

**Master's thesis**

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

Marius Blom

# nRF52 with OpenThread

Master's thesis in Cybernetics and Robotics

Supervisor: Tor Onshus

February 2020



Norwegian University of  
Science and Technology





Norwegian University of  
Science and Technology

Master

# nRF52 with OpenThread

Marius Blom

2020

Master-thesis

Department of Engineering Cybernetics  
Norwegian University of Science and Technology

---

# Problem

The LEGO-robot project have been ongoing since 2004 and have been the subject of several master thesis and projects. The goal with these robots is to map an unknown labyrinth and thus the robots that are available this year have different specifications and algorithms. The robots communicate with a server written in Java that acts like a brain, where the different commands are sent to the robots involved.

This theses will focus on continuing the work of Grindvik[6] with rewriting the current server from Java to C++. This thesis will also continue the work with the NRF robot used by Leithe[9] and get the robot communicate through OpenThread with the new C++ server.

The student will look closer at the following bullet points in this thesis.

- Inspect and analyse the server written in Java
- Inspect server written in C++
- Add navigation algorithms to the server written in C++
- Implement thread with nRF52840 for communication
- Set up a border router to use with future projects/thesis
- Implement new communication with existing NRF robot

---

# Summary

The purpose of this thesis was to continue the work done by Grindvik with rewriting the existing Java server application to C++. This work also include using the NRF robot to communicate with the new C++ application by adding new hardware to the robot, which requires implementing Thread for a local network to use with the robots, setting up a border router to forward information in any direction to an external network and implementing a MQTT-SN gateway for communication with a MQTT broker.

For the NRF robot to be able to communicate with the C++ application, the Tread network was set up by using OpenThread, which is the open-source implementation of Thread. A Raspberry PI was set up to function as a border router and as a MQTT-SN gateway. Since the Raspberry PI lack the IEEE 802.15.4 standard used by Thread, a nRF52840-dongle had to be connected and set up to be used as the network co-processor in the application layer.

The NRF robot already had the software to communicate with peripherals with the I2C protocol but lacked the necessary hardware to communicate via Thread. The communication of the NRF robot was updated from using Bluetooth Smart via the system-on-chip and instead were set up to communicate through the MQTT protocol by using a nRF52840-dongle via I2C.

The C++ application were tested with a real robot by subscribing to topics on an online MQTT broker. The NRF robot would publish measurements made by the infrared sensors to the same topic through the MQTT-SN gateway that is set up on the Raspberry PI and vice versa.

The code in the C++ application have been cleaned up by moving some of the functionalities from the main class and into their own classes. A new function was made in the C++ application, where a real robot will read coordinate pairs published by the C++ application , the coordinate pairs will be relative to the robots current position in the real-world to achieve an implementation of a simple navigation algorithm.

---

# Oppsummering

Formålet med oppgaven var å fortsette med jobben Grindvik hadde gjort når det kommer til å skrive om Java applikasjonen til C++. Denne oppgaven inkluderte også jobben med å bruke en NRF robot for å kommunisere med den nye C++ applikasjonen ved å implementere ny maskinvare til roboten. Dette krever også å implementere Thread for lokalt nettverk for å bruke med roboter, sette opp en grense router for å videresende informasjon i hvilken som helst retning til et eksternt nettverk og implementere en MQTT-SN inngangsport for å kommunisere med en MQTT megler.

NRF krever at det blir satt opp et Thread nettverk ved å bruke den åpne kildekode implementasjonen OpenThread for å kunne kommunisere med C++ applikasjonen. En Raspberry PI ble satt opp som å fungere som en grense router, og som en MQTT-SN inngangsport. Siden Raspberry PI mangler IEEE 802.15.4 standarden som brukes av Thread, så måtte en nRF52840-dongel bli koblet til og brukes som en nettverks ko-prosessor i applikasjonslaget.

NRF roboten hadde allerede programvare for å kommunisere med periferiutstyr gjennom I2C protokollen, men manglet den nødvendige maskinvaren for å kommunisere med Thread. Kommunikasjonen på NRF roboten ble oppdatert fra å bruke Bluetooth Smart gjennom system-on-chip på roboten, i stedet ble det satt opp kommunikasjon gjennom MQTT protokollen ved å bruke en nRF52840-dongel via I2C.

