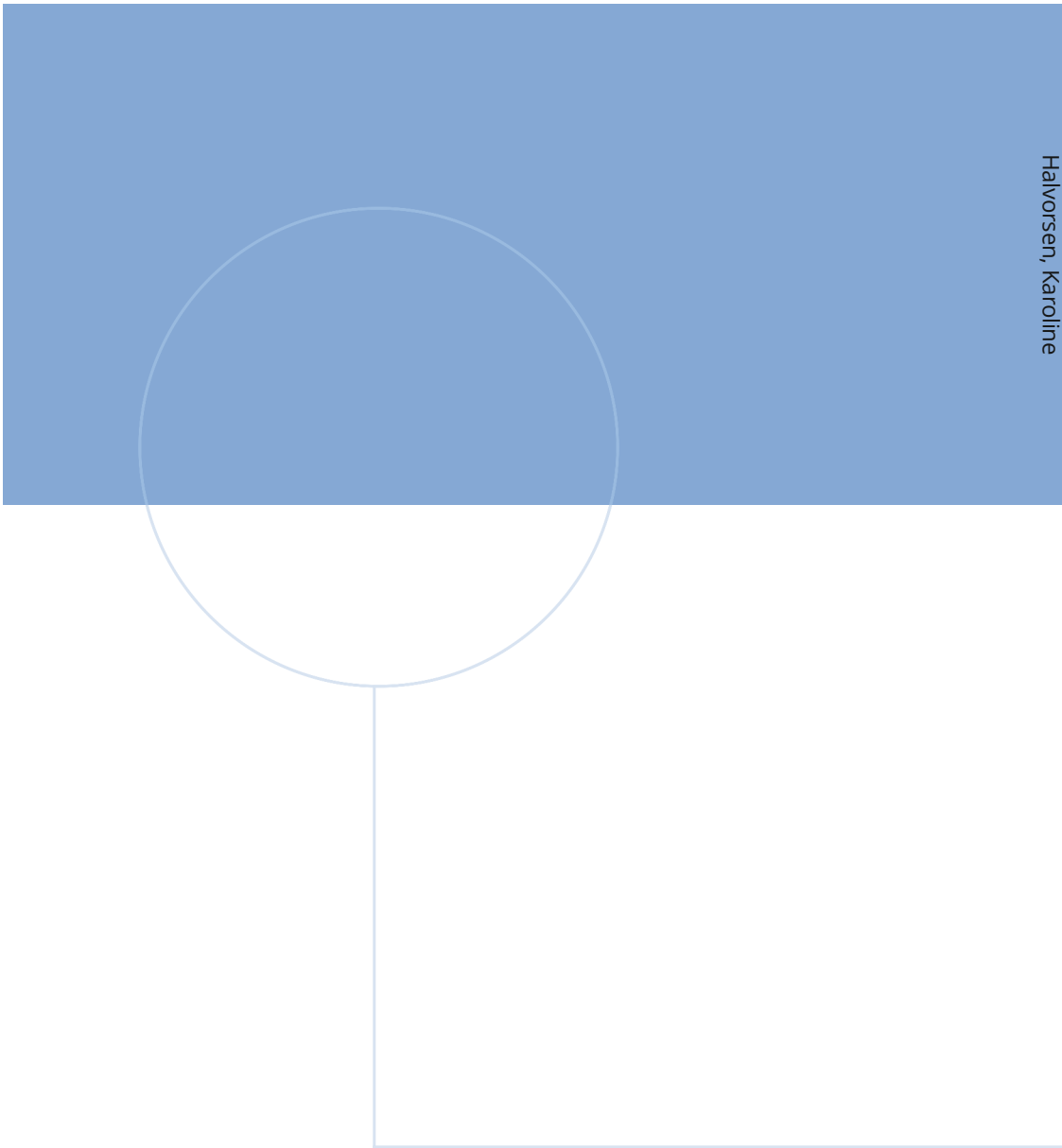
Figure B.10: Noise in the gyroscope, result from test explained in chapter 3.3.1

B.6 Data from IR calibration

The data is found by executing the method described in chapter 3.3.5. The result is shown in table B.3.

cm	10	10	10	11	11	11	12	12	12	13	13	13	*unit*
FrW	2.234	2.244	2.239	2.068	2.079	2.063	1.929	1.924	1.926	1.81	1.80	1.85	[V]
Left	2.306	2.312	2.295	2.125	2.141	2.115	1.960	1.965	1.955	1.836	1.852	1.836	[V]
Right	2.275	2.264	2.270	2.135	2.141	2.140	1.996	2.007	1.991	1.862	1.857	1.867	[V]
Rear	2.450	2.507	2.440	2.105	2.130	2.099	2.022	2.048	2.012	1.878	1.878	1.867	[V]
cm	14	14	14	15	15	15	20	20	20	25	25	25	
FrW	1.707	1.712	1.707	1.594	1.599	1.604	1.233	1.238	1.227	1.021	1.016	1.31	[V]
Left	1.687	1.681	1.707	1.594	1.599	1.609	1.269	1.274	1.289	1.037	1.042	1.057	[V]
Right	1.774	1.785	1.769	1.666	1.661	1.671	1.315	1.300	1.305	1.037	1.042	1.057	[V]
Rear	1.764	1.790	1.759	1.651	1.676	1.645	1.289	1.320	1.289	1.057	1.062	1.088	[V]
cm	30	30	30	40	40	40	50	50	50	60	60	60	
Frw	0.866	0.882	0.892	0.670	0.675	0.706	0.572	0.562	0.557	0.454	0.450	0.444	[V]
Left	0.887	0.892	0.871	0.675	0.696	0.681	0.557	0.562	0.577	0.495	0.500	0.515	[V]
Right	0.902	0.897	0.913	0.711	0.706	0.722	0.593	0.588	0.593	0.495	0.500	0.505	[V]
Rear	0.908	0.928	0.933	0.711	0.706	0.737	0.588	0.624	0.593	0.495	0.521	0.490	[V]
cm	70	70	70	80	80	80							
Frw	0.397	0.400	0.395	0.355	0.354	0.356							[V]
Left	0.438	0.454	0.438	0.381	0.376	0.397							[V]
Right	0.438	0.433	0.443	0.397	0.407	0.400							[V]
Rear	0.433	0.428	0.464	0.397	0.417	0.392							[V]

Table B.3: Voltage measurement from the IR sensor calibration



Halvorsen, Karoline

NTNU
Norwegian University of
Science and Technology