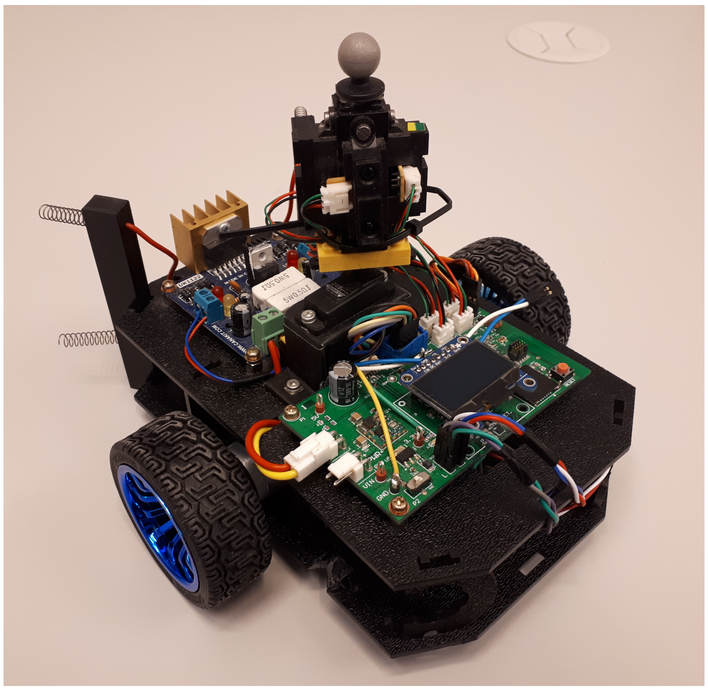
Master's thesis

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

Arild Stenset

nRF52 robot with OpenThread

Master's thesis in Cybernetics and Robotics
Supervisor: Tor Onshus
June 2020



NTNU
Norwegian University of
Science and Technology

Arild Stenset

nRF52 robot with OpenThread

Master's thesis in Cybernetics and Robotics
Supervisor: Tor Onshus
June 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

Problem description

The goal of this thesis is to improve the performance of an existing robot which is part of a project where the long term goal is to have several robots work together to map unknown areas. Earlier reports based on the same theme has reported software and hardware issues with the robot which affects its performance. The hardware issues are mainly related to mechanical problems with the motors driving the wheels. The software issues concerns suggested improvements. The robot communicates with a server application which receives information about the robots surroundings and constructs a map from these. The server application also has the ability to control the robot remotely. Two server applications was available for use at the beginning of this thesis, while a third is being developed parallel to this thesis. For the third server application a new communication protocol is developed. Support for the new communication protocol has to be implemented for the robot's software as well. The following points will form the basis for this thesis:

- Evaluate and analyze current performance
- Replace existing powertrain
- Integrate and test new powertrain
- Implement anti-collision
- Adapt robot software to the new server application
- Document performance after changes

Summary and conclusion

This thesis describes the changes done to the robot in detail and covers how the implementation was done and which design choices were made.

At the beginning of the thesis an issue with the right motor and the existing powertrain was known from earlier reports. Before the evaluation of the current navigation performance could be tested, the right motor was replaced. A new powertrain was ordered and while waiting for this, the navigation performance revealed issues which has not been mentioned in earlier reports. The issue is a slow drift in the robot's heading, this not only causes problems with navigation, but is also affecting the distance measurements sent to the server application. A thorough investigation to analyse the drift problem has been done, and a solution to the drift issue has been implemented in the software. The applied solution reduces the sensitivity of the gyro, but this has showed no signs of being a problem through all the performed tests. During the testing of the gyro, an issue with the update rate of the estimated heading from the kalman filter was discovered. The sum of the angular rates from the gyro showed to be a closer estimate for the robot's heading.

The new powertrain was installed without major problems, however the full potential of the new encoders can not be utilized because of lacking input pins. Testing of the new powertrain revealed a deviation between the actual gear ratio and the ratio mentioned in the specifications. The previous motor controller showed to be difficult to tune to fit the new powertrain. A new controller design was made, where the heading and distance controllers now are separated into individual functions which makes it possible to tune them separately. The new powertrain combined with the current tune of the controllers offers good navigation results, but there is still room for a better tune of the controllers to give the robot better performance for different floor conditions.

The current implementation of the anti-collision is only ran when the robot is driving forward. Because it only uses the forward distance sensor the robot can still hit objects outside the detection sector of the sensor. The anti-collision stops the robot, but has room for improvements, especially for detecting a possible collision outside the narrow sensor sector. Since the detection sector for a sensor is very narrow, the collision object may be larger. For the robot to discover the whole object, collision sectors was introduced and implemented. The collision sectors are used by the robot to validate waypoints before processing them. The collision sectors works as intended, but the slow turning speed of the sensors caused issues with clearing the sectors taking some time.

The new communication protocol for use with the new server application showed to have unexpected limitations. Testing revealed that the limitations do not affect small changes in message size for messages received at the robot. To expand the protocol and the message size further, extended testing has do be done to ensure received messages arrives correctly.

Oppsummering og konklusjon

Denne oppgaven beskriver endringer gjort på roboten i detalj, hvordan de ble implementert og hvilke vurderinger som ble gjort underveis.

Fra tidligere rapporter var problemer med høyre motor kjent i starten av oppgaven. Høyre motor ble byttet før den første navigasjonstesten ble gjennomført. Den første navigasjonstesten ble gjennomført mens man ventet på de nye motorene. Denne testen avslørte problemer som ikke er nevnt i de tidligere rapportene. Roboten viste tegn til at vinkelen sakte drifter. Siden avstandsmålingene regnes ut fra robotens vinkel, vil de også drifte. Dette medførte at målingene som blir sendt til serveren også flytter seg med vinkelen. En grundig undersøkelse av hva som forårsaker dette ble gjennomført, og en løsning som løser det ble implementert i koden. Løsningen medfører at sensitiviteten til gyroskopet er noe redusert, men har så langt ikke vist problemer under resten av testene. En annen oppdagelse som ble gjort samtidig er at vinkelestimatet fra kalman filteret ikke oppdateres like raskt som summen av vinkelhastighetene fra gyroen. Over noe tid førte dette til et betydelig avvik i vinkelen mellom de to. Vinkelen gitt av summen av vinkelhastigheter viste seg å være et bedre estimat for robotens vinkel.

Installasjonen av de nye motorene, inkludert girboks, hjul og encodere ble installert uten store problemer. På grunn av manglende innganger på roboten kan ikke de nye encoderne benyttes fult ut. Et avvik mellom faktisk og oppgitt utveksling for girboksen ble avdekket gjennom testing. Kontrolleren for de forrige motorene viste seg å være vanskelig å justere for å passe sammen med de nye motorene. På bakgrunn av dette ble nye kontrollere laget og implementert. De nye kontrollerne består av en kontroller for vinkel og en kontroller for avstand. De to er implementert i to separate funksjoner som gjør det mulig å justere de individuelt. De nye kontrollerne sammen med de nye motorene gir gode resultater under navigasjonstester, men bør ha en ny justering for å fungere optimalt på flere typer underlag.

Anti-kollisjons funksjonaliteten som ble implementert kjøres bare når roboten kjører fremover. Den benytter avstandsmålinger fra sensoren som peker forover for å detektere en mulig kollisjon. Sensoren har en ganske smal dekningsgrad som gjør at roboten fortsatt kan kollidere om et objekt er utenfor dekningssektoren til sensoren. Anti-kollisjonen har vist seg å stoppe roboten for hindringer rett foran den, men bør oppgraderes til å dekke et større område av fronten av roboten. Selv om roboten detekterer et objekt og stopper, kan objektet dekke en betydelig del av robotens front. For at roboten skal kunne håndtere dette ble kollisjons-sektorer implementert. Når roboten står i ro lager den sektorer der den har detektert kollisjoner. Disse sektorene brukes videre for å validere nye punkter den skal kjøre til. Om vinkelen til punktet er innenfor en sektor blir det forkastet før det blir håndtert videre av roboten. Kollisjonssektorene har vist seg å fungere, men på grunn av at sensorene roterer så sakte, tar det lang tid før en eventuell sektor blir klarert igjen.

Den nye kommunikasjonsprotokollen for den nye serverapplikasjonen viste seg å ha uventede begrensninger. Størrelsen på meldinger som roboten mottar er begrenset fra en tidligere implementasjon. På tross av dette viste tester at en litt større melding fortsatt ble mottatt riktig av roboten. En videre utvidelse av protokollen må inkludere grundige undersøkelser om mottatte meldinger blir riktig mottatt av roboten om den nåværende maskinvaren fortsatt skal benyttes.

Preface

This master's thesis forms the foundation for evaluation in the 30 credit course *TTK4900 - Engineering Cybernetics, Master's Thesis*. The thesis concludes a 5-year master's degree program at the Department of Engineering Cybernetics at the Norwegian University of Science and Technology (NTNU). Work was done through the spring of 2020, from January to June.

The work conducted in this thesis is based on little to no previous knowledge about the Lego robot project. This led me to spending a significant amount of time understanding both the software, the hardware and the history of the project. The learning curve has been steep and wide, but rewarding. Developing embedded systems has always been a large interest and this project contributed to valuable knowledge and insight into Real-Time Operating Systems and modern communication protocols for wireless systems.

Available resources at the beginning of the thesis consists of a desktop computer, the nRF52 robot, a nRF52 Development Kit, a MQTT gateway and access to previously written reports and software. However the COVID-19 situation led to a lockdown of the university in February which resulted in most of the development being done at home using my personal laptop.

First and foremost I would like to thank my family for the support, help and motivation given throughout the time of studying. A special thank to Ragnhild Janis Stenset for proofreading the thesis.

Secondly I would like to thank my friends, my fellow students and my girlfriend for contributing to funny and memorable moments during my time studying at NTNU in Trondheim.

I also want to thank my supervisor Tor Onshus for facilitating, guidance and fast response throughout the project.

Also thank to the people at the ITK workshop and the Omega Workshop for access to equipment, parts and tools.

Lastly I want to thank the fellow students at the office for rewarding technical discussions during the project period.

[illegible]

Arild Stenset - Trondheim, June 2020

Table of Contents

List of Tables

List of Figures

Abbreviations

Symbol		Definition
IR	=	Infrared
IOT	=	Internet Of Things
LIDAR	=	Light Detection And Ranging
FreeRTOS	=	Free Real-Time Operating System
PCB	=	Printed Circuit Board
IMU	=	Inertial Measurement Unit
PWM	=	Pulse-Width Modulation
SoC	=	System on Chip
I2C	=	Inter-Integrated Circuit
ADC	=	Analog to Digital Converter
SES	=	Segger Embedded Studio
MQTT	=	Message Queuing Telemetry Transport
DK	=	Development Kit
SLAM	=	Simultaneous localization and mapping
MSB	=	Most Significant Bit
LSB	=	Least Significant Bit

Chapter 1

Introduction

1.1 Background and motivation

Robotics has been increasing its importance in our lives throughout the years, both for simple and advanced tasks. An increasing level of autonomy and the Internet Of Things (IOT) has mainly been the field of latest improvements. Developing advanced connected systems has been made easier due to cheaper and more powerful hardware and increased access to optimized software.

The development of the Lego robot project has the purpose of combining modern wireless communication with Simultaneous Localisation And Mapping (SLAM) using low price and easily accessible components and sensors. The combination of components and sensors with suitable software forms the future goal of the project, which is to have several robots that works together and talks with each other to map an unknown area autonomously to a server application.

1.2 Previous work

During the Lego robot project, several different robots have been developed. Some based entirely on Lego components and some with other components from different manufacturers. They all share the same differential driven layout with one motor at each side and a caster ball at the rear. For distance measurements, infrared (IR) sensors are mainly used, while some have Light Distance And Ranging (LIDAR) sensors.

Several people has contributed to the current stage of the particular robot used throughout this thesis. Ese[1] converted the software to use Free Real-Time Operating System (FreeRTOS) in 2016. The custom pcb currently in use was created by Korsnes[2] in 2018, the pcb is known as the SLAM Control System. Installation and the final hardware driver development of the SLAM Control System was conducted by Leithe[3] in 2019. The robot was previously known as the Arduino robot, but is now referred to as the nRF52 robot because of the nRF52832 System on Chip (SoC) running on the SLAM Control System.

Concerning the communication and the server side of the project, the C++ server application developed by Grindvik[4] in 2019 has been the main server application used during this thesis. The setup of the new communication using Thread in order to use the C++ server application was done by Blom[5] in 2020.

This thesis is mainly a continuation of the work done by Leithe[3] and based on his future work notations. The hardware replacements is also based on experiences made by Blom[5].

1.3 Thesis structure

Chapter 1 - Introduction:

Chapter 2 - Theory: Gives a brief introduction about the robot, its communication capabilities and the different coordinate frames.

Chapter 3 - Development environment: Describes both the hardware and the software used throughout this thesis.

Chapter 4 - Initial performance: Describes how the initial tests was performed and presents the results from them.

Chapter 5 - Gyro calibration: Goes into what causes the problems with the gyro measurement, how it was solved and presents the results from the solution.

Chapter 6 - Hardware: Covers the installation of the new motors and the results using the previous controller. Also describes the calibration of the ir-sensors.

Chapter 7 - New controllers: Gives a good understanding of how and why the controllers were implemented in the way they were. Results from tuning are also covered.

Chapter 8 - Software: Covers the details around the anti-collision, collision sectors and mention's applied software changes.

Chapter 9 - Server Communication: Walkthrough of the new communication protocol and the limitations.

Chapter 10 - Test Setup: Gives an overview of the test setup used for the final tests.

Chapter 11 - Results: Presents the results from the test setup and describes them.

Chapter 12 - Discussion and future work: Describes the results in a wider context and with more detail. Also presents the main points of future work.

Chapter 2

Theory

2.1 nRF52 robot

The main components of the nRF52 robot can be summed in the following list:

- Chassis
- Sensortower, consisting of a servo and 4 ir distance sensors
- Motor controller pcb
- SLAM Control System pcb
- MPU6050 Inertial Measurement Unit (IMU)
- Motors, including gearboxes, encoders and wheels
- Nordic dongle for wireless communication
- Battery

Further details is found in [3] and [2].

Figure 2.1 shows how the robot looked when work first began on the thesis.

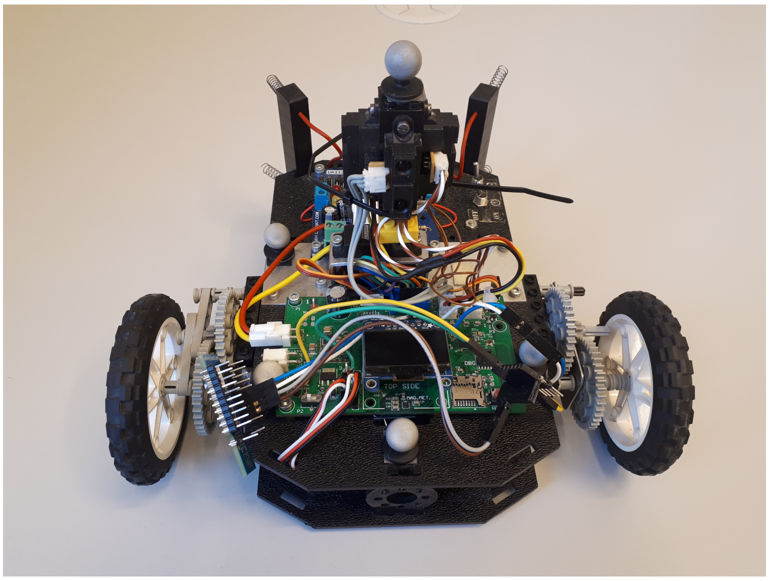


Figure 2.1: Appearance of the nRF52 robot at the beginning of this thesis.

2.2 Coordinate frames

In order to give a better understanding on how the robot's navigation works and how different frames relates to each other, some illustrative figures have been made. Note the wheel placement in fig. 2.2 and fig. 2.3 is related to the new wheel placement covered in chapter 6 and not fig. 2.1 because the turning axis has been relocated.

There are three main coordinate frames; the initial-frame, the robot-frame and the sensor-frame. Both cartesian and polar coordinates are used. The relation between initial-frame and robot-frame are shown in figure 2.2. The coordinate axis for the initial-frame is represented with a subscript i and the robot-frame use subscript r. When the robot is turned on, these two coordinate frames are aligned on top of each other with the robot's heading set to 0 and the x axis pointing forward. The centre of the robot chassis is aligned with the origin of the robot-frame. When the robot drives, the origin of the robot-frame, is calculated from the origin of the initial-frame which is stationary from the spot where the robot was turned on. The robot's internal heading θ_r is measured in radians from the initial-frame's x-axis and ranges from $-\pi$ to π as shown. Positions in cartesian coordinates will in this thesis be marked with square brackets such as: [X,Y].

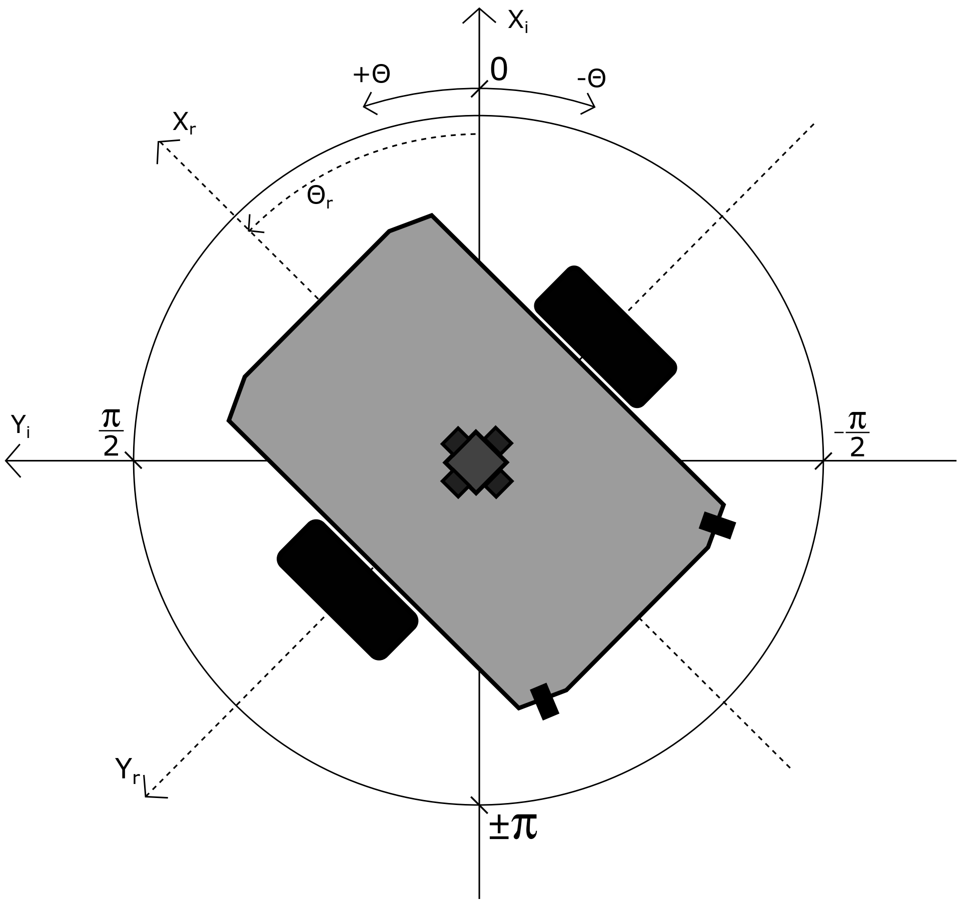


Figure 2.2: Robot frame in reference to initial frame.