C++ applikasjonen ble testet med en ekte robot ved å abonnere på emner fra en MQTT megler på nett. NRF roboten gjorde målinger med infrarøde sensorer, og publiserte til samme emne gjennom MQTT-SN inngangsporten som var satt opp på Raspberry PI.

Kildekoden i C++ applikasjonen ble ryddet opp i ved å flytte noen av funksjonalitetene fra main klassen over til egne respektive klasser. En ny funksjon ble laget i C++ applikasjonen der en ekte robot vil lese et koordinat par som C++ applikasjonen publiserer. Koordinat parene vil være relative til robotens nåværende posisjon i den ekte verden for å oppnå en implementasjon av en enkel navigasjon algoritme.

---

# Conclusion

The main goal of this thesis was to bring more functionality from the Java application into the C++ application. Whereas the Java application have many intelligent ways to steer a real robot when it comes to mapping and exploration, then the C++ application is more simple in that regard.

The Thread network were already existing in previous thesis, but by not having the necessary hardware and software available to be able to communicate with the C++ application resulted in having to research and redo the process again.

The border router and MQTT-SN gateway can be run on any Linux machine, but since the Thread network were set up with the intention of it being available for future thesis and projects it was decided to purchase and use a Raspberry PI for this instead. and is therefore essential for use with the C++ application. The Thread network have proven to be a robust solution and will self-heal and assign various roles if a unit is disconnected. The Thread network itself will still work without the Raspberry PI, but will lack a way to communicate with the C++ application without the MQTT-SN gateway on the Raspberry PI.

Using a public MQTT broker have been a decent solution and I did not notice any problems with high traffic on the broker. Even if I had success with a public broker, I would still highly consider running it locally for more control.

Using a legacy layer on older robots that lack the necessary hardware to support Thread will ease the process of moving from Bluetooth Smart to Thread. This does however add some issues to the overall project in the long term as the MQTT protocol is currently tied together with the legacy layer and hence any changes to the MQTT protocol will also require changes to the legacy layer. This will add complexity to the project and remove many of the benefits with running OpenThread natively on the robots.

The new navigation algorithm on the C++ application is very simple and will not steer the robot in any intelligent manner. The intention of the algorithm was to have a way for the C++ application to steer a real robot without any manual input from the user. The coordinate pair published by the server will always be in relation to the current position of the real robot. The navigation algorithm have not been thoroughly tested because of issues mentioned in chapter 7.3.1. The C++ application still lack higher level SLAM tasks that is running on the Java application.

---

# Preface

All students at Cybernetics and Robotics at The Norwegian University of Science and Technology are doing a master thesis. The thesis was mostly written during the autumn of 2019 and finished early winter of 2020 and forms the basis of evaluation.

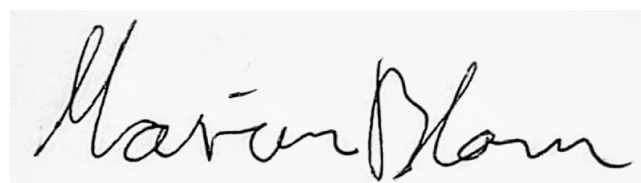
This thesis is a continuation on the LEGO projects that started in 2004 and have been developed through many different projects and master thesis. This thesis have been individual. The purpose of this thesis was to continue the work with creating a C++ server for communication through OpenThread.

At the start of the thesis I was given a desktop computer, the source code for the C++ application from spring 2019, Java application from autumn 2018, legacy\_layer from spring 2019, NRF robot from spring 2019. I was also given the NRF robot itself and a nRF52832 development kit.

Later in the thesis I realised that I was missing crucial hardware and had to order the hardware needed for Thread myself. I got the nRF52840-dongle(s) and Raspberry PI from the Omega Workshop and got the nRF52840 development kit and various cables from an external store. The hardware that was ordered is now part of the LEGO projects.

First of, thanks to my supervisor, Tor Engebret Onshus who were available for discussions whenever I got stuck researching and provided me with a follow up on the thesis throughout the semester.

Also a big thanks to Åsmund Stavadahl from the department of engineering cybernetics at NTNU who provided me with guidance when soldering with a microscope on the new hardware needed in the thesis.