The sensor-frame shown in figure 2.3 shows how the four ir-sensors rotates related to the robot-frame. The origin of the sensor-frame and the robot-frame are always aligned. One important difference from the other frames is that the sensor-frame internally uses degrees instead of radians. For a $360°$ scan, each sensor scans $90°$ back and forth. θ_{IR1} is the angle to ir-sensor number 1 (IR1) and is measured from the robot-frame's x-axis. The other sensors angles are related to this by $90°$ increments.

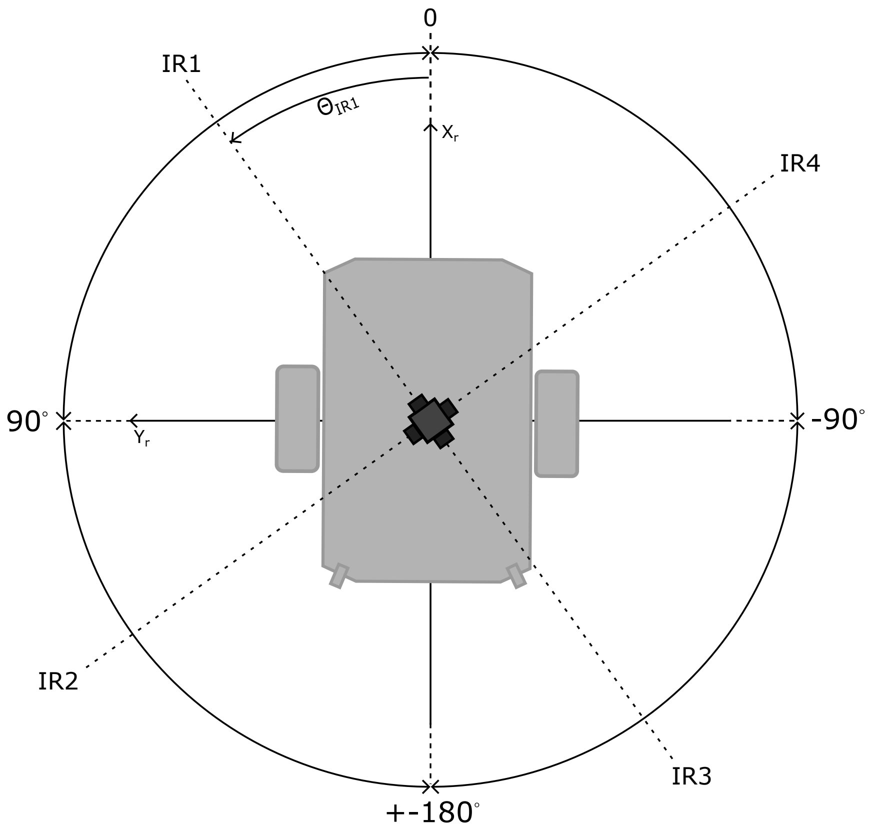


Figure 2.3: Sensor-frame in reference to robot-frame.

2.3 External communication

Data about the surroundings collected by the robot are sent to a server which constructs a map from them. The server application also has a control panel for remote control of the robot. From earlier years a server application made in Java has been used. The Java server application receives data from the robots through bluetooth and primarily uses the older nRF51 dongles for communication. Communication with the Java server is covered in detail in [1]. The C++ server application is designed to use the newer nRF52840 dongle for communication using OpenThread. OpenThread is the open-source implementation of the wireless mesh networking protocol Thread. Parallel to this thesis the second version of the C++ server application was developed by Mullins[6]. The two C++ server applications will during this thesis be referred to as v1C++ and v2C++. This robot compared to the others has the capability of communicating with all three servers by changing some configuration parameters in the software, this is covered in section 2.5. The communication protocol and support for v2C++ was made during this thesis and details is found in chapter 9.

The use of v1C++ and v2C++ requires a Message Queuing Telemetry Transport(MQTT) gateway, details on this setup is found in [5].

2.4 FreeRTOS

In order to divide processor time for different processes the robot runs FreeRTOS[7]. FreeRTOS is suitable for microcontrollers and is distributed under the MIT license. FreeRTOS uses tasks and a scheduler to share computer power to different processes. Table 2.1 covers the most important information about the tasks running on the robot. Further details about the tasks on this robot is found in [3]. Note the periods may increase during high processor loads.

Table 2.1: FreeRTOS tasks

Task:	**Period[ms]:**	**Priority:**
MainPoseEstimatorTask: - IMU calibration and pose estimation.	40	3
MainPoseControllerTask: - Anti-collision and controllers. - Signaled by EstimatorTask.	40	1
MainSensorTowerTask: - Rotates sensortower and sends messages.	200	1
MainCommunicationTask: - Takes care of incomming messages.	500	1
microsd_task: - Writes data to microSD card. - Blocked while its queue is empty.	-	1
user_task: - Initialization of drivers. - Priority decreased to 1 after driver initialization.	1000	4/1
display_task: - Writes data to onboard oled display. - Blocked while its queue is empty.	-	1

2.5 Configuration parameters

In order to set up the software to work for a particular server application, configuration parameters has been introduced. USEBLUETOOTH was introduced in [3], while the other two was made during this thesis. The last parameter was introduced to be able to turn off an additional check of waypoints, details are covered in section 8.2.

- **USEBLUETOOTH**: A boolean variable used to switch between using bluetooth or Thread. If set to true, the robot is set up to use the communication protocol for the Java server. This also requires the use of a nRF51 dongle. When false the robot uses Tread with a C++ server and a nRF52840 dongle is needed.

- **newServer**: This boolean variable only has an affect when USE_BLUETOOTH is false. It is used to switch between different versions of the C++ server because different communication protocols is used. When false, the protocol suits the C++ server developed in [4]. The protocol covered in chapter 9 is used when set to true.

- **validateWP**: Also a boolean, but this is used to turn the validation of waypoints on(true) or off(false) according to detected collision sectors. Details are covered in section 8.2.

2.6 Camera Tracking

The OptiTrack motion capture system consists of several cameras which track reflective spheres attached to the robot. This system is used to track the actual movement of the robot during 1 meter square tests. The upper half of figure 2.4 shows how the robot looks in OptiTrack motion capture software after a rigid body has been made from the visible spheres. The rigid body can be formed with as many reflective spheres as needed. The lower half of the figure shows how the reflective spheres are placed on the robot during tracking. The tracking system tracks the larger sphere located in the centre and it is favorable that this sphere also matches the centre of the robot. Since the tracking system uses ir to track the robot, bare metal surfaces will cause unwanted noise in the tracking data. This is easily solved by covering the metal surfaces with tape. The lower picture also shows how the robot looked at the end of this thesis.

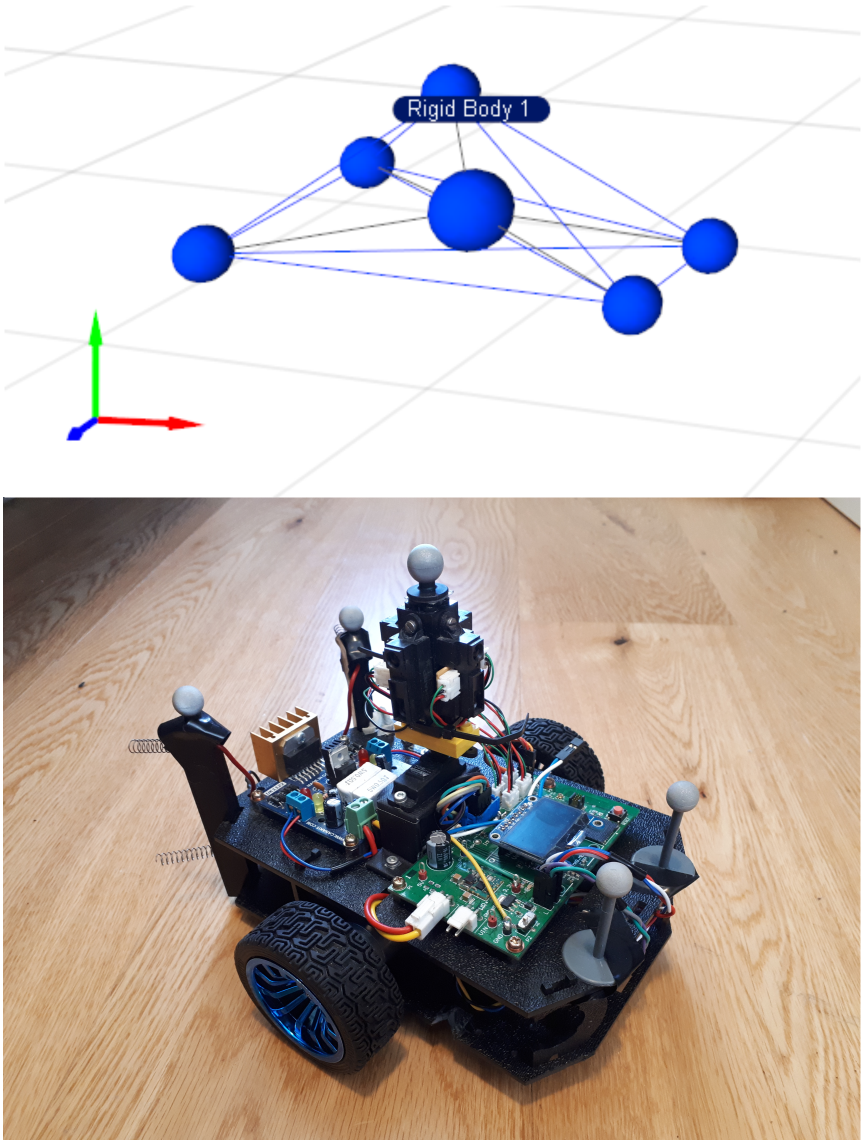


Figure 2.4: Comparison of how the robot looks in the tracking program and in reality.

Chapter 3

Development environment

Section 3.1.1 and section 3.1.2 are also covered in [2] and [3], but added to make the report meaningful on its own. Data-logging and debugging is covered in detail in [3].

3.1 Software

3.1.1 nRF5 Software Development Kit (SDK)

The SDK is filled with examples and modules for development based on the nRF51 and nRF52 boards, the nRF5 SDK gives access to different libraries for all kinds of projects. The nRF5 SDK comes in a wide variety of versions, but throughout this project the version 15.0.0 has been used and it is downloadable from [8]. The length of the file path can cause issues, of this reason it is to be recommended that nRF5 SDK is located in C:\. The project software folder, named slam_application, is then placed in the SDK folder C:\nRF5_SDK_15.0.0_a53641a\examples\ble_peripheral.

Both [2] and [3] has mentioned a bug in the nrfx_ppi.c file for the SDK, but no solution. The bug that was experienced at the beginning of this thesis was caused by a typo in nrfx_ppi.c. The solution is replacing line 47 in nrfx_ppi.c with:

```
#define NRFX_LOG_MODULE PPI
```

Listing 3.1: Bug fix in nrfx_ppi.c

This and other known issues for the nRF5 SDK is found at [9].

3.1.2 Segger Embedded Studio (SES)

For software development SES v4.50 32-bit was used. It is downloadable from Seggers webpage [10] and a free license is available to use with Nordic Semiconductors nRF SoC's.

An .emProject file is located in slam_application\pca10040\s132\ses which can be opened in SES when the SDK process above is done. The .emProject file opens all the files belonging to the project.

3.1.3 MQTT.fx

Release v1.7.0 was used for testing during this thesis. MQTT.fx is a program that is used to debug and test the Message Queuing Telemetry Transport (MQTT) protocol. Prior to use the broker address has to be applied. When connected to the broker one can subscribe or publish to different topics. By subscribing to the advertising topic of the robot, the payload data sent from the robot can be evaluated. For the C++ servers the advertising topic has the following form: v1/robot/”Name of robot”/adv. Further details is found in [5].

3.1.4 nRF Connect

Simplified program to flash the dongles when a .hex file is created from Segger. However, the currently attached nRF52840 dongle for use with Thread and the C++ servers cannot be flashed directly at this point. The dongle has two solder-bridges that have been modified in order to use an external regulated power supply. One possible solution is to solder a 10-pin JTAG header to the dongle, another is to reverse the solder-bridges. Further details about this process is found in [5].

3.1.5 J-Link RTT Viewer

Version 6.54c was used during this project to print and log data from the nRF52832 SoC on the robot with the NRF_LOG_INFO() function.

3.1.6 Server application

Details covered in section 2.3. During this thesis, testing has only been done with the C++ server applications, where v1C++ has been the main server application used.

3.2 Hardware

3.2.1 Battery charger

3.2.2 MicroSd card

Used for logging data from the robot during driving.

3.2.3 nRF52 Development kit (DK)

A nRF52 DK is needed to be able to program and debug the nRF52832 SoC located on the SLAM Control System pcb at the robot. Note if the DK is connected to the robot and the robot is not turned on, any programming will be done at the onboard nRF52832 SoC at the DK and not the robot. More information about suitable DK's for different SoC is found at [11].

3.2.4 MQTT gateway

The MQTT gateway consists of a Raspberry Pi and a nRF52840 dongle. The dongle on the robot connects to the dongle on the Raspberry Pi and the Raspberry Pi connects to an online MQTT broker. Details about this setup is found in [5].

Chapter 4

Initial performance

The purpose of this chapter is to evaluate the robots performance at the beginning of the thesis, in order to have something to compare with.

4.1 1 meter square tests

The 1 meter square tests are carried out at the test-lab in room B333 at NTNU. Waypoints to complete a 1 meter square are sent to the robot using the server application. These tests are important both to see the robots navigation performance, as well as to evaluate the communication with the robot. Square tests are usually carried out in pairs, one clockwise and one counter-clockwise run. Before such a test, the robots heading is aligned with the x axis as good as possible. For a counter-clockwise run, a straight forward movement is carried out first, while a 90 degree turn is carried out first for the clockwise run.

Prior to this test the right motor was replaced. How the robot looked during this test is shown in figure 2.1.

4.2 Initial results

For the counter-clockwise run seen in figure 4.1 the 1 meter square is marked with the red line. The real path, the blue line indicates the tracked path by the camera system. All the distances from the blue line are close to one meter. The first turn is very close to 90 degrees, but the second and third is getting further and further away from the 90 degrees target. During the turning from the first waypoint to the second, a curve is present. This curve shows that the turning axis of the robot is behind the tracking-sphere shown in fig. 2.4. The noise present in the figures is caused by uncovered ir-reflections from metal parts on the robot.

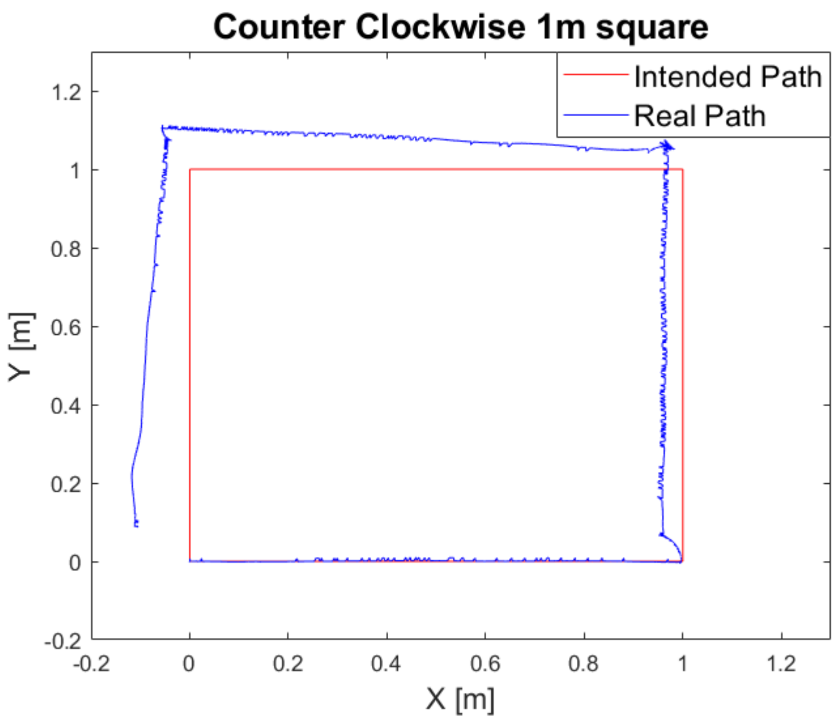


Figure 4.1: Initial 1m square test counter-clockwise.

Figure 4.2 shows the clockwise run where the robot starts with a 90 degree left turn. The same curves are seen here as mentioned above. An interesting thing is the second turn, where the opposite happens, the robots turn axis is ahead of the tracking-sphere. The two last turns are also outside 90 degrees, but in the opposite direction compared to the counter-clockwise turn.

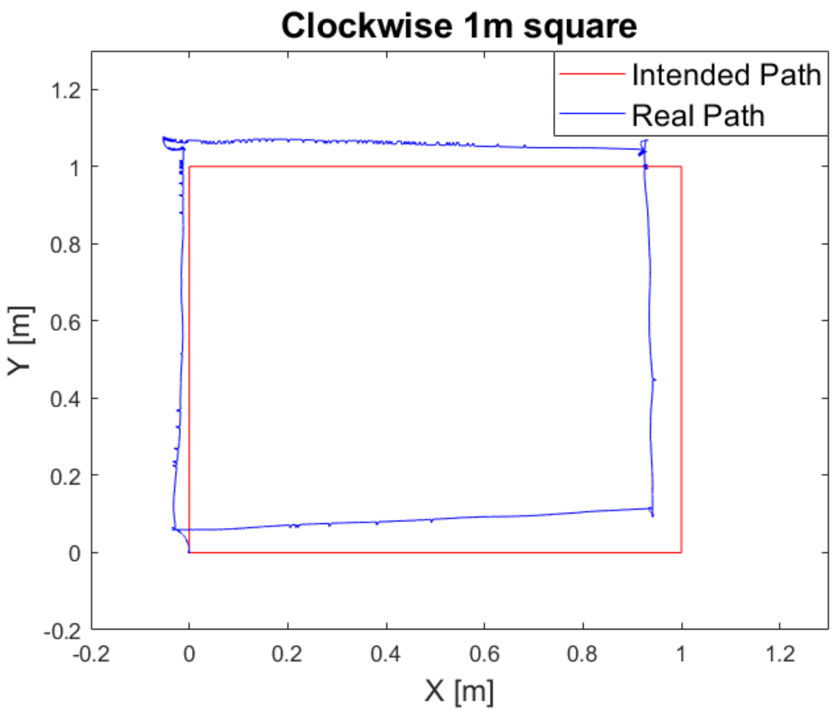
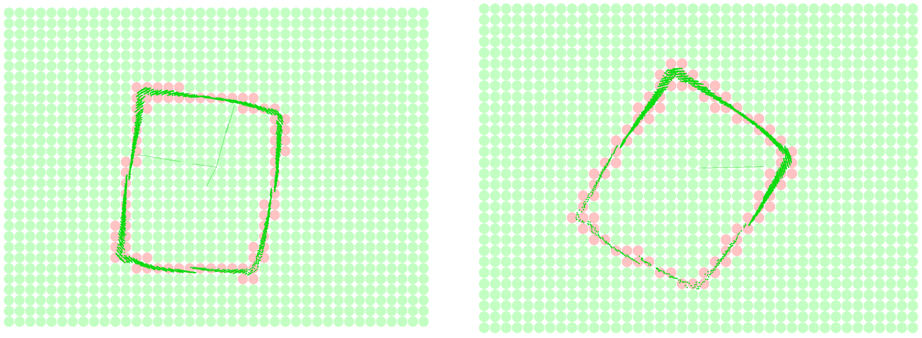


Figure 4.2: Initial 1m square test clockwise.

The curves shown in the corners were found to be caused by a close match between the friction of the wheels and the friction caused by the rear caster ball. The caster ball is like the ball in a ballpoint pen, it can roll in all directions. The robot turns around the caster ball axis when the wheels lacks grip and it turns around the midpoint between the wheels if the wheel grip is higher than the rear caster ball. The floor at the test-lab is quite soft which causes extra friction, especially at the rear caster ball. The battery, which is almost 300 grams is also placed on top of the caster ball, which worsen this problem. At harder surfaces, this is less of a problem, but the wheels can still spin.

The continuously increasing error in heading was found to be caused by a slow drift from the gyro, and it takes a couple of minutes before it is visible. The drift in heading has not been found mentioned in any earlier reports. Besides navigation, the heading is also used in the calculation of the x and y distances to detected objects. This causes the data sent to the server to drift accordingly.

In order to analyse the drift in heading further the robot was placed in a 40cm x 50cm box. By recording the reported map of the box at the server application for 1 hour the drift became more visible. The results from this test is shown in fig. 4.3. Note the robot's front point to the right in the figures. The total heading change during the 40 minutes between fig. 4.3a and fig. 4.3b is $-20°$. These results led to the introduction of chapter 5.


(a) Box mapping start.

(b) Box mapping 40 minutes later than fig. 4.3a

Figure 4.3: Box mapping used to analyse drift in heading.

Chapter 5

Gyro calibration

During the initial testing a small drift in the robot's estimated heading, $\hat{\theta}_r$, was noticed. $\hat{\theta}_r$ is the estimated heading internally on the robot. The drift appeared in both directions. The first thing the robot does when it is powered up is to run the calibration sequence to calculate the offsets of the accelerometer's x and y axis and the gyroscope's z-axis. All three offsets are calculated from an average of 300 samples. The robot's position has not shown any signs of drift, so the accelerometer readings are not investigated.

Equations 5.2, 5.2 and 5.3 summarizes how the raw data from the gyro translates to the robot's heading in the EstimatorTask. In the software radians are used in equation 5.3, while equation 5.1 and equation 5.2 are in degrees/second.

$$\dot{\theta}_{g,offset} = \frac{1}{n}\sum_{k=0}^{n}\dot{\theta}_{g,raw}[k], \quad n = 300 \tag{5.1}$$

$$\dot{\theta}_g[k] = \dot{\theta}_{g,raw}[k] - \dot{\theta}_{g,offset}, \quad k > n \tag{5.2}$$

$$\hat{\theta}_r[k] = \hat{\theta}_k[k], \quad k > n \tag{5.3}$$

The k denotes which step the EstimatorTask is at and the time between the steps are the period of the task, which in this case is 40ms. $\dot{\theta}_{g,raw}[k]$ is the raw data from the gyro at step k. The estimate from the kalman filter at step k, $\hat{\theta}_k[k]$, is calculated from both encoder ticks and $\dot{\theta}_{g,raw}[k]$. $\theta_r[k]$ is the heading the robot thinks it has and is used throughout the software as a global variable. The drift is caused by a mismatch between $\dot{\theta}_{g,raw}[k]$ and $\dot{\theta}_{offset}$ in equation 5.2, causing $\dot{\theta}_g[k]$ to be unequal to 0.

5.1 IMU calibration

In order to check the calibration, a calibration algorithm for the Inertial Mesurement Unit (IMU) written by Ródenas, found at [12], was implemented on an Arduino Nano which was connected to the IMU using I2C. This algorithm averages the offsets over several calibration runs. The offsets for the gyroscope's z-axis was the only one used. The implementation of the resulting offset still showed drift. As stated in [12], temperature is a challenge to the IMU, the offset may change with temperature. Since the drift is still present, the results from this calibration is not used further.

5.2 Gyro data logging

In order to evaluate different solutions to solve the drift problem, data from the calibration, the gyro and the kalman filter was logged using the NRF_LOG_INFO() function and the J-Link RTT Viewer's data logging capabilities. This makes a text file of the content printed to the terminal. To visualize the logged data a matlab script was made. The logged data is shown in figure 5.1 and figure 5.2.

Figure 5.1 illustrates the uncompensated drift in estimated heading. This figure has a data point every 1.6 seconds. $\hat{\theta}_g$ is the sum of the angular rates from equation 5.2, which for a discrete system is calculated as:

$$\hat{\theta}_g = \sum_{k=0}^{n} \dot{\theta}_g[k]\Delta T \tag{5.4}$$

where $\hat{\theta}_g$ is the estimated heading from the gyro. The n denotes the total number of steps, while k is the current step. ΔT is the time between each step. Eq. 5.4 was made during this thesis to compare the sum of angular rates from the gyro to the estimated heading by the kalman filter.

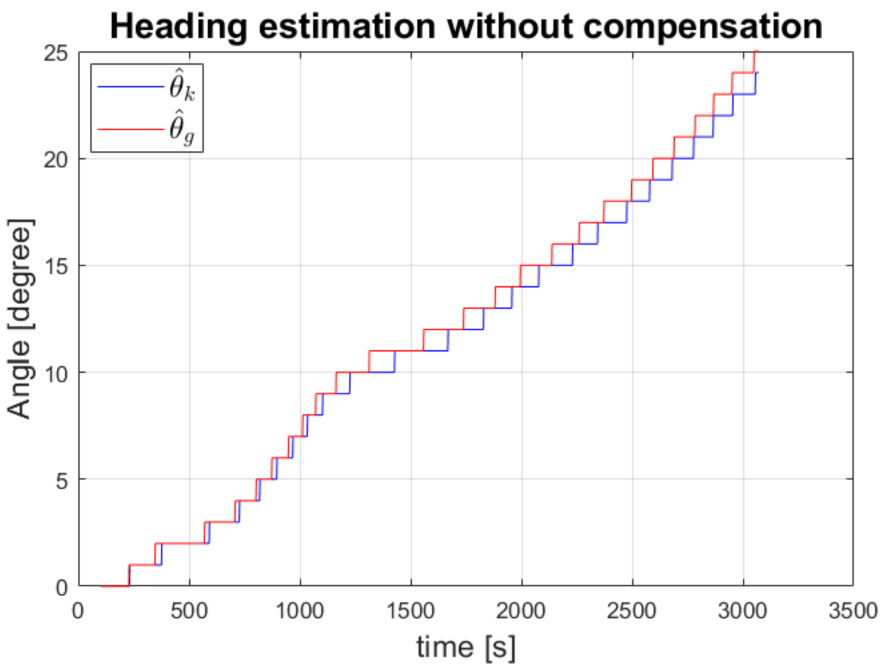


Figure 5.1: Uncompensated heading estimate from kalman filter and gyro summation.

The most interesting observation in figure 5.1 is the time between the updates of $\hat{\theta}_g$ and $\hat{\theta}_k$. Both are updated at the same time with the same angular rate from the gyro and no encoder ticks are affecting the kalman filter. $\hat{\theta}_k$ is increasing its time to update the angle compared to $\hat{\theta}_g$. From 10 degrees to 11 degrees, the $\hat{\theta}_k$ is updated 116 seconds later than $\hat{\theta}_g$. For the last two updates shown in the figure, the $\hat{\theta}_g$ is two degrees ahead of $\hat{\theta}_k$. The logging was manually stopped at 3100 seconds. The mean of the estimates up to 3050 seconds from figure 5.1 are shown in table 5.1.

Table 5.1: Mean values of $\dot{\hat{\theta}}_k$ and $\dot{\hat{\theta}}_g$.

Parameter:	**Value [deg/sec]:**
$\bar{\dot{\hat{\theta}}}_g$	0.0082
$\bar{\dot{\hat{\theta}}}_k$	0.0075

A closer inspection to the cause of the drift problem is shown in figure 5.2. This is logged at a 40ms interval and some precision is lost due to float to int conversion, which also causes several of the values of the raw data to have the same value. Several logs were made and the data presented in figure 5.2 is the data with the largest difference between the highest and lowest $\dot{\theta}_{g,raw}$. The highest and the lowest readings from $\dot{\theta}_{g,raw}$ are marked with circles. $\dot{\theta}_{g,offset}$ presented as the red line is the mean from the calibration sequence shown in equation 5.1. The mean of $\dot{\theta}_{g,raw}$ from the figure is the $\bar{\dot{\theta}}_{g,raw}$ shown as the blue line. The respective values are shown in table 5.2.

Table 5.2: Values from figure 5.2

Symbol:	Value[deg/sec]:
$\dot{\theta}_{g,offset}$	-0.257
$\bar{\dot{\theta}}_{g,raw}$	-0.2488
$\dot{\theta}_{g,high}$	-0.183
$\dot{\theta}_{g,low}$	-0.305

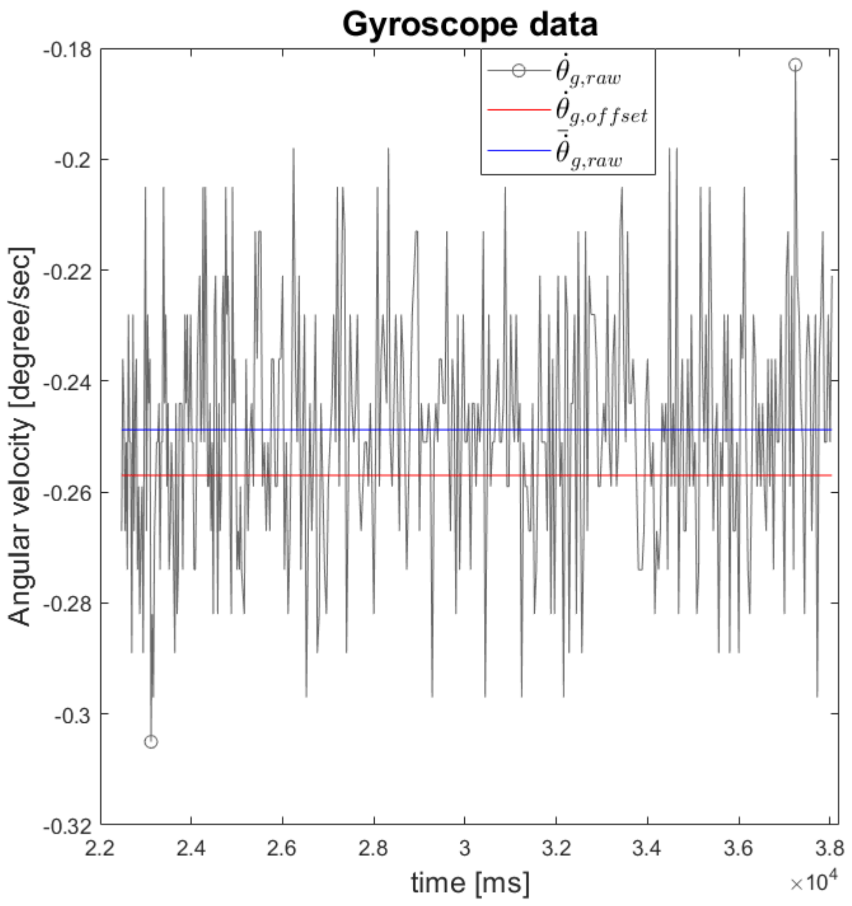


Figure 5.2: Gyroscope raw data and offset.

5.3 Gyro calculations and results

The edge cases for the highest and the lowest $\dot{\theta}_g$ gives the following maximum and minimum rates:

$$\dot{\theta}_{g,min} = \dot{\theta}_{g,low} - \dot{\theta}_{g,offset} = -0.048 \left[\frac{deg}{s}\right] \tag{5.5}$$

$$\dot{\theta}_{g,max} = \dot{\theta}_{g,high} - \dot{\theta}_{g,offset} = 0.074 \left[\frac{deg}{s}\right] \tag{5.6}$$

Ensuring that $|\dot{\theta}_g|$ is set to zero when it is between the min limit $\dot{\theta}_{g,min}$ and the max limit $\dot{\theta}_{g,max}$ the cause of drift is possibly eliminated. In order to reduce the drift for any spikes outside the min and max, the limit it set to 0.1. The $\dot{\theta}_g$ is in the software set as in equation 5.7 for all k steps above 300:

$$\dot{\theta}_g[k] := \begin{cases} 0.0, & \text{if } |\dot{\theta}_g[k]| < 0.1, \quad k > 300 \\ \dot{\theta}_g[k], & \text{else}, \quad k > 300 \end{cases} \tag{5.7}$$

The particular case shown in figure 5.2 gives a mean angular rate of:

$$\bar{\dot{\theta}}_g = \bar{\dot{\theta}}_{g,raw} - \dot{\theta}_{g,offset} = 0.0082 \left[\frac{deg}{s}\right] \tag{5.8}$$

Observe the difference between $\bar{\dot{\theta}}_g$ from eq. 5.8 and $\bar{\dot{\hat{\theta}}}_k$ from table 5.1.

The result from equation 5.8 matches the mean for $\bar{\dot{\theta}}_g$ in table 5.1, while $\bar{\dot{\theta}}_k$ is noticeably lower. This led to a replacement of equation 5.3 to:

$$\hat{\theta}_r[k] = \hat{\theta}_g[k] = \sum_{k=300}^{n} \dot{\theta}_g[k]\Delta T \tag{5.9}$$

This means the robot's internal heading is now solely based on the sum of angular rates from the gyro. Throughout this thesis this replacement has been used.

Figure 5.3 shows how the estimated angles from the gyro, $\hat{\theta}_g$, and the kalman filter, $\hat{\theta}_k$, developed when the compensation from eq. 5.7 was applied.

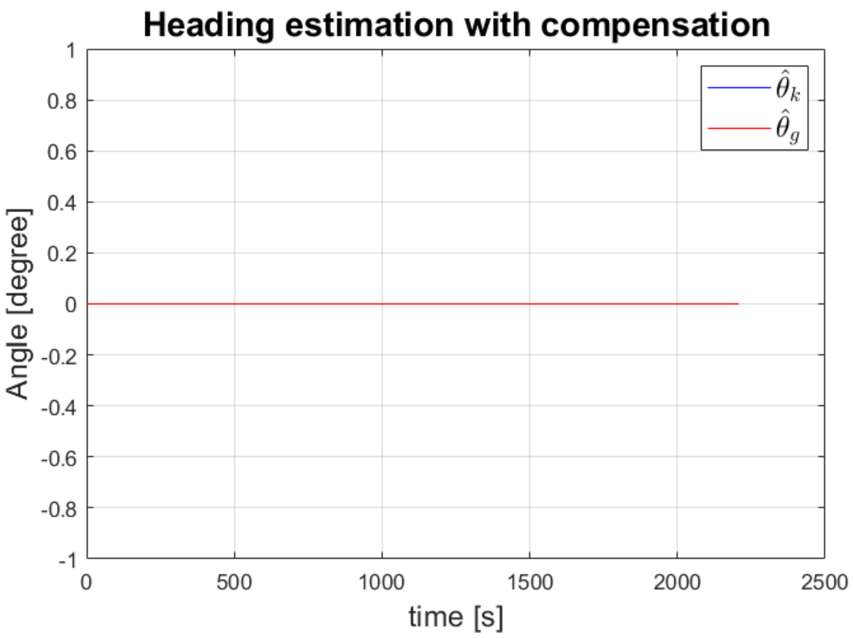


Figure 5.3: Compensated heading estimate from kalman filter and gyro summation.

Chapter 6

Hardware

This chapter will cover how installation of new hardware is integrated to existing hardware and software. A section about calibration of already existing hardware is also covered.

6.1 Motors, wheels and encoders

A number of weaknesses has been presented in earlier reports on the motors, wheels and encoders for this particular robot. The motor, gearbox, wheel and encoder package will also be referred to as the powertrain. Based on the weaknesses the main focus points when looking for the new powertrain was:

- Wider wheels for increased grip
- Compact and tough gearbox
- Quadrature encoder

The new powertrain was bought as a complete set, including motors, wheels, encoders and mounting brackets. The specifications for the new powertrain is found in figure 12.1 in the Appendix. Figure 6.4 shows a comparison of the previous and new powertrain. The following subsections will cover the specifications of the new drivetrain and what had to be done to get it to work with existing software and hardware.

6.1.1 New wheels

The new wheels are 27mm wide compared to the 14mm of the old ones. In addition the contact patch of the new tires adds a lot more grip to the surface. The increased grip reduces the need for ramp-up and ramp-down functions in the controllers, which was used to avoid wheelspin with the previous powertrain. The new diameter and circumference is

68mm and 214mm respectively, which is defined in the constants WHEEL_DIAMETER_MM and WHEEL_CIRCUMFERENCE_MM in the software.

6.1.2 New motors and gearboxes

The new dc motor and the gearbox comes as one unit with a sealed gearbox with internal gears. The dc motor is rated for 6 volts, but the battery is 11.1 volt. In order to avoid damage to the motor caused by overvoltage the duty cycle of the pwm signal going to the motor controller was limited to 50%, which equals 5.55 volts. This is limited with the constant MAX_DUTY. From the specifications found in the Appendix the no-load speed of the wheel at 6 volts is 210 rpm. With a wheel circumference of 214 mm rotating at 210 rpm, the linear velocity of the robot becomes:

$$210 \left[\frac{rev}{min}\right] \cdot 214 \left[\frac{mm}{rev}\right] = 749 \left[\frac{mm}{s}\right] \tag{6.1}$$

Since this is the no-load linear velocity, the reality is slightly lower, but it is still high. This is way faster than the robot needs to move so the MAX_DUTY is mainly to avoid overvoltage of the motor. In the controllers the duty cycle is limited further, which is described in chapter 7.

In the specifications the gearbox ratio is stated to be 34:1. Which means 34 motor revolutions is needed for one revolution at the output shaft connected to the wheels.

6.1.3 New encoders

The previous encoder setup consisted of a magnetic disc at the rear end of the dc motor shaft and a hall-effect sensor placed close to this disc to detect ticks. Because of a loose fit, the hall-effect sensors could slide out of position from small vibrations and shocks. This caused the robot to gradually loose ticks which caused it to drive further than it should to compensate.

The new encoders consists of two hall-effect sensors that are soldered on a small pcb at the rear of the motor which offers high vibration tolerance and no problems with missing pulses. The new encoders consists of two sensors placed 90 degree apart, known as a quadrature encoder. In addition the new rotating magnetic disc offers 11 ticks for one revolution of the dc motor compared to the previous one which had 4 ticks per motor rotation. Figure 6.1 shows the difference between the encoders.