---

Marius Blom

Trondheim, February 10, 2020

---



# TABLE OF CONTENTS

<b>Problem</b>	<b>i</b>
<b>Summary</b>	<b>ii</b>
<b>Oppsummering</b>	<b>iii</b>
<b>Conclusion</b>	<b>iv</b>
<b>Preface</b>	<b>v</b>
<b>Table of Contents</b>	<b>ix</b>
<b>List of Figures</b>	<b>x</b>
<b>Listings</b>	<b>xi</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Motivation . . . . .	1
1.2 How the thesis is presented . . . . .	1
<b>2 Existing system</b>	<b>3</b>
2.1 Starting point . . . . .	3
2.1.1 Robots . . . . .	3
2.1.2 NRF robot . . . . .	3
2.1.3 Java server . . . . .	4
2.1.4 C++ server . . . . .	4
2.1.5 IEEE 802.15.4 . . . . .	4
2.1.6 nRF52832 vs nRF52840 . . . . .	5
2.2 Testing existing Java system . . . . .	6
2.2.1 Mapping a circle with Java-application . . . . .	6
2.2.2 Accuracy tracking . . . . .	7
2.2.3 Driving robot in a straight line . . . . .	8

---

2.3	Running robots with C++ . . . . .	10
<b>3</b>	<b>Java application</b>	<b>11</b>
3.1	Toolchain . . . . .	11
3.1.1	JDK 8 . . . . .	11
3.1.2	Netbeans IDE . . . . .	11
3.2	SSNAR . . . . .	11
3.2.1	Simulation . . . . .	11
3.2.2	SLAM . . . . .	12
3.2.3	Navigation . . . . .	12
3.2.4	Communication . . . . .	12
<b>4</b>	<b>C++ application</b>	<b>14</b>
4.1	Toolchain . . . . .	14
4.1.1	Visual Studio . . . . .	14
4.1.2	vcpkg . . . . .	14
4.2	Third-party libraries . . . . .	14
4.2.1	boost . . . . .	15
4.2.2	sfml . . . . .	16
4.2.3	imgui . . . . .	16
4.2.4	paho-mqtt . . . . .	16
4.2.5	thor . . . . .	16
4.3	Server running . . . . .	17
4.4	Namespace . . . . .	17
4.4.1	NTNU::application . . . . .	18
4.4.2	NTNU::graph . . . . .	18
4.4.3	NTNU::gui . . . . .	19
4.4.4	NTNU::networking . . . . .	21
4.4.5	NTNU::utility . . . . .	21
4.5	Changes and additions to the application . . . . .	22
<b>5</b>	<b>Communication with C++ application</b>	<b>23</b>
5.1	Thread network . . . . .	23
5.1.1	OpenThread . . . . .	23
5.1.2	MQTT . . . . .	23
5.1.3	MQTT-SN . . . . .	24
5.1.4	Raspberry PI as Border Router . . . . .	24
5.1.5	nRF52840 as the dongle for RPi . . . . .	25
5.1.6	Ready for Thread . . . . .	26

---

---

5.2	Thread with existing robots . . . . .	27
5.2.1	Toolchain . . . . .	27
5.3	nRF52840 dongle for robot . . . . .	28
5.3.1	Internal regulated source . . . . .	28
5.3.2	External regulated source . . . . .	28
5.3.3	Legacy layer on the NRF robot . . . . .	29
5.3.4	Thread topology with the nRF robot . . . . .	29
<b>6</b>	<b>C++ application with a real robot</b>	<b>31</b>
6.1	Testing with the NRF robot . . . . .	31
6.1.1	Testing messages sent by the robot . . . . .	31
6.1.2	Testing messages sent by C++ application . . . . .	33
6.1.3	Testing movement and sensor in a controlled space . . . . .	34
6.2	Improving . . . . .	35
6.2.1	New robot event . . . . .	36
6.3	User interface . . . . .	36
6.3.1	New simulation panel . . . . .	36
6.3.2	New manual panel . . . . .	37
6.3.3	Changes to robots panel . . . . .	38
<b>7</b>	<b>Discussion and Further work</b>	<b>43</b>
7.1	Discussion . . . . .	43
7.1.1	MQTT versus BLE communication . . . . .	43
7.1.2	C++ application versus Java application . . . . .	43
7.2	Further work . . . . .	44
7.3	NRF robot . . . . .	44
7.3.1	Right motor . . . . .	44
7.3.2	Trackball . . . . .	44
7.3.3	Anti-collision . . . . .	44
7.3.4	Robot initialization . . . . .	44
7.3.5	Gear on the wheels . . . . .	44
7.3.6	Upgrade SoC . . . . .	45
7.4	C++ application . . . . .	45
7.4.1	SLAM . . . . .	45
7.4.2	navigate_square . . . . .	45
7.4.3	Robot status . . . . .	45
7.4.4	Better support for multiple robots . . . . .	46
7.5	General suggestions to the project . . . . .	46