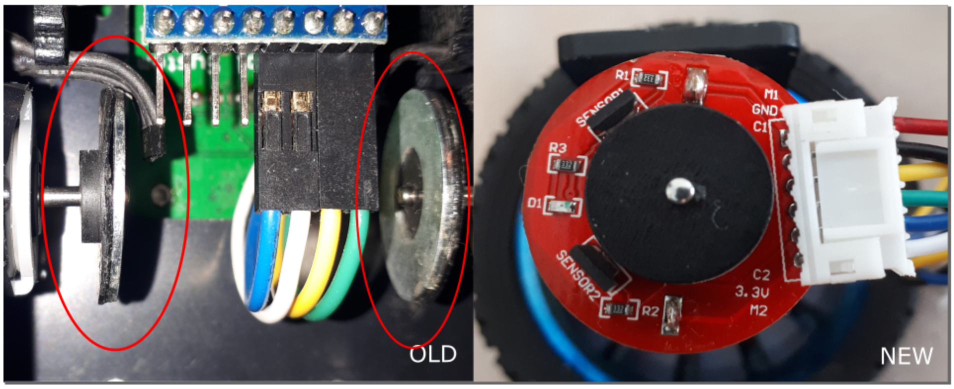


Figure 6.1: A comparison of previous and new encoders.

Table 6.1 shows a description of what the pcb text from figure 6.2 means and the associated wire color to the motor and encoder.

Table 6.1: Motor wiring.

Description:	**Pcb text:**	**Wire color:**
Motor input 1	M1	Red
Sensor Ground	GND	Black
Sensor1 output	C1	Yellow
Sensor2 output	C2	Green
Sensor Voltage	3.3V	Blue
Motor input 2	M2	White

The benefit of a quadrature encoder is that direction of rotation can easily be obtained because of the phase difference between the two encoder outputs. Figure 6.2 illustrates how the two hall-effect sensors outputs are phase shifted by 90 degrees. When the sensor outputs a high signal, this corresponds to the voltage applied to the hall-effect sensors, which for this particular setup is 3.3V, while low is 0V/ground. Depending on the direction of the rotation one sensor will be outputting its pulse 90 degrees after the other. If moving from left to right in the figure, Hall 1 is leading Hall 2 and the opposite happens when the motor rotates in the other direction.

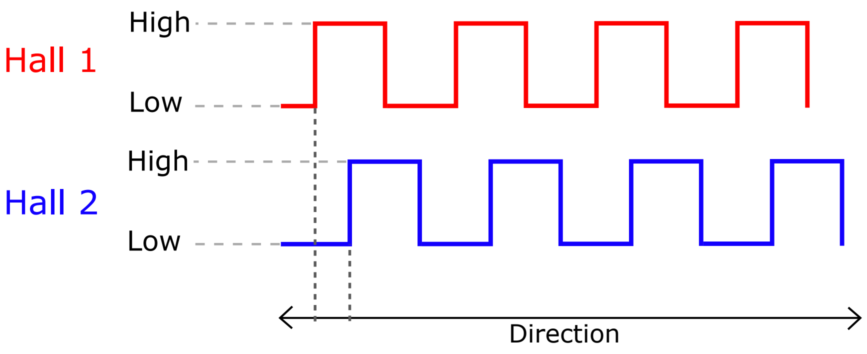


Figure 6.2: Phase shift of a quadrature encoder.

A limiting factor with this setup is that an additional two input pins on the microcontroller is needed. The SLAM Control System pcb only supports one encoder input per motor. Because of this, the output from the second encoder is currently not connected.

A constant named WHEEL_FACTOR_MM is used by the microcontroller to calculate the distance driven based on the number of ticks recorded. WHEEL_FACTOR_MM is measured in [mm/tick]. The number of encoder ticks and the gearbox ratio gives a total of $11 \cdot 34 = 374$ ticks per wheel revolution. So the new WHEEL_FACTOR_MM becomes:

$$WHEEL\_FACTOR\_MM = \frac{214}{374} = 0.57 \left[\frac{mm}{tick}\right] \tag{6.2}$$

where 214 is the wheel circumference from subsection 6.1.1.

6.1.4 Installation

The decision on where to install the new powertrain on the robot frame is based on several considerations. There are several benefits with moving the powertrain to the center of the robot frame:

- Relieve the rear caster ball from some of the battery weight to reduce the friction at the rear when turning.
- Reduce the chances of the rear hitting obstacles during turning.
- Reduce the tracking problem illustrated in figure 6.3.
- Avoid distance to object compensation because of misalignment between sensor-tower and turning axis.

The placement is not a big issue for the tracking since the tracking sphere, the blue dot, can be manipulated by relocating the reflective spheres or choose another selection of reflective spheres to make the rigid body for tracking. The main issue is when the robot is mapping its surroundings because the distance from the red dot to the blue dot currently is not compensated for in the software. This causes the readings to be about

40mm wrong after a turn. It can however be compensated for in the x and y calculation to the objects by adding ROBOT_AXEL_OFFSET_MM $\cdot \cos(\hat{\theta}_r)$ in the x axis and ROBOT_AXEL_OFFSET_MM $\cdot \sin(\hat{\theta}_r)$ in the y axis. $\hat{\theta}_r$ is the robots estimated heading and ROBOT_AXEL_OFFSET_MM is the distance between the blue and red dot.

Figure 6.3 show a more detailed illustration for the curves noticed during the camera tracking in section 4.2. The figure illustrates a 90 degree left turn for a counter-clockwise run, where situation 1 is before the turn and situation 2 is after. The red dot represents both a waypoint and the turning axis. The blue dot represents the tracking sphere from the upper picture in figure 2.4. The blue line is the tracked path from the camera system and the red line the path to follow between the waypoints, as in figure 4.1. The robot turns by applying the same voltage, but in opposite directions to the motors to turn, thus the turning axis is between the wheels. A real view of the robot with this setup is shown in figure 2.1.

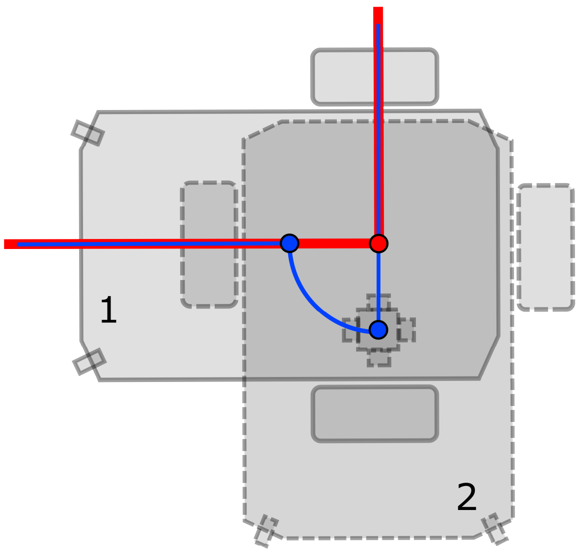


Figure 6.3: Tracked path when there is a mismatch between the turn axis and the tracking point.

Since the sensortower, the center of the robot and the tracking sphere already is aligned with the center of the frame, the easiest thing is to align the new powertrain with this. This alignment makes the tracking path look smoother and there is no need to compensate the distance measurements since the red and blue dot will be aligned in the centre of the robot frame. The new powertrain mounted on the frame can be seen in fig. 6.4. While the robot was dismantled, the IMU was moved from the lower part of the frame to the upper part and also relocated to the center of the frame. This makes it easier to dismantle the robot in the future since the only wires going to the bottom part of the frame is the wires to the motors.

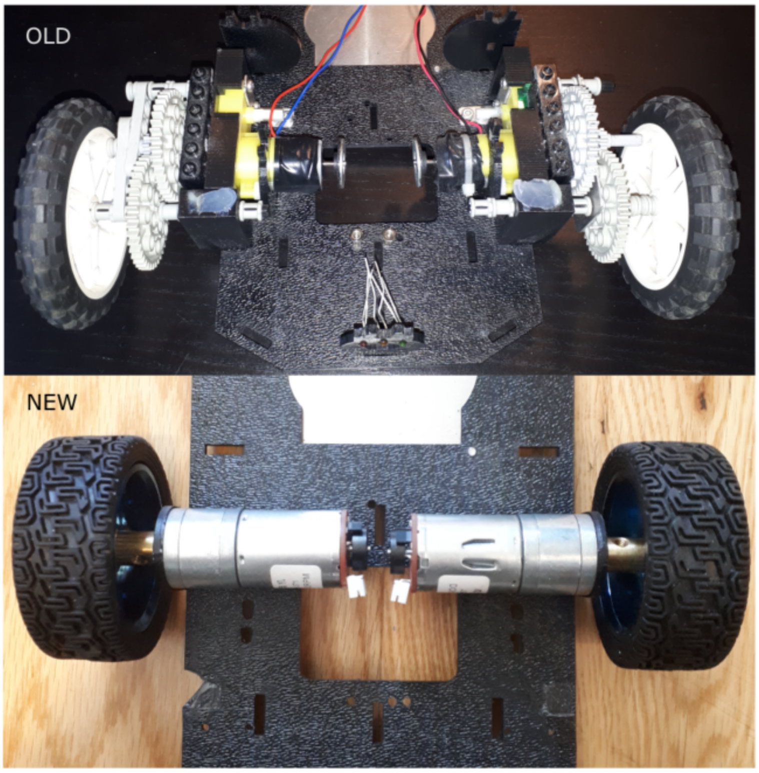


Figure 6.4: Previous motor setup compared with the new.

6.1.5 Testing and results

The first test to check the new drivetrain was a distance test. The floor was marked with two pieces of tape one meter apart. The robot was started at one piece of tape with its front pointing towards the other. By giving the robot a waypoint at [1000, 0], it should go 1000mm straight ahead and stop. However the robot passed the waypoint by over 200mm.

This was caused by a mismatch between the gear ratio and/or the number of encoder pulses and the calculated WHEEL_FACTOR_MM from eq. 6.2. Since the robot drove further than the one meter mark, and the only distance reference it has is encoder ticks, the calculated WHEEL_FACTOR_MM from eq. 6.2 was too low. The two things that could be the cause of this is would either be number of encoder ticks, or that the gear ratio is lower than what is specified in the specification in fig. 12.1.

In order to test the number of encoder ticks per motor revolution, a simple schematics was made and connected to the output of the hall-effect sensors. This is shown in figure 6.5. By manually rotating the motor and counting pulses visualized by the leds, the 11 ticks were verified to be correct according to the specifications.

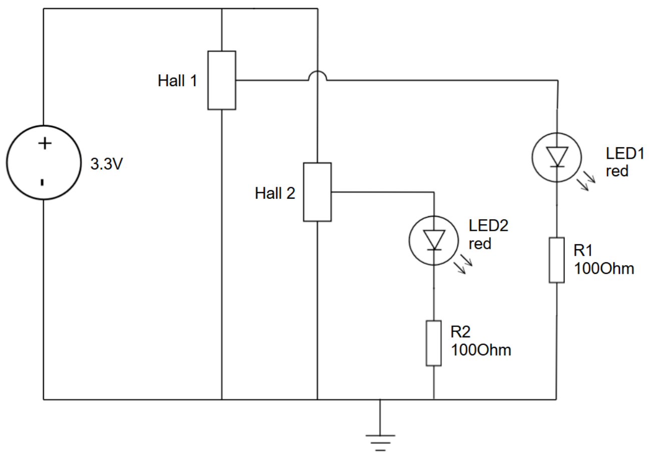


Figure 6.5: Encoder schematics for counting encoder ticks.

A similar method was used to check the gear ratio. By marking both the motor and the wheel and manually turning the motor until one revolution of the wheel was completed revealed that the actual gear ratio is 27:1, not 34:1. The new encoder ticks per wheel revolution is $11 \cdot 27 = 297$, which corrects the WHEEL_FACTOR_MM from eq. 6.2 to:

$$WHEEL\_FACTOR\_MM = \frac{214}{297} = 0.72 \left[\frac{mm}{pulse}\right] \tag{6.3}$$

The robot has a 15mm radius circle around the waypoint it is headed to, where it stops the controller. It stopped inside this circle with the correction from eq. 6.3. Going straight ahead with the previous controller showed to be no problem, but turning and keeping a steady course to the next waypoint caused some unwanted results.

After hours of tuning and testing the previous controller, fig. 6.6 shows the best result from a 1 meter square test, that was achieved with the new motor setup and the previous motor controller. This result forms the foundation for chapter 7.

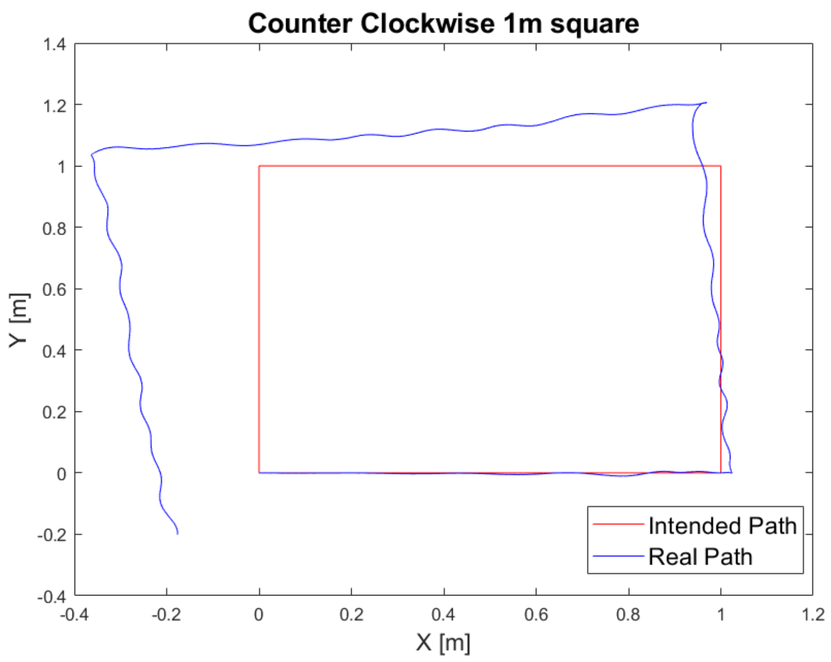


Figure 6.6: 1 meter square test with new drivetrain and old controller.

6.2 Ir sensors

During the implementation of a for-loop to read the four sensors distances, it was noticed that ir-sensor 3 was connected to input 4 and ir-sensor 4 to input 3. The wires were switched and the software changed to accommodate the changes. The wires for ir-sensor 1 and 2 suffered from metal fatique because they had a solid core. All four wires were changed to the same multi core to be better suited for the applied movement. A change in wires also changes the resistance. The voltage close to the max detection distance of the sensors is low, so the change in resistance can possibly play a key role in accurate distance measurements. A new calibration was performed and the process is described in section 6.2.1.

6.2.1 Calibration

Calibration of the ir-sensors was performed in a similar way as in [3]. The analog sensor outputs were logged together with the distance to a white flat surface pointing towards the sensors. In this case the sensor outputs was logged from 125mm up to 800mm with 25mm increments. Curve-fitting the data to a power function was done in excel. The function used to calculate the distances is shown in eq. 6.4:

$$d_i = a_i \cdot x_i^{b_i} \quad i = 1, 2, 3, 4 \tag{6.4}$$

where d_i is the calculated distance from the analog value x_i for sensor i. a_i and b_i are calculated constants from the curve-fitting for each of the sensors. Sensor locations are shown in fig. 2.3. The resulting curve-fitting constants in eq. 6.4 are presented in table 6.2.

Table 6.2: Ir calibration results.

Sensor:	a_i**:**	b_i**:**	R^2**:**
IR1	327138	-1.062	0.9994
IR2	444818	-1.1	0.9991
IR3	313397	-1.046	0.9985
IR4	649282	-1.154	0.9983

6.2.2 Testing and results

The test of the previous and the new calibration of the ir-sensors was conducted on the floor with the closest object 120cm away. Direct sunlight was blocked to avoid disturbances, following the manufacturers recommendations for using these sensors [13]. For reference a white round obstacle was placed at an angle 40 cm away from the sensor tower. The obstacle and its distance is indicated in the figures with a blue arrow. Two tests were completed for each calibration for comparison. The difference between the tests is a 90 degree turn of the robot. Test 2 corresponds to a 90 degree left turn of the robot. The front of the robot is pointing to the right in the following figures. An angular reference to the sensors is found in fig. 2.3.

Figure 6.7 and figure 6.8 shows the different test results from the previous calibration. The figures with the previous calibration shows some noise different angles away from the blue arrow. The 90 degree difference between the two figures would reveal if the environment caused the false detections. Notice how the edges of the object contributes to several points away from the edges.

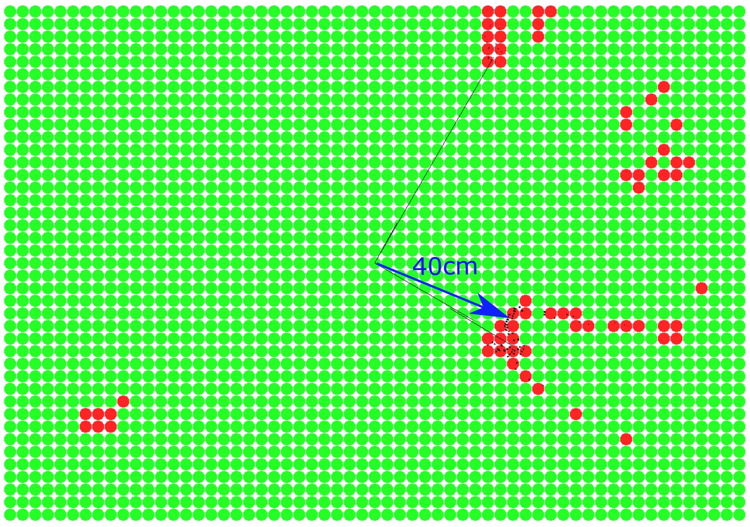


Figure 6.7: Test 1 of previous sensor calibration.

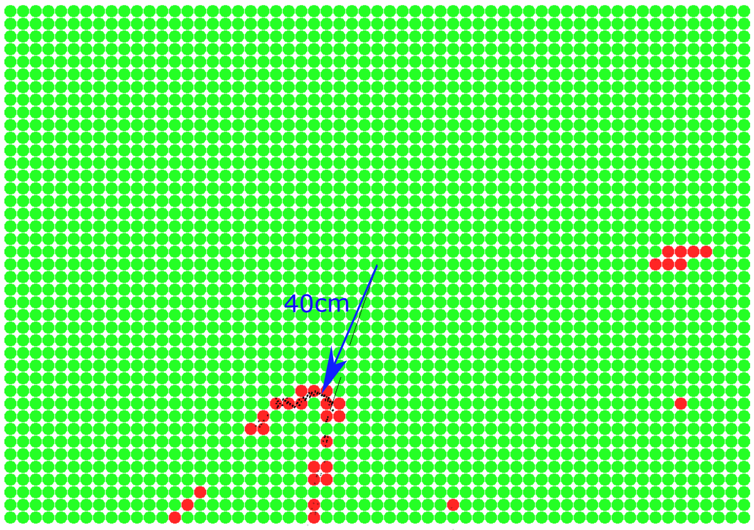


Figure 6.8: Test 2 of previous sensor calibration. Robot is turned 90 degrees left compared to fig. 6.7

Figure 6.9 and figure 6.10 shows the different test results from the new calibration. In fig. 6.9 a significant amount of false detections compared to fig. 6.7 is present. Some of the false detections also seem to be at the same angle. For fig. 6.10 compared to fig. 6.8 the false detections are slightly reduced.

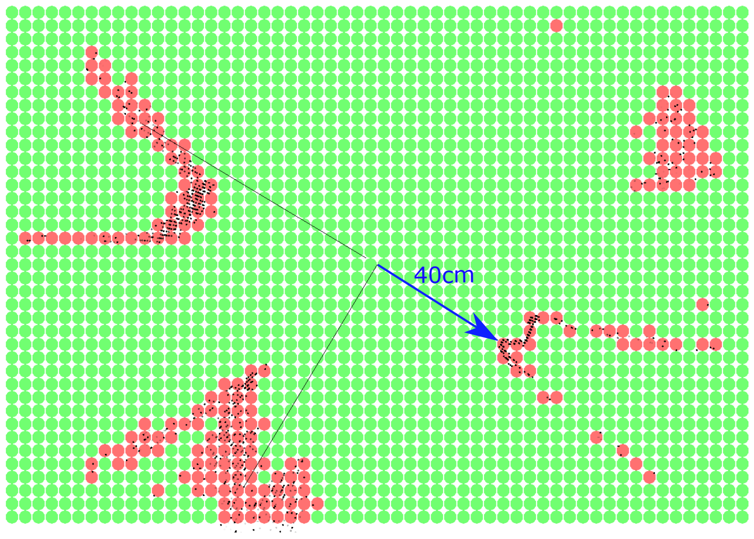


Figure 6.9: Test 1 of new sensor calibration.

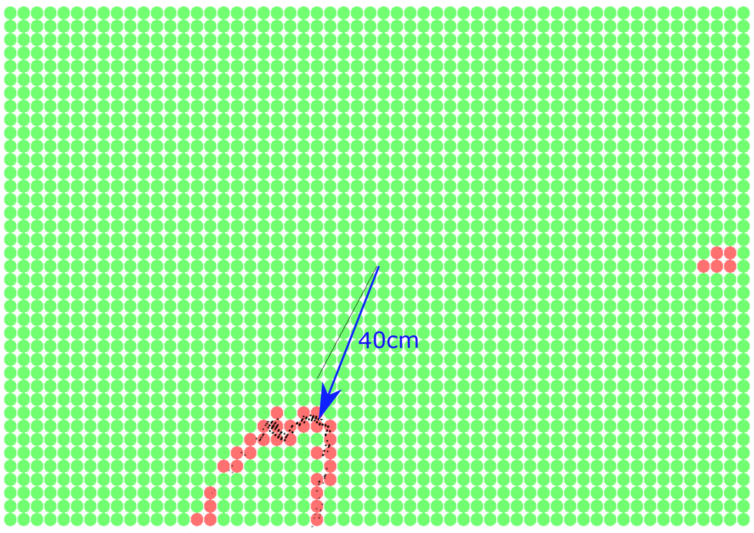


Figure 6.10: Test 2 of new sensor calibration. Robot is turned 90 degrees left compared to fig. 6.9

The previous calibration has a better overall rejection against false detections and is the calibration used further throughout this thesis.

Chapter 7

New Controllers

From the test results in section 6.1.4 the need for new controllers was revealed. The new controller design is divided into two controllers where one handles heading and one takes care of distance. Each controller is located in their own function for easier tuning of the individual controllers. The errors are passed as arguments to the functions and the functions calculates the desired PWM duty cycle for the left and right motor. Calculated duty cycle is also applied to the motors at the end of each function. The controller gains for the controllers are based on trial and error. For a new waypoint, the robot first aligns its heading before running the distance controller. The heading error has to be inside ± 2 degrees for 20 consecutive steps before it proceeds to driving forward from turning. Note that waypoints are received in cartesian coordinates and the controllers runs based on errors in polar coordinates.

7.1 Constraints

Experiments revealed that a limitation in the applied voltage to the motors was necessary. A too low voltage caused the motors not to turn, while too high voltage caused the robot to get out of control. As mentioned in section 6.1.2 the duty cycle is already constrained to avoid overvoltage of the motors. The controller constraints limits the duty cycle further. The constraints are shown in table 7.1.

For constrained systems care must be taken when an integral term is used in the calculation of the controller output to avoid integral windup.

Table 7.1: Constraints for controller output.

Description:	**Parameter:**	**Value[%]:**
Minumum duty cycle	minU	20.0
Maximum duty cycle	maxU	25.0

7.2 Heading controller

For the heading controller the output duty cycle is limited to the values shown in table 7.1. A PD controller showed sufficient performance for controlling the heading. Controller gains are shown in 7.2.

The heading error and its discrete derivative is calculated as shown in eq. 7.1 and eq. 7.2:

$$e_\theta[k] = \theta_{ref} - \hat{\theta}_r[k] \tag{7.1}$$

$$\dot{e_\theta}[k] = \frac{e_\theta[k] - e_\theta[k-1]}{\Delta T} \tag{7.2}$$

where θ_{ref} is the reference angle to the next waypoint given in a non-rotating robot frame. The non-rotating robot frame has its x and y axis parallel with the x and y axis in the initial frame, but the origin at the robot frame. $\hat{\theta}_r[k]$ is the estimated heading for the robot at step k from eq. 5.9 and $\hat{\theta}_r[k-1]$ is the estimated heading from the previous step. ΔT is the time between the steps, which is 40ms for the ControllerTask. Since θ_{ref} is constant during the steps the controller runs, eq. 7.2 reduces to eq. 7.3:

$$\dot{e_\theta}[k] = \frac{\hat{\theta}_r[k-1] - \hat{\theta}_r[k]}{\Delta T} \tag{7.3}$$

The controller output for the heading controller is calculated as shown in eq. 7.4:

$$U_\theta[k] = K_{p,\theta} \cdot e_\theta[k] + K_{d,\theta} \cdot \dot{e_\theta}[k] \tag{7.4}$$

where $K_{p,\theta}$ and $K_{d,\theta}$ is from table 7.2. While $e_\theta[k]$ and $\dot{e_\theta}[k]$ is from eq. 7.1 and eq. 7.3.

The controller output, U_θ , is limited following eq. 7.5 for all k steps:

$$U_\theta := \begin{cases} \text{sgn}(U_\theta) \cdot \text{minU}, & \text{if } |U_\theta| < \text{minU} \\ \text{sgn}(U_\theta) \cdot \text{maxU}, & \text{if } |U_\theta| > \text{maxU} \\ U_\theta, \text{ else} \end{cases} \tag{7.5}$$

where the sgn(x) function is defined as:

$$\text{sgn(x)} := \begin{cases} 1, & \text{if } x > 0 \\ 0, & \text{if } x = 0 \\ -1, & \text{if } x < 0 \end{cases} \tag{7.6}$$

The sgn(x) function is important since the robot turns by applying the same voltage output to both motors, but in opposite directions.

7.3 Distance controller

The distance controller consist of two controllers. One to control the distance and one to correct the heading while driving. The distance controller gains are shown in table 7.3.

Fig. 7.1 shows the result from having both the distance controller and the heading correction as proportional controllers. The heading correction was expanded to include an integral term to cope with the angle deviation.

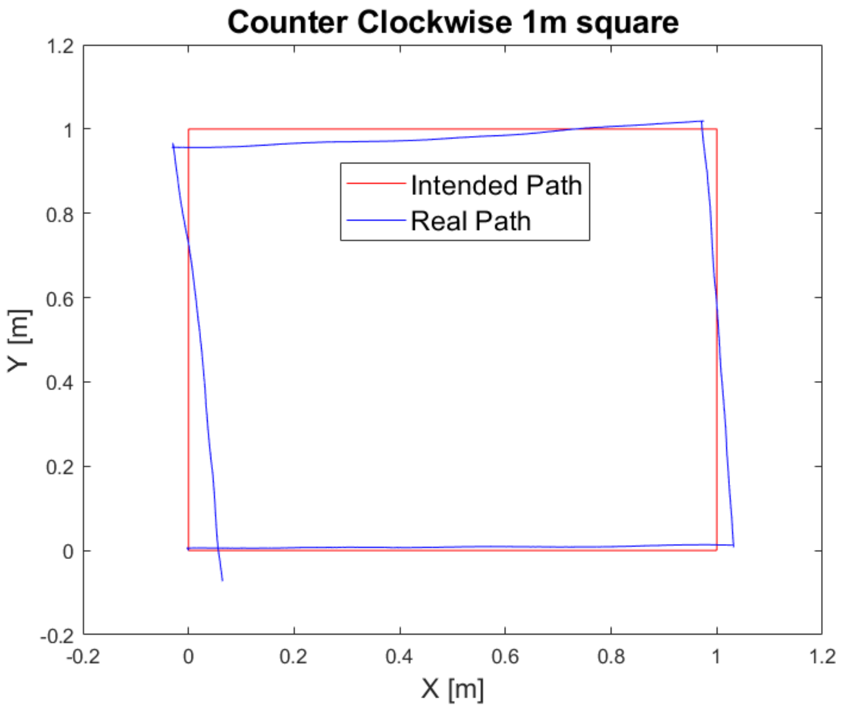


Figure 7.1: Results from proportional distance controller and proportional heading correction.

$K_{p,d}$ is the pure distance controller gain, while $K_{p,h}$ and $K_{i,h}$ is the proportional and integral gain for heading correction during driving.

The proportional controller follows the constraints in table 7.1. Since the controller gains from the two parts are summed, the total output can be outside these constraints.

The distance error at step k, $e_d[k]$, is calculated as shown in eq. 7.7:

$$e_d[k] = \sqrt{(X_{wp,i} - \hat{X}_{r,i}[k])^2 + (Y_{wp,i} - \hat{Y}_{r,i}[k])^2} \tag{7.7}$$

$\hat{X}_{r,i}[k]$ and $\hat{Y}_{r,i}[k]$ are the internal estimates of the robots position at step k in the initial frame. $X_{wp,i}$ and $Y_{wp,i}$ is the position of the next waypoint with reference to the initial frame.

The proportional distance controller output U_d is calculated as in eq. 7.8:

$$U_d[k] = K_{p,d} \cdot e_d[k] \tag{7.8}$$

where $K_{p,d}$ is found in table 7.3 and $e_d[k]$ is from eq. 7.7. $U_d[k]$ is limited as shown in eq. 7.9 for all steps k:

$$U_d := \begin{cases} \text{minU}, & \text{if } U_d < \text{minU} \\ \text{maxU}, & \text{if } U_d > \text{maxU} \\ U_d, & \text{else} \end{cases} \tag{7.9}$$

The minU and maxU values are the same as in table 7.1.

Equation 7.1 and its sum shown in eq. 7.10 form the main components of the heading correction during driving.

$$E_\theta = \sum_{i=0}^{n} e_\theta[k] \tag{7.10}$$

The n is the total number of steps the distance controller has been running from the heading controller finished, and i denotes the start-step when the distance controller starts running. When the robot reaches its waypoint and the controller is stopped, E_θ is reset to zero. $U_{d,h}$ describes the heading correction term and is calculated as:

$$U_{d,h}[k] = K_{p,h} \cdot e_\theta[k] + K_{i,h} \cdot E_\theta \tag{7.11}$$

where $K_{p,h}$ and $K_{i,h}$ is the proportional and integral gain respectively. $e_\theta[k]$ is the heading error from eq. 7.1, while E_θ is the sum of errors for the number of steps the distance controller runs. As mentioned in the introduction to this chapter, the heading controller stops when the heading is inside a ± 2 degree of the reference. $e_\theta[k]$ and E_θ are measured in radians and since they contribute to a heading correction of the motor speed, the integral term for this compensation does not cause problems with integral windup. By combining eq. 7.8 and eq. 7.11, the result is eq. 7.12 and eq. 7.13:

$$U_{d,left}[k] = U_d[k] - U_{d,h}[k] \tag{7.12}$$
$$U_{d,right}[k] = U_d[k] + U_{d,h}[k] \tag{7.13}$$

where $U_{d,left}[k]$ and $U_{d,right}[k]$ is the duty cycle applied to the left and right motor at step k.

7.4 Controller results

Since the two controllers are split into individual functions, they can be tuned separately. By setting $U_{d,left}[k]$ = $U_{d,right}[k]$ = 0 the heading controller can be tuned without interruptions from the distance controller. Trial and error resulted in the controller gains found in table 7.2, where $K_{p,\theta}$ is the proportional gain and $K_{d,\theta}$ is the derivative gain.

Table 7.2: Heading controller gains.

Parameter:	**Value:**
$K_{p,\theta}$	30.0
$K_{d,\theta}$	2.0

Figure 7.2 and fig. 7.3 are logged in the same way as the gyro data in section 5.2 but at a higher rate, every 40ms. The difference between the two figures is the distance to the waypoint. Observe the change in reference in fig. 7.2.

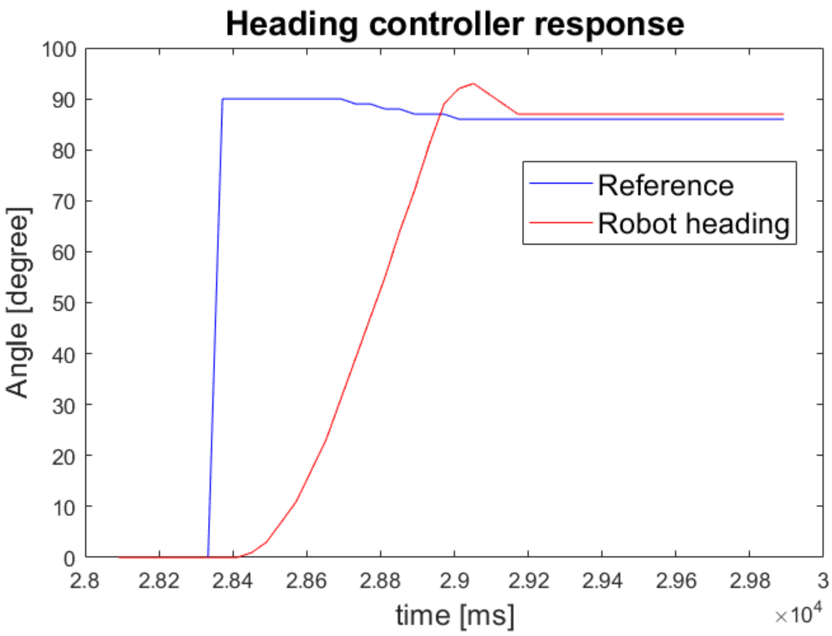


Figure 7.2: Heading controller response for a waypoint at [0mm, 100mm].

During the test in fig. 7.2 the robot changed its position to [-6, -10]. For closer waypoints the position change of the robot during the turn makes a significant change in the reference, as shown in fig. 7.2. For waypoints further away, like in fig. 7.3 the position change does not affect the reference angle. The results from a minus 90 and a 180 degree turn are shown in fig. 12.2 and fig. 12.3 in the Appendix. The oscillations shown in fig. 7.3 is mainly the result of the controller being tuned at the test-lab where larger friction at the rear caster ball is present, and the figure showing the response for harder wooden floor with lower friction at the rear caster ball. The deviation from the reference is cause by the $\pm 2^\circ$ limit.

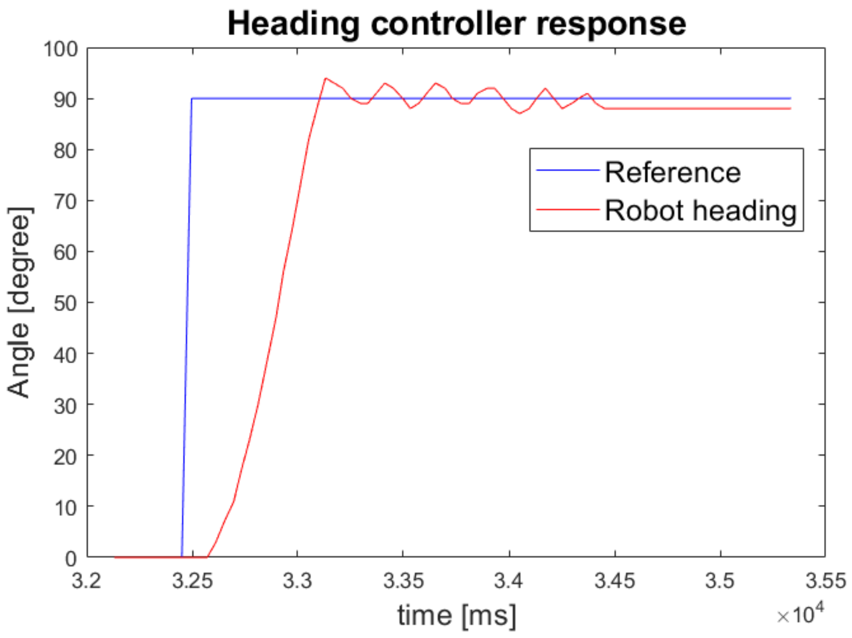


Figure 7.3: Heading controller response for a waypoint at [0mm, 1000mm].

Controller gains for the distance controller is found in table 7.3. The distance controller was tuned during 1 meter square tests at the test-lab. The result after tuning is presented as part of the square-test results presented in section 11.1.

Table 7.3: Distance controller gains.

Parameter:	Value:
$K_{p,d}$	0.03
$K_{p,h}$	60.0
$K_{i,h}$	5.0

Listing 7.1 shows a simplified implementation of the heading controller in the software.

```
void runHeadingController(thetaError, thetaDerivative){

  K_p,theta = 30.0;
  K_d,theta = 2.0;

  Utheta = K_p,theta*thetaError + K_d,theta*thetaDerivative;

  if(|Utheta| > maxU){
    Utheta = sgn(Utheta)*maxU;
  }
  else if(|Utheta| < minU){
      Utheta = sgn(Utheta)*minU;
  }

  LeftDuty = -Utheta;
  RightDuty = Utheta;

  setMotorDuty(LeftDuty, RightDuty);
}
```

Listing 7.1: Heading controller implementation in the software.

Listing 7.2 shows a simplified software implementation of the distance controller.

```
void runDistanceController(distanceError, thetaError){

  K_p,d = 0.03;
  K_p,h = 60.0;
  K_i,h = 5.0;

  thetaErrorSum += thetaError;
  U_d,h = K_p,h*thetaError + K_i,h*thetaErrorSum;
  U_d = K_p,d*distanceError;

  if(Ufwd > maxU){
    Ufwd = maxU;
  }
  else if(Ufwd < minU){
      Ufwd = minU;
  }

  LeftDuty = U_d - U_d,h;
  RightDuty = U_d + U_d,h;

  setMotorDuty(LeftDuty, RightDuty);
}
```

Listing 7.2: Distance controller implementation in the software.

Chapter 8

Software

8.1 Anti collision

This section will cover collision detection when the robot drives forward, while the next section will cover an implementation for detection when is has stopped. For collision detection the robot uses the ir-sensors. Several versions of anti collision has been tested. The last version will be presented here.