---

7.5.1	Run robot with nRF52840 DK . . . . .	46
7.5.2	Local MQTT Broker . . . . .	46
	<b>Bibliography</b>	<b>46</b>
	<b>Appendix</b>	<b>47</b>
.1	Files overview . . . . .	47

# LIST OF FIGURES

2.1	Layers of the OSI model . . . . .	5
2.2	Mapping a circle . . . . .	6
2.3	Matlab plot of NRF . . . . .	7
2.4	Robot reach destination . . . . .	8
2.5	Robot does not reach destination . . . . .	9
4.1	Server running . . . . .	17
5.1	MQTT-SN gateway illustration . . . . .	24
5.2	Raspberry Pi 3 Model B+ . . . . .	25
5.3	Raspberry Pi 3 Model B+ with NCP dongle . . . . .	26
5.4	SB2 cut, SB1 soldered . . . . .	28
5.5	Thread topology . . . . .	30
6.1	charging station . . . . .	32
6.2	charging station2 . . . . .	33
6.3	drive office01 . . . . .	34
6.4	drive circle . . . . .	35
6.5	gui simulation . . . . .	37
6.6	manual . . . . .	38
6.7	gui robot . . . . .	39
6.8	navigate off . . . . .	40
6.9	navigate on . . . . .	41

# LISTINGS

4.1	install vckpg . . . . .	14
4.2	install boost . . . . .	15
4.3	install sfml . . . . .	16
4.4	install imgui . . . . .	16
4.5	install paho-mqtt . . . . .	16
4.6	install thor . . . . .	16
5.1	Wireless Personal Area Network Status . . . . .	26
5.2	Legacy Layer main.c . . . . .	29
6.1	ROBOT_IDLE callback . . . . .	36
6.2	navigate_square . . . . .	41

# 1 INTRODUCTION

This thesis is a continuation of the work done by Torstein Grindvik in his master thesis containing the use of a C++ application and the use of Thread communication instead of the Bluetooth Low Energy used by the Java application.

## 1.1 Motivation

Robotics are often used in both simple and advanced repetitive tasks where autonomy is one of the greatest achievements in robotics. This could take all kinds of form, from cutting grass, vacuuming the home of people, to large robotic arms in an assembly line.

To move forward, we often have to take a step back and see where we are. Instead of trying to invent the wheel, we can look at what technology already exist around us and adapt them to our needs. A simple block of LEGO with an infrared sensor attached can be used to measure distance, combine this with an electrical motor and some wheels and we have a robot that can detect obstacles.

The collaboration between robots is an essential challenge in robotics. The ability for cooperation between robots make the sum of all parts larger than the parts themselves and open new possibilities. By combining simple existing materials and sensor with software are we able to achieve an autonomous system that can do the repetitive task we want it to perform.

We are only limited by our own imagination, the future is already here.

## 1.2 How the thesis is presented

The contents of the thesis

- Chapter 2 contain information on how the system was at the beginning of the thesis
- Chapter 3 contains some more information as to how the current Java system is working
- Chapter 4 contains information about how the C++ system is working

- 
- Chapter 5 contains information about how the communication with the C++ system is working and how to set up what is needed for it to run.
  - Chapter 6 contains information as to how the C++ system and the Thread network work together with a real robot
  - Chapter 7 contains discussion and suggestion on how to improve the system and robot.



# 2 EXISTING SYSTEM

The LEGO project have been around at NTNU since 2004 as an attempt at mapping a labyrinth using simple existing technology. Today there exist several different types of robots, the NXT robot, AVR robot, EV3 robot, several Arduino robots and the NRF robot.