The initial idea was to check all ir-sensors while driving to detect object closing in on all sides. Before driving forward, the robot sets the sensortower to 0 degrees, which means sensor 1 is pointing forward. The layout for the other sensors is illustrated in fig. 2.3. If a collision was detected inside the collision threshold for sensor 1, the robot stopped fine as it should. However when it turned and was supposed to go another waypoint, some of the other sensors detected the collision object during the turn and stopped the robot. This would probably have worked if the turning axis had been behind the sensortower. The collision threshold for this test was set to 200mm.

The second approach was to reduce the collision check to only check ir sensor 1 while driving. When the robot has ran the heading controller and aligned its heading to the next waypoint it starts to check for collisions. The current collision threshold at 200mm had to be expanded to 250mm. The reason for this is that the robot rolls some distance after the distance controller is stopped. The second approach is illustrated in fig. 8.1. The collision threshold is presented as the black dotted line. Ir-sensor 1's detection sector is shown in red.

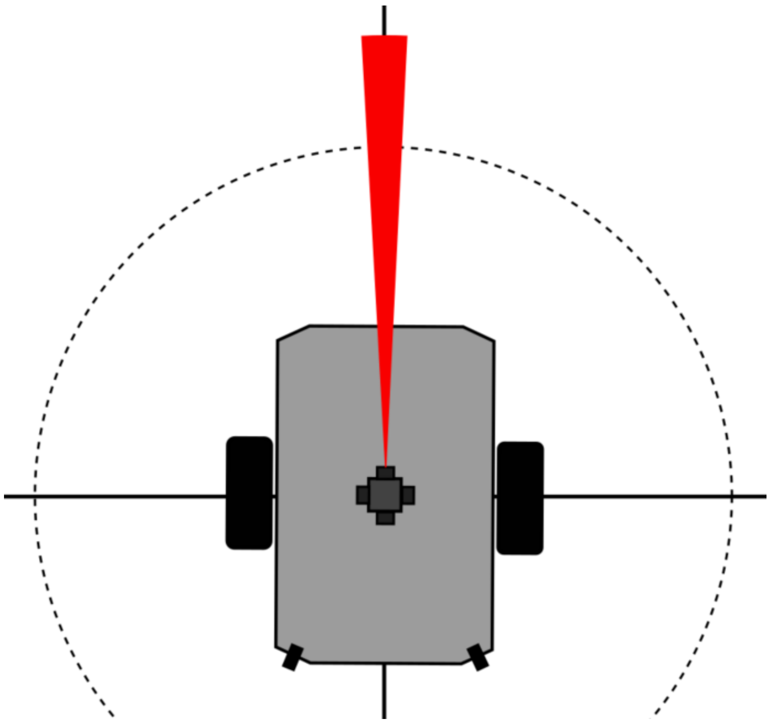


Figure 8.1: Illlustration of the current anti-collision implementation.

8.2 Collision sectors

Ir sensor 1 only covers a few degrees forward, while a detected object may cover larger parts of the front of the robot, which are not detected. The idea behind the implementation of collision sectors was to make the robot discover the whole collision object. By using the angles to a detected object, sectors where the object is can be made. Instead of implementing this functionality only for the sensors covering the forward 180 degrees, it was designed to cover all sectors around the robot. The final use of the sectors is to validate new waypoints. If a new waypoint is inside the sector, the robot can discard the waypoint without further processing.

The idea is illustrated in fig. 8.2 for an object detected ahead of the robot. The black dotted line represents the collision threshold. Collision object is shown as the black rectangle in top of the figure. The red lines and red dots are the lower and upper angles for the detection inside the threshold. Note that the situation shown consists of two sectors since the detected object crosses the 0 angle line straight ahead. The blue dotted lines represents an extension of the upper and lower limit where waypoints are discarded. The waypoint marked in green will in this case be discarded.

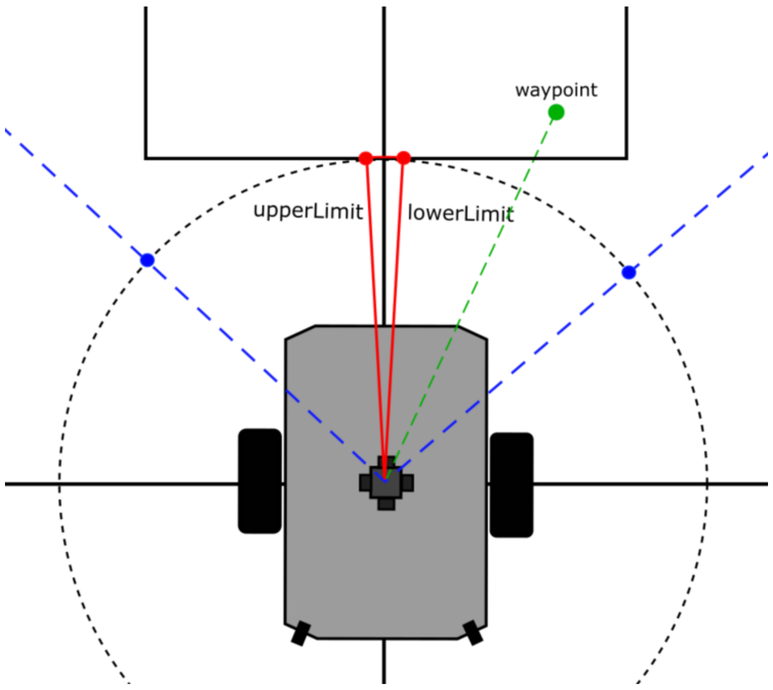


Figure 8.2: Illustration of the idea behind collision sectors.

The implementation consists of an array with 8 elements. Two elements for each ir sensor, holding the lower sector angle and the upper sector angle. The current implementation only support one sector for each sensor. If two separate collision detections appear for the same sensor they are grouped into one larger sector. Two functions was made to adjust the sectors based on the detection angle, shown in Listing 8.2 and 8.3. Listing 8.4 shows the function to validate waypoints. Note that the following listings is simplified and only covers the sector for one sensor. As mentioned earlier the array is larger for the software running on the robot.

Listing 8.1 shows the code ran in the SensorTowerTask. The SensorTowerTask runs approximately every 200ms and increments the tower angle by 1 degree each time.

```
if(detection < collisionThreshold){
    increaseSector(detectionAngle);
}else{
    decreaseSector(detectionAngle);
}
```

Listing 8.1: Collision sectors

The noCollision constant in the sector array showed in Listing 8.2 can have any value above 181 degrees. This ensures that it always is larger than the the maximum possible upper limit, which for a collision detection at 180 degrees is 181 degrees. As seen in line 9 and 10 the smallest sector possible is 2 degrees. Listing 8.2 to Listing 8.4 are located in the same source file, making the sector array accessible for all three functions.

```
1  sectorArray[] = {noCollision, noCollision};
2  
3  void increaseSector(angle){
4  
5    lowerLimit = collisionSectors[0];
6    upperLimit = collisionSectors[1];
7  
8    if(lowerLimit == noCollision && upperLimit == noCollision){
9      sectorArray[0] = angle-1;
10     sectorArray[1] = angle+1;
11   }
12   else{
13     if(angle < lowerLimit){
14       sectorArray[0] = angle;
15     }
16     else if(angle > upperLimit){
17       sectorArray[1] = angle;
18     }
19     else{
20         return;
21     }
22   }
23 }
```

Listing 8.2: Collision sectors

The decreaseSector() function splits the collisionsector into two at the angle and compares their sizes to determine which limit to update if a sector is present.

```
void decreaseSector(angle){

  lowerLimit = sectorArray[0];
  upperLimit = sectorArray[1];

  if((angle >= lowerLimit) && (angle <= upperLimit)){
    if((angle-lowerLimit) < (upperLimit-angle)){
      sectorArray[0] = angle;
    }
    else if((angle-lowerLimit) > (upperLimit-angle)){
      sectorArray[1] = angle;
    }
    else{
      sectorArray[0] = noCollision;
      sectorArray[1] = noCOllision;
    }
  }
  else{
      return;
  }
}
```

Listing 8.3: Collision sectors

The SECTOR_OFFSET shown in Listing 8.4 is a constant currently set to 60 degrees. The offset was added to expand the sector where waypoints are blocked by 120 degrees. This was introduced to avoid situations where the collision sector in the sector array is small, but the object is in reality large. Which would cause the robot to hit the object without this expansion. The sector array contains collision sectors with reference to the robot frame, but the waypoint is received with reference to the initial frame. A conversion to robot frame was added in the MainCommunicationTask before the validWaypoint() function is ran.

```
bool validWaypoint(waypointAngle){

  lowerLimit = sectorArray[0] - SECTOR_OFFSET;
  upperLimit = sectorArray[1] + SECTOR_OFFSET;

  if((waypointAngle >= lowerLimit) && (waypointAngle <= upperLimit)){
      return false;
  }
  return true;
}
```

Listing 8.4: Function for validating waypoints based on collision sectors.

Testing has shown that the collision sectors works as intended, however for small tracks the collision sectors quickly builds up and tends to block a large portion of the waypoints. Another challenge is that it takes some time to clear the sectors because of the slow turning rate of the sensortower. This led to the introduction of the third configuration variable, validateWP, introduced in section 2.5. If this variable is set to false, the MainCommunicationTask processes all new waypoints by not running validWaypoint(). If set to true, all waypoints are checked against the sector array. increaseSector() and decreaseSector() runs independently from this variable.

8.3 Code refactoring

Because the software base on the robot is very similar to software ran on the other robots, there is a limit to how much it should be changed. Experiments has been done with changing some parameters for the tasks in FreeRTOS, but has not been permanently changed. Other changes to the software outside what is specified in the chapters of this thesis has mainly been done to increase readability and modularization by moving tasks from main to separate source files, grouping code with similar purposes and changing variable names.

Chapter 9

Server Communication

This section will mainly contain the development of the communication protocol used with v2C++. This section is written from the robots view, where sent messages is messages transmitted from the robot and received messages is messages transmitted from the server to the robot.

The development of the new communication protocol for v2C++ proved not to be as straight forward as first thought. Some research of the existing protocol for v1C++ was required. v1C++ and the dongle at the robot communicates using the OpenThread network protocol and the MQTT application protocol. However, the SLAM Control System at the robot runs a nRF52832 SoC which does not support OpenThread.

9.1 Legacy layer

The legacy layer was made to translate the MQTT messages to I2C [4]. According to section 10.2 in [4] the legacy layer currently has a limit of 5 bytes for received messages, where only the last 4 bytes are contains the received data. The first byte is used as a signal mechanism for new data. If new data is present, the first byte is set to 0x72, which is the dongles I2C address. The first byte is set to 0 if no new data.

By introducing message codes for the received messages, the message size increased to 6 bytes. The massage code is located in the second byte. Testing has revealed that 6 bytes works, but there is no guarantee increasing the size further will work without testing.

9.1.1 New message types

Because of the information mentioned about the legacy layer, v2C++ sort the messages based on the message code in the second byte, not the first.

START_POSITION, NEW_WAYPOINT and UPDATE_POSITION are new message codes introduced during this project to match the development of v2C++. Messagecodes and lengths are showed in table 9.1. The purpose of the START_POSITION message is to relocate the robot in the initial coordinate system before it starts the scanning of the environment. The server makes a map based on the environmental information the robot provides to it. Based on the map and information about the surroundings, v2C++ has the ability to estimate the robot's position. The estimated position from the server may be more accurate than the robot's internal estimates. This led to adding the UPDATE_POSITION code. Which action to perform based on message codes is implemented using a switch-case in the MainCommunicationTask, which makes it easy to add, change or remove message types.

Table 9.1: Received message codes and lengths.

Message name:	**Message code:**	**Total length[bytes]**
START_POSITION	1	8
NEW_WAYPOINT	2	6
UPDATE_POSITION	3	8

NEW_WAYPOINT is the currently message code for v2C++ that is fully operational and has been verified through testing. A general setup of the received message is found in table 9.2. Common for all message codes in table 9.1 is that byte 2 and 3 contains a x coordinate and byte 4 and 5 a y coordinate. All coordinates with reference to the initial frame. Byte 6 and 7 is only used for message code 1 and 3 and contains the heading in degrees.

Table 9.2: General format for received messages.

Byte:	Low/High byte	Parameter
0	-	0x72 or 0x00
1	-	Message code
2	Low	16-bit data
3	High	
4	Low	16-bit data
5	High	
6	Low	16-bit data
7	High	

9.2 Sent messages

The messages sent to both C++ servers contains information about the robot's position and the distance measurements from the ir-sensors. The current setup for the messages sent to the v1C++ includes the robot's position and the position to a detected object. The position to the detected objects for v1C++ is converted to the initial frame before it is sent. This message currently has a total length of 8 bytes.

9.2.1 New Message format

The main motivation for developing a new message format for v2C++ was to extract all available information given by the robot and its sensor measurements into one message. Several different formats was tested both to find the necessary parameters to send and to see if the communications could handle an increased message size. The new format was developed in cooperation with Mullins[6]. The resulting message format is seen in table 9.3 and consists of 24 bytes compared to 8 for v1C++. Transmitting this amount of data showed to be no problem during the time it was tested. During development the payload data was inspected using the MQTT.fx program described in section 3.1.3.

Table 9.3: New sent message format

Byte nr.:	Low/High byte:	Parameter:
1	-	Message code
2	Low	Robot x position change
3	High	
4	Low	Robot y position change
5	High	
6	Low	Robot heading change
7	High	
8	Low	$x_{1,r}$
9	High	
10	Low	$y_{1,r}$
11	High	
12	Low	$x_{2,r}$
13	High	
14	Low	$y_{2,r}$
15	High	
16	Low	$x_{3,r}$
17	High	
18	Low	$y_{3,r}$
19	High	
20	Low	$x_{4,r}$
21	High	
22	Low	$y_{4,r}$
23	High	
24	-	Valid detection

The first 6 bytes after the message code contains the change in position and heading from the last message sent. Byte 8 to 23 contains x and y coordinates to ir-sensor measurements with reference to the robot frame. Byte 2 to 23 consists of 11 16-bit integers. The heading is measured in radians and stored as a float locally on the robot, but is converted to degrees before it is sent. Since the I2C-bus sends one byte at a time, the 16 bit integers are split into their lower and upper bytes before sending. 16-bit is used rather than 8-bit because distance measurements for the C++ servers uses mm instead of cm, which will exceed the $(2^8 - 1) = 255$mm limit for one byte.

The conversion from 16-bit to 8-bit was converted from using pointers to using bit operations which decreased required lines of code and increased readability:

```
int16_t x
int8_t xlowbyte = (x & 0xFF);
int8_t xhighbyte = (x >> 8);
```

Listing 9.1: One 16-bit integer to two 8-bit

The new message is only sent to v2C++ when the robot has stopped. This means the position change and heading change will contain the difference from the last time the robot had a stop. For each time the SensorTowerTask is ran the angle of the sensortower is incremented by 1 degree. When the angle reaches 0 or 90 degree a 1 is sent to v2C++ to tell the server that a new 90 degree scan is starting.

The idea behind the validation byte, byte 24, is to allow the server to clear angles where the measurements exceeds the DETECTION_THRESHOLD_MM. The detection threshold is set to 800mm, which is the maximum distance for the ir sensors. The content in byte 24 is illustrated in figure 9.1. The four Most Significant Bits (MSB) is not used, while the 4 Least Significant Bits (LSB) contains a boolean value each. If ir-sensor 1 has a object detected below DETECTION_THRESHOLD_MM, bit 3 in byte 23 is set to 1. The same applies for the other sensors and the other bits. When a bit is set to 1, the x and y data corresponding to the same sensor is used by v2C++ to map an obstacle at the detected position. If the bit is set to 0, the server uses the coordinates to calculate the angle and clear that angle out to 800mm.

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
-	-	-	-	IR_1	IR_2	IR_3	IR_4

Figure 9.1: Illustration of the content in byte 24 in the new outgoing message.

9.3 Implementation

During this thesis, two separate functions were made for the transmission of messages to v1C++ and v2C++. The two functions are located in the Appendix, where Listing 12.1 shows the new function for message sending to v1C++, and Listing 12.2 shows the new function for v2C++. Note they are not simplified.

The key differences between the message transmission formats for v2C++ and v1C++ is summed up in the following bullet points:

- Message size, v2C++ uses 24 bytes compared to 8 for v1C++ to include all sensor measurements at the same time.
- v2C++ uses message codes to be able to handle different message types.
- X and y measurements are for v2C++ sent in robot-frame, not the initial-frame.
- Valid detection byte.
- Messages sent to v2C++ contains robot position change since last message, not the position.

The main difference for the received messages is the inclusion of message codes and the added length.

The MainCommunicationTask at thesis start only took care of the messages received from the Java server. During the development process for the new communication protocol for v2C++ the MainCommunicationTask was expanded in order to handle messages from all three servers.

Chapter 10

Test setup

10.1 Final 1 meter square tests

This square tests forms the final results for the robots navigation capabilities and was done in the same way as for the initial square test in section 4.1. The final square test is performed with the changes made to the robot throughout the chapters of this thesis. Compared to the initial square test, the final square test also includes logging of the robot's internal position. This test's purpose is to document the navigation performance of the robot after the applied changes. This performance is used for comparison with the initial navigation performance.

10.2 Mapping tests

The mapping tests were primarily used to test the new communication protocol with v2C++ and its SLAM implementation[6]. During the mapping tests, the robot received manually sent waypoints from the servers and reports distances to objects back to the servers. These tests were only conducted with v1C++ and v2C++. Each server application was tested for two different tracks.

10.2.1 Circular track

For the mapping test in the circular track using v2C++, the robot started inside the garage in the center of fig. 10.1. The small red circle marks the starting point. Robot is starting with the front pointing to the right. Waypoints were given around the garage until the robot completed its way around it and returned outside the opening at the garage.

Since v1C++ does not have SLAM capabilities, the garage was removed during this test to avoid getting a lot of clutter in the measurements. The robot was given 3 waypoints forming an ”L” inside the circle for this test. A picture from this particular track is found in fig. 12.4 in the Appendix.

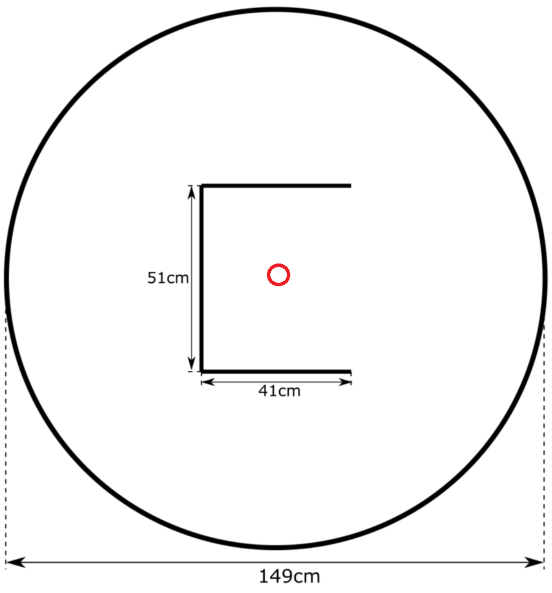


Figure 10.1: Illustration of the circular track.

10.2.2 Labyrinth track

During the mapping tests in the Labyrinth the test conditions for v1C++ and v2C++ were the same. The robot starts at the red circle in fig. 10.2 with the front facing to the right. A picture from this track is found in fig. 12.5 in the Appendix.

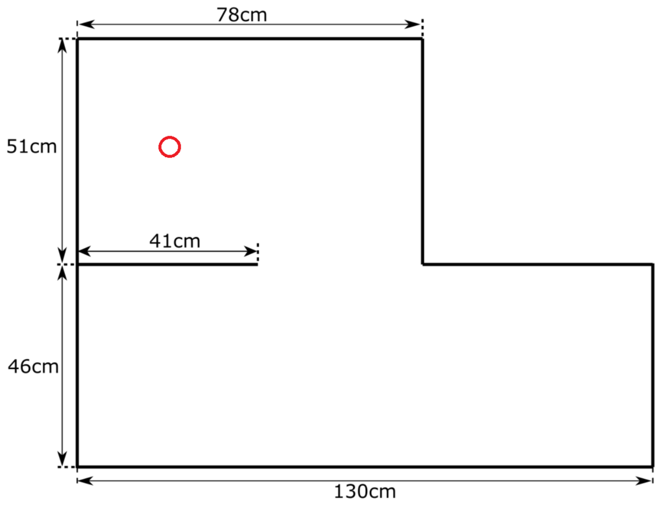


Figure 10.2: Illustration of the Labyrinth track.

Chapter 11

Results

11.1 Final 1 meter square tests

Fig. 11.1 shows the test results from the counter-clockwise run of the final 1m square test. The red and the blue line are the same as in section 4.2, the square and the tracked path respectively. The black line however is the internal position estimate from the robot. This was logged using a microSd card while running the test. The robot starts in point [0,0]. As seen, the robot's estimates are quite close to the red square, but the reality is not. Even though a integral term was added for the heading correction while driving, the blue line still looks similar to what is shown in fig. 7.1. There is more oscillations on the straight sections in fig. 11.1 compared to fig. 7.1. The distances shown by the blue line is be close to 1 meter in length, but the black line is slightly longer for every straight. Notice how the real path tends to creep inwards throughout the square. Also notice the black path looks like a parallelogram, where the straights parallel to the y axis lean left. The robot thinks it arrives close to the starting point, but the blue line stops at about 5cm from [0,0].

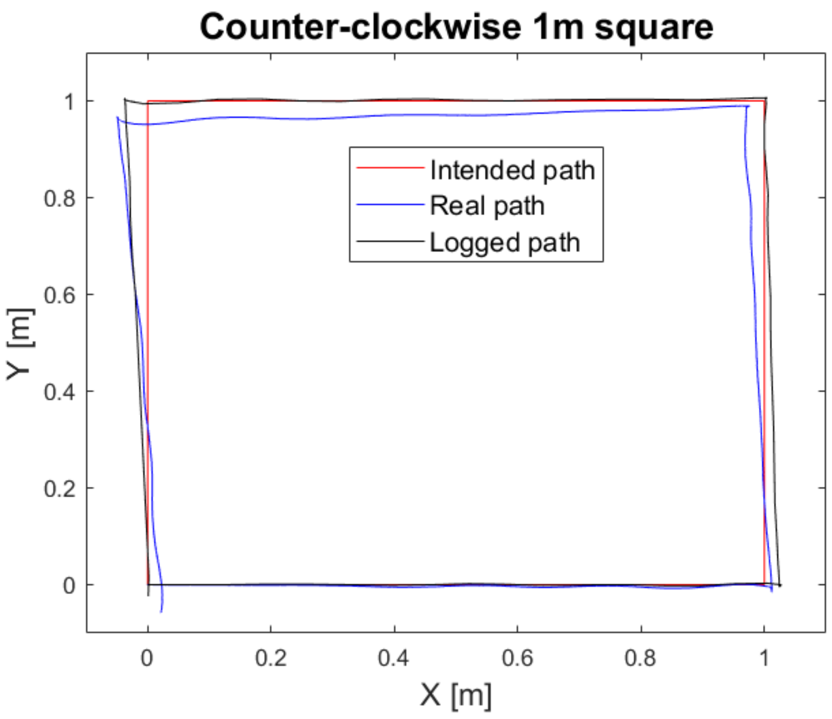


Figure 11.1: Counter-clockwise run of the final 1m square test.

The clockwise run of the final 1m square test shown in 11.2 is ran with the exact same controller tune as for the counter-clockwise run. Distances are close to being one meter. The oscillations on the straights are still present but is not as visible as in the counter-clowise run. The robot's internal position is a very close match to the red square throughout the whole test and the parallelogram shape is not seen. Unlike the counter-clockwise run the blue line now tends to lean outwards during the test.

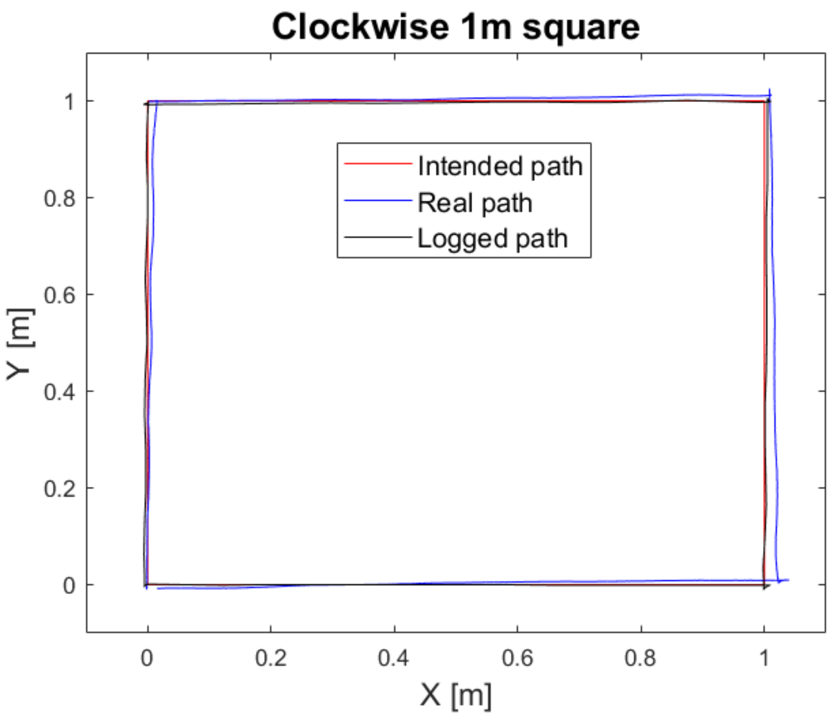


Figure 11.2: Clockwise run of the final 1m square test.

11.2 Mapping

Throughout these mapping tests the previous ir-sensor calibration from [3] was used.

11.2.1 Circular Track

Fig. 11.3 shows the resulting map from the circular track using v2C++ which uses SLAM. Notice how some of the edges of the circle is mapped into lines instead of curves, further details about the SLAM in v2C++ is found in [6]. The robot has in this figure completed a run around the garage. Notice how the points seen around the garage seems to be appearing in lines from the corners of the garage, this was also seen in section 6.2.1.

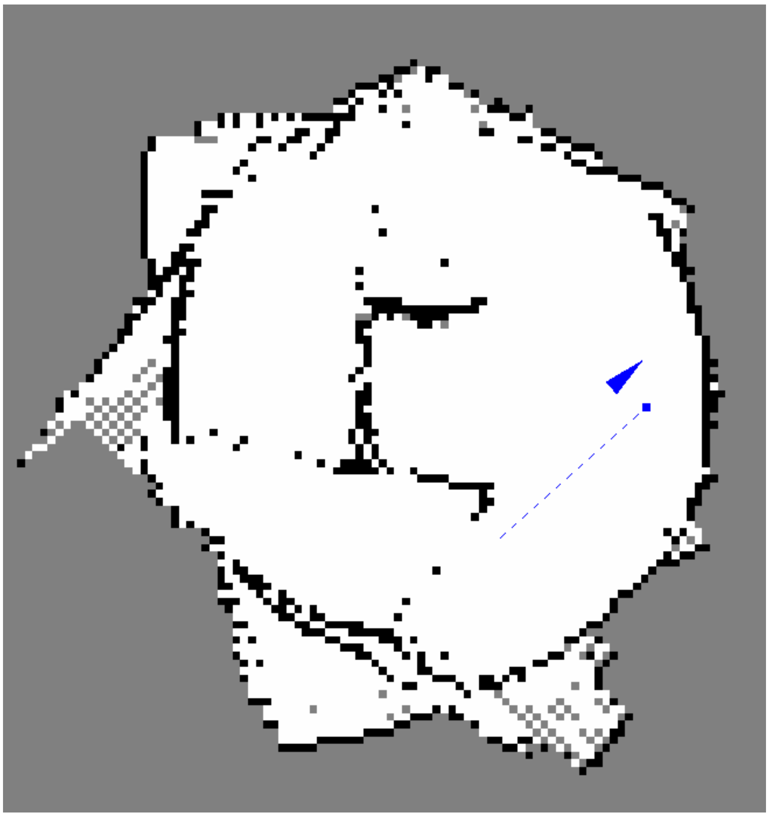


Figure 11.3: Mapping in the circular track using v2C++.

The circular track was also mapped using v1C++, shown in fig. 11.4. The black dots indicates the starting point and the two waypoints, where point 1 is the starting point. This mapping is a close match to being a circle with some deviations. The vertical line shown close to waypoint 2 is false points reported from the start position.

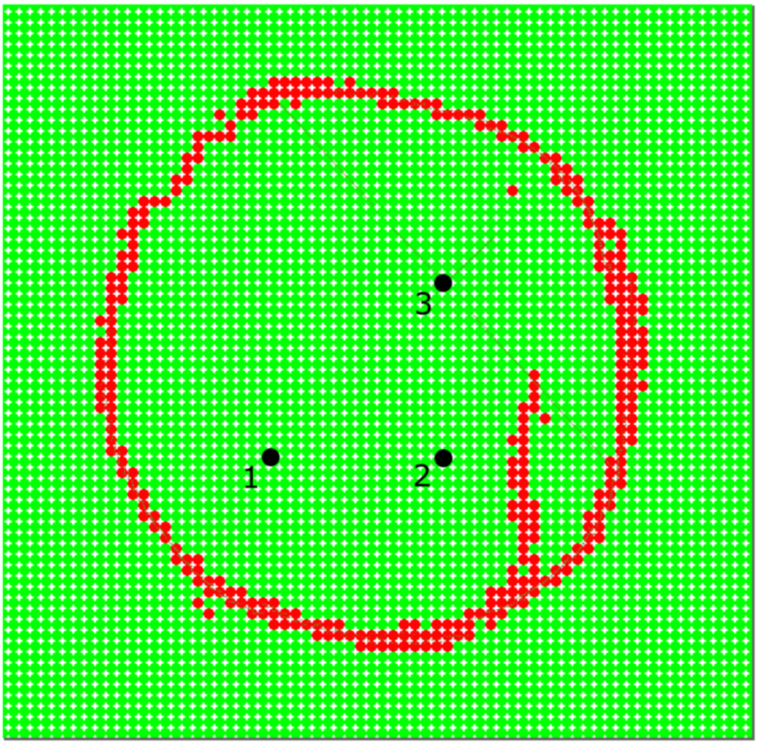


Figure 11.4: Mapping in the circular track using v1C++.

11.2.2 Labyrinth

The v2C++ shows way better performance in environments with straight walls, as in fig. 11.5. The match with the actual labyrinth is quite good. The right side of the labyrinth is not fully discovered in this figure due to the low range of the ir-sensors. An important result from fig. 11.3 and fig. 11.5 is that the new communication protocol with v2C++ works.

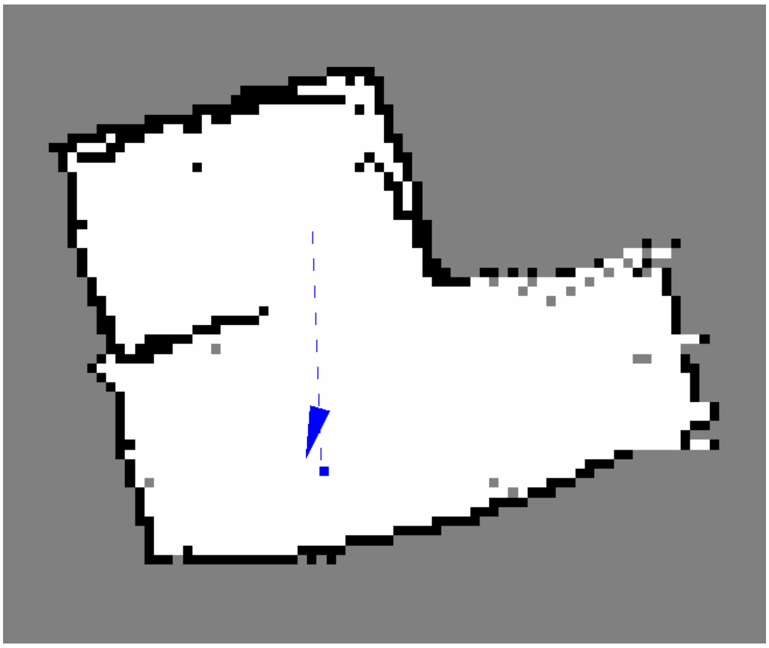


Figure 11.5: Mapping in the labyrinth track using v2C++.

Fig. 11.6 shows the same mapping test in the labyrinth, but using v1C++. Because v1C++ does not have SLAM capabilities, all detected points to objects are stored. As seen in the figure this results in a lot of false points. However, the section surrounding the starting point looks to be a good representation of a rectangle. Notice the walls in the right side of the figure, they do not align. It is not seen in the figure, but the front of the robot is pointing downwards, so the angle of wall misalignment is at 90 degrees in the robot-frame where IR1 ends its scan and IR2 starts.

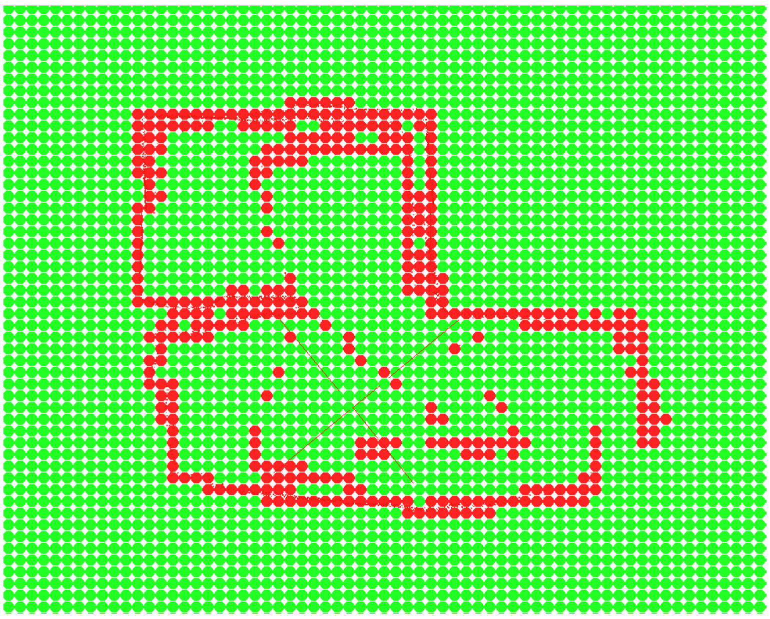


Figure 11.6: Mapping in the labyrinth track using v1C++.

Chapter 12

Discussion and future work

12.1 Discussion

12.1.1 Gyro

The estimated heading from the kalman filter has a significant lag behind the sum of angular rates from the gyro. The mean angular rates from the gyro presented in table 5.1 and in eq. 5.8 is equal, while the mean angular rate from the kalman filter is noticeably lower. This result shows that the sum of angular rates from the gyro is a better estimate for the robot's heading than the estimated heading from the kalman filter. Lacking performance of the kalman filter is probably caused by its current tune. The kalman filter is likely tuned to suppress the previously noisy updates from the gyro. This can possibly be corrected by changing the according variances for the different inputs to the filter.

12.1.2 Hardware

The new calibration of the ir-sensors showed very varying results. For the first test the previous calibration showed best performance with the lowest number of detections. The new calibration results for the first test is very noisy in most directions. Results from test 2 in the same environment shows that the new calibration has the least noise present, however the difference is only 2 detected points. The overall performance is better for the previous calibration. The problems experienced with the new calibration is likely caused by different daylight conditions between the calibration environment and the testing environment. The most interesting results are shown in fig. 6.9. The upper left corner is pointing towards the spot with the most shadow, while the lower left corner pointing towards a partially blocked window.

The powertrain including motors, gearboxes, wheels and encoders has shown good overall performance. The new wheels offer more than enough grip to the surface and no spinning has been experienced throughout testing. From the navigation results, the encoders is performing good as well even though only one of two is used per motor. The motors are strong and have no trouble turning and driving the robot even for low voltages. Gear ratio is a thing that could have been changed. With a higher gear ratio, a larger variation in the applied voltage can be used, which will provide easier control of the motors.

12.1.3 Anti-collision

The usual environments used to test these robots are quite small because of the low range of the ir sensors. Bad results was experienced from the anti-collision during testing in small environments where the obstacles and the waypoints are closer together, typically 30-40cm. The problem is that the forward movement was started before the sensortower has turned to 0 and aligned ir-sensor 1 forward. This problem's main cause is that the ControllerTask, where the controllers and the anti-collision is ran, runs 5 times faster than the SensorTowerTask, which turns the sensortower. During turning of the robot the sensortower are stopped at the last position it had. Which causes the controller and anti-collision to start before the sensortower has been turned to 0. The possible best solution to this issue is to make the SensorTowerTask align the sensortower while the robot turns.

The 200ms period of the SensorTowerTask causes some challenges for the collision sectors aswell. For every time the SensorTowerTask is ran, the sensortower is moved 1 degree. A complete 90 degree scan thus takes 18 seconds. A collision sector made early in a scan takes about 36 seconds to be cleared again. Experiments with a decreased period of the SensorTowerTask has been made, but the robot showed signs of stability issues when changing the period.

12.1.4 Navigation

The heading controller shows quite good results for all reference angles. A problem with close waypoints is that the reference moves while the robot turns, as shown in fig. 7.2. The reason for this is that the robot moves its position during a turn, even though it turns around the center axis of the frame. For a 180 degree turn, the largest position change has been 23mm in y direction. This is most likely caused by the microcontroller counting encoder ticks in wrong direction during the turn.

For the square test results shown in section 11.1 the distance controller shows quite good results. Even though the robot overshoots for several of the waypoints, the most of the lines is close to being 1 meter. The overshoot is mainly caused by the robot rolling some distance after the distance controller is stopped. This is probably also caused by the current tuning of the proportional part of the distance controller. The oscillations on the straights is likely caused by the integral gain for the heading correction in the distance controller being too high.

For both fig. 11.1 and fig. 11.2 there is a close relationship between where the robot thinks it is, marked with the black line, and where it actually is, the blue line. For both figures the blue line shows a small deviation increasingly turning the robot left. Since the integral part of the heading correction could not solve this compare to fig. 7.1, it must be caused by something else than heading deviation. The most likely cause is a speed difference between the motors for the same duty cycle, where the right motor runs faster than the left one. A contradiction to this is the robot records its movement in almost perfectly straight lines, which means both encoders should output the same number of ticks. A larger wheel circumference for the right wheel compared to the left wheel could also be a possible cause of this issue, but is unlikely.

The overall results for the navigation presented in section 11.1 compared to the initial results from section 4.2 is very good. In both the clockwise and the counter-clockwise test, the robot follows the intended path closer and arrives closer to the starting point.

12.1.5 Mapping

The results from the mapping tests is showed in section 11.2. For the circular track v1C++ created a better representation of the circle. It seemed like v2C++ was optimized for mapping straight lines since it performed quite well for the labyrinth track. The vertical line in the lower right of the circle in figure 11.4 is likely caused by noise from the ir sensors. Noise and false detections can be caused by daylight conditions, as mentioned in the datasheet[13]. A special condition showed to produce false points for every test is corners and edges. This is likely caused by the decreasing reflections being sensed even when the sensor has passed the corner/edge of an object. The scan-matching part of SLAM solves this when the robot keeps changing its position. However, if the robot stays close to a corner for some time, the same false points continues to appear, which will eventually cause problems even when using scan-matching.

12.1.6 Communication

During the mapping tests using v2C++, waypoints were sent from the server by clicking on the map and sensor readings sent using the new protocol. The mapping results shows the new communication protocol to work quite well together with v2C++. Waypoints are received in the correct manner and sensor readings are sent correct as well. Even though the current communication protocol works as intended it should be optimized further. There is for example no need to send position change if there is no position change, the position and sensor measurements should be split into separate messages. The legacy layer is also a limitation worth looking into.

The mapping results however does not represent the communication over time. Some problems were experienced during the test runs. The robot and the connected dongle share the same power switch. Several restarts are sometimes needed for the onboard dongle to connect properly to the gateway. Even when the dongle is showing to be connected, with a blue led, the transmitted messages are sometimes not shown at the server application. These problems were experienced with both v1C++ and v2C++ and the restart problem has also been reported in [5]. During some longer duration tests up to an hour, data sent to the server seemed to gradually disappear, like in fig. 4.3. The restart problem could possibly be solved by adding a separate power switch for the dongle, to keep it connected to the gateway when the robot is restarted. Further investigation is needed to find the cause of the gradually disappearing data.

12.2 Future work

12.2.1 Hardware

Upgrading the controller pcb on the robot to a nRF52840 DK solves several issues with the current setup. The nRF52840 DK supports both bluetooth and Thread, which means there is no need for the nRF52840 dongle. The additional GPIO pins makes it possible to utilize the installed quadrature encoders. Another nice feature with the DK is that the header layout is compatible with Arduino Uno Rev. 3 shields.

The rear caster ball should be replaced to reduce the friction load at the rear of the robot.

For a continued use of the ir sensors, an optimization of the software to turn the sensor-tower faster should be looked into.

Alternatives to the ir sensors should be considered to increase the detection range.

12.2.2 Communication

The current communication protocol with the new server should be expanded to include more types of messages. A switch-case to sort incomming messages at the robot is already implemented, which should make further development of the protocol easier.

Experienced communication problems described in section 12.1.6 should be investigated to increase the robustness of the communication between the robot and the server. If the current hardware setup is used further, the limitations of the legacy layer should be inspected closer.

12.2.3 Software

The software running on the robot could use a clean-up and further modularization to increase readability and maintainability.

12.2.4 Navigation

An investigation of the kalman filter is needed to increase the precision of the estimated heading.

The distance controller with the heading correction needs some more tuning to perform well on all floor conditions. An inspection of the motor speed for the same duty cycle is advised to look for the issues found in fig. 11.1 and fig. 11.2.

Bibliography

[1] E. Ese, *Sanntidsprogrammering på samarbeidande mobil-robotar*, Master's thesis. NTNU, Trondheim, 2016.

[2] J. Korsnes, *Development of a Real-Time Embedded Control System for SLAM Robots*, Master's thesis. NTNU, Trondheim, 2018.

[3] E. Leithe, *Embedded nRF52 robot*, Master's thesis. NTNU, Trondheim, 2019.

[4] T. Grindvik, *Creating the foundations of a graphical SLAM application in Modern C++*, Master's thesis. NTNU, Trondheim, 2019.

[5] M. Blom, *nRF52 with OpedThread*, Master's thesis. NTNU, Trondheim, 2020.

[6] M. S. Mullins, *Implementation of Simultaneous Localisation and Mapping in Robotic System using the improved Rao-Blackwellixed Particle Filter*, Master's thesis. NTNU, Trondheim, 2020.

[7] FreeRTOS. Real-time operating system for microcontrollers. [Accessed: 2020-06-16]. [Online]. Available: https://www.freertos.org/Documentation/FreeRTOS_Reference_Manual_V10.0.0.pdf

[8] N. Semiconductor. nrf5 sdk. [Accessed: 2020-06-11]. [Online]. Available: https://www.nordicsemi.com/Software-and-tools/Software/nRF5-SDK

[9] ——. What are sdk 15.x.0 known issues. [Accessed: 2020-06-11]. [Online]. Available: https://devzone.nordicsemi.com/f/nordic-q-a/34155/what-are-sdk-15-x-0-known-issues

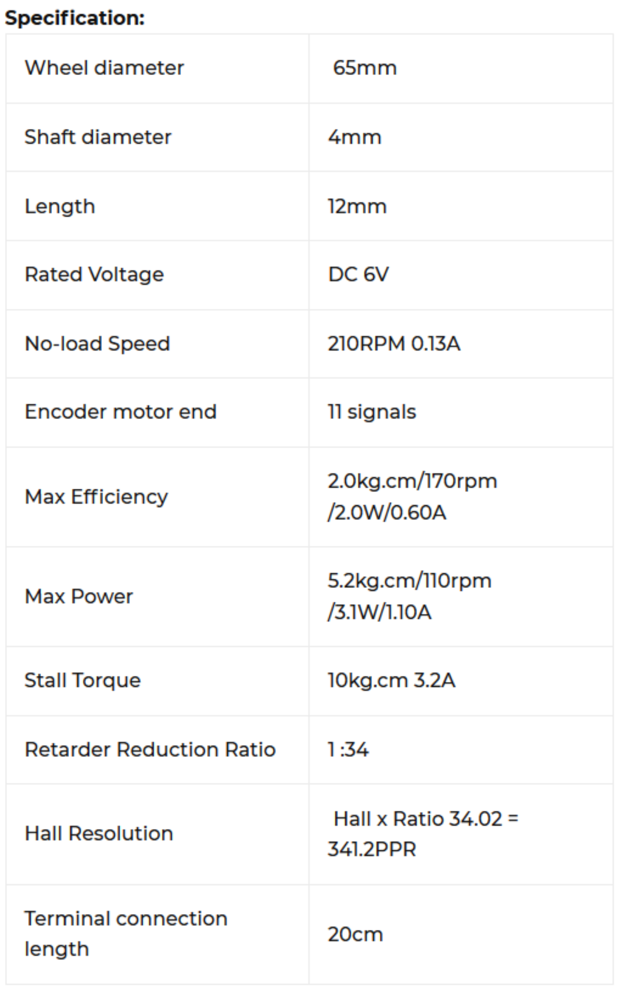
[10] Embedded studio downloads. SEGGER. [Accessed: 2020-01-17]. [Online]. Available: https://www.segger.com/downloads/embedded-studio/

[11] N. Semiconductor. Development kits. [Accessed: 2020-06-11]. [Online]. Available: https://www.nordicsemi.com/Software-and-tools/Development-Kits

[12] chillibasket. (2015) Calibrating & optimising the mpu6050. [Accessed: 2020-03-09]. [Online]. Available: https://wired.chillibasket.com/2015/01/calibrating-mpu6050/

[13] SHARP. Gp2y0a21yk0f. [Accessed: 2020-06-12]. [Online]. Available: https://global.sharp/products/device/lineup/data/pdf/datasheet/gp2y0a21yk_e.pdf

Appendix



Specification:

Wheel diameter	65mm
Shaft diameter	4mm
Length	12mm
Rated Voltage	DC 6V
No-load Speed	210RPM 0.13A
Encoder motor end	11 signals
Max Efficiency	2.0kg.cm/170rpm /2.0W/0.60A
Max Power	5.2kg.cm/110rpm /3.1W/1.10A
Stall Torque	10kg.cm 3.2A
Retarder Reduction Ratio	1 :34
Hall Resolution	Hall x Ratio 34.02 = 341.2PPR
Terminal connection length	20cm

Figure 12.1: Wheel, motor and encoder specifications.

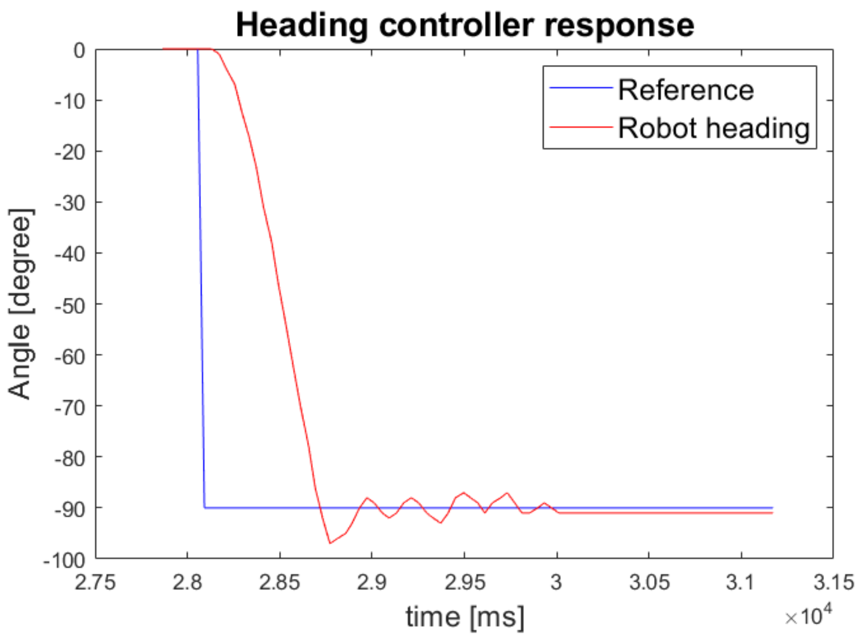


Figure 12.2: Heading controller response for a minus 90 degree reference.

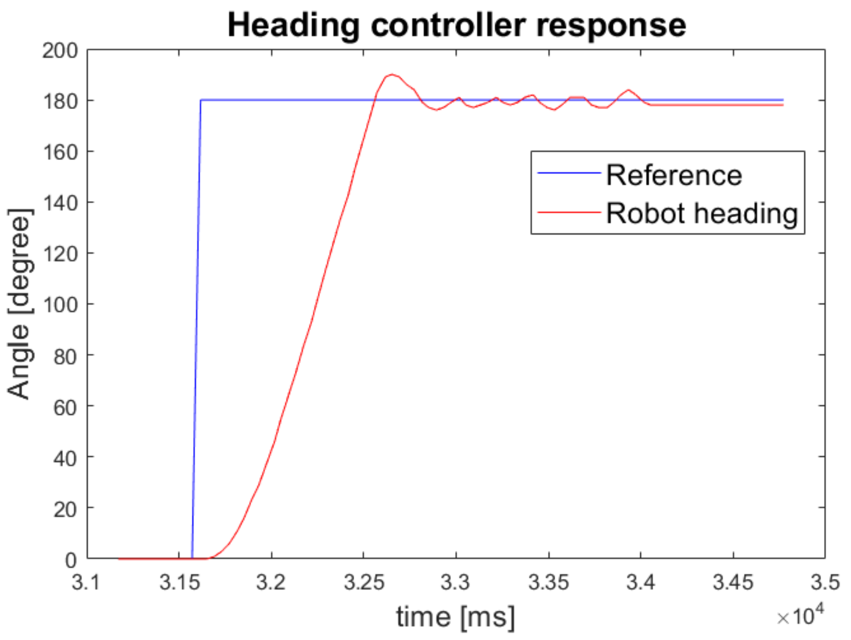


Figure 12.3: Heading controller response for a 180 degree reference.

```
void sendOldPoseMessage(int16_t x, int16_t y, float theta, int8_t
    servoAngle, int16_t* sensorData){
  uint8_t msgLength = 8;
    int8_t data[msgLength];
    int16_t xObject;
    int16_t yObject;

    data[0] = (x & 0xFF);           // lowbyte
    data[1] = (x >> 8);             // highByte
    data[2] = (y & 0xFF);
    data[3] = (y >> 8);

    for(int i = 0; i < NUM_DIST_SENSORS; i++){
        if(sensorData[i] < DETECTION_THRESHOLD_MM){
            xObject = distObjectX(x, theta, servoAngle, sensorData, i);
            yObject = distObjectY(y, theta, servoAngle, sensorData, i);
            data[4] = (xObject & 0xFF);
            data[5] = (xObject >> 8);
            data[6] = (yObject & 0xFF);
            data[7] = (yObject >> 8);
            i2cSendNOADDR(I2C_DEVICE_DONGLE, data, msgLength);
        }
    }
}
```

Listing 12.1: Function for sending data to v1C++.

```
void sendNewPoseMessage(int16_t x, int16_t y, float theta, int8_t
    servoAngle, int16_t* sensorData){
  uint8_t scanMessageID = 2;
  uint8_t msgLength = 24;
  int8_t data[msgLength];
    int16_t xObject;
    int16_t yObject;
  int16_t xDiff = x - lastX;
  int16_t yDiff = y - lastY;
  int16_t thetaDiff = (theta - lastTheta)*RAD2DEG;
  lastX = x;
  lastY = y;
  lastTheta = theta;

    data[23] = 0;

  data[0] = scanMessageID;
    data[1] = (xDiff & 0xFF);         //xLowByte
    data[2] = (xDiff >> 8);           //xHighByte
    data[3] = (yDiff & 0xFF);
    data[4] = (yDiff >> 8);
    data[5] = (thetaDiff & 0xFF);
    data[6] = (thetaDiff >> 8);

    for(int i = 0; i < NUM_DIST_SENSORS; i++){
        xObject = distObjectXlocal(theta, servoAngle, sensorData, i);
        yObject = distObjectYlocal(theta, servoAngle, sensorData, i);
        data[i*4+7] = (xObject & 0xFF);
        data[i*4+8] = (xObject >> 8);
        data[i*4+9] = (yObject & 0xFF);
        data[i*4+10] = (yObject >> 8);

        if(sensorData[i] < DETECTION_THRESHOLD_MM){
            data[23] |= (1 << ((NUM_DIST_SENSORS-i)-1));
        }
        else{
            data[23] &= ~(1 << ((NUM_DIST_SENSORS-i)-1));
        }
    }
  i2cSendNOADDR(I2C_DEVICE_DONGLE, data, msgLength);
}
```

Listing 12.2: Function for sending data to v2C++.

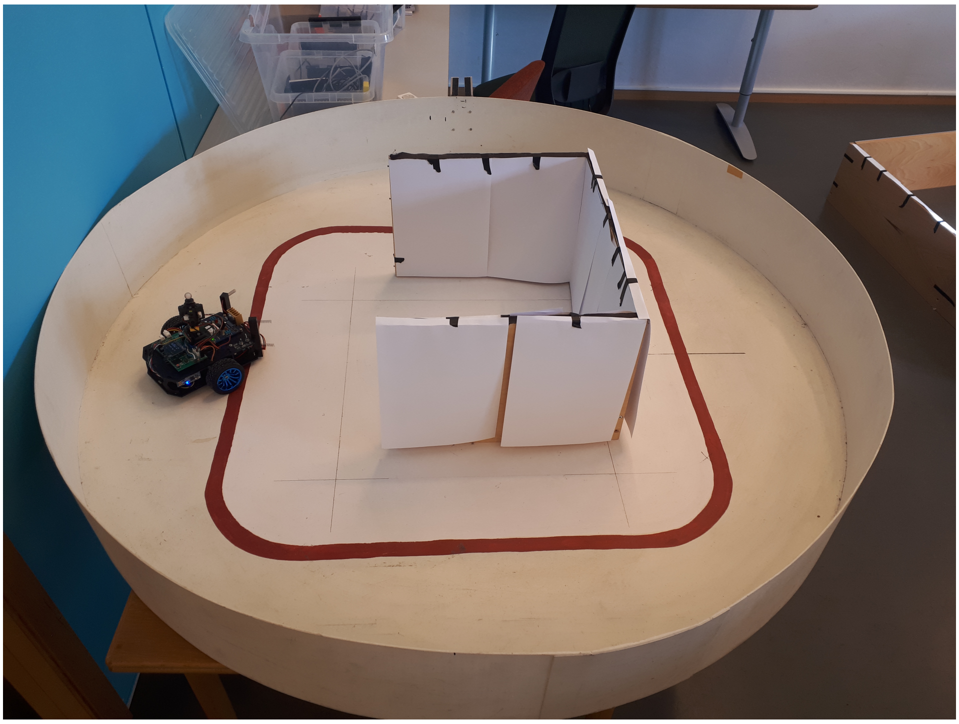


Figure 12.4: Circular track used to test slam.

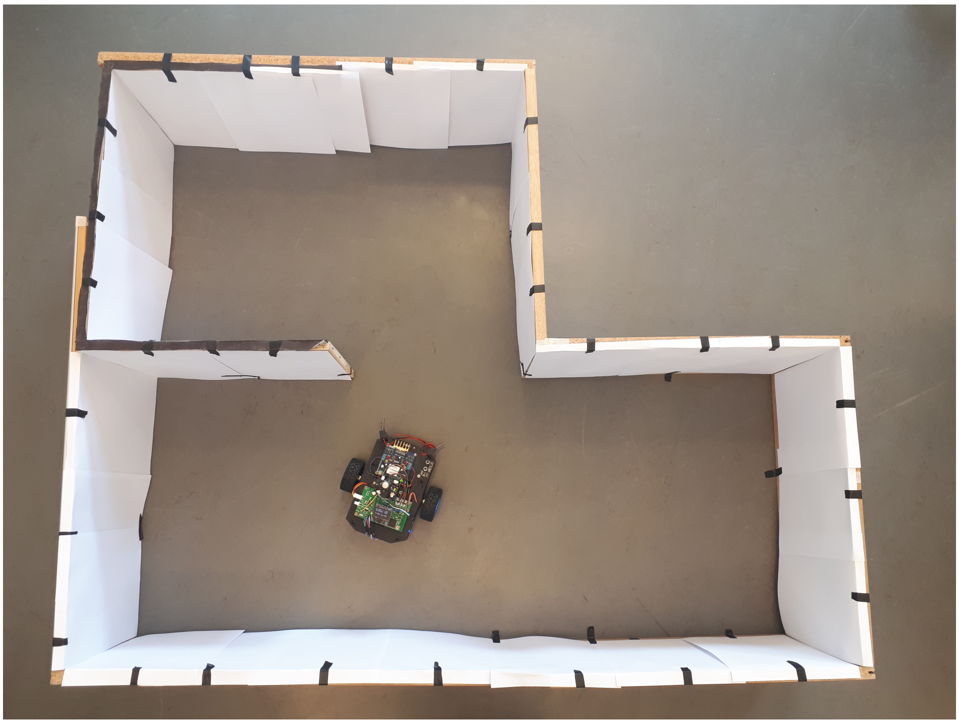


Figure 12.5: Labyrinth track used to test slam.

Arild Stenset

nRF52 robot with OpenThread