## 2.1 Starting point

### 2.1.1 Robots

There exist several different robots with different hardware and algorithms. The main purpose of these robots are mainly the same even if they are built with different specifications. The main goal of the robots is to receive coordinates to move towards from the Java server, then detect objects, navigate around them and communicate this information back to the Java server, where the Java server draw a map with the information sent by the robots.

### 2.1.2 NRF robot

The NRF robot is equipped with infrared (IR) sensors that emit an infrared light that detects if the light gets reflected back to the sensor. The wheel rotation is measured with a magnetic rotary encoder. This use magnetic force and sensor to decode rotation into an electrical signal. The communication is done wireless with Bluetooth Smart. A new printed circuit board (PCB) was developed in 2018 by Korsnes[8]. This new PCB use the nRF52832 System-on-Chip (SoC) which comes with an increase in RAM and program storage capacities compared to the older ATmega2560 and was improved with new functionality as an on-board display and a microSD slot for portable storage. There was a continuation of this work in 2019 by Leithe[9] where many of the sensors used on the robot were calibrated. A Kalman filter dedicated to estimating the robot position was implemented.

---

### **2.1.3 Java server**

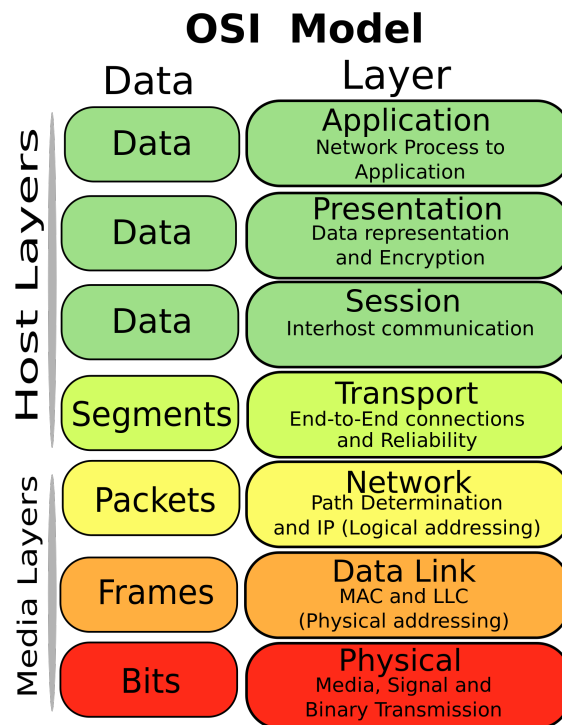
Most of the functionality from an existing MATLAB application was ported over to Java in 2016.[12] It was also created a graphical application with an user interface. The communication went from using Bluetooth to Bluetooth Low Energy (BLE, which is also known as Bluetooth Smart), with the Nordic Semiconductor nRF51 USB dongle.[13] The Java application was later revised to use Cartesian coordinates instead of the relative polar coordinates ported over from MATLAB.

### **2.1.4 C++ server**

There have been expressed a wish to use a more familiar language more suited to the background of the students which tend to work on the SLAM project and Grindvik began to port over the functionality from the Java application to C++ in 2019.[6] This included creating a graphical application for drawing measurements sent by the robots, creating a user interface in order to let users issue control over various tasks and implementing a future-proof communication method with the robots. As mentioned in section 2.1.3, the communication stack used by the Java application is built upon BLE. This implementation was entirely discarded in the C++ application. The new stack use Thread networking technology where this Thread networking technology utilize the IEEE 802.15.4 standard which is not supported by the hardware on any of the existing robots, including the NRF robot mentioned in section 2.1.2. This have been mitigated to some degree by a creation of a legacy layer by Grindvik[6], this layer is more a band-aid fix then a solution to the problem. The C++ application is according to Grindvik more flexible in development and have more useful basic features, but it does not yet have higher level SLAM features.

### **2.1.5 IEEE 802.15.4**

IEEE 802.15.4 is a standard which has specifications on the data link- and physical layers of the OSI model.



[1]

**Figure 2.1:** Layers of the OSI model

### Physical layer

Data transmission occurs on this layer. Different channels on the chosen frequency are managed here. Signal strength and energy usage and monitoring also happens here.

### Data link layer

MAC frames are sent here. The layer acts as a higher level manager of the physical layer, controlling access. Beacons are also performed here, where beacons are sent and received in order to exchange information about the network.

### 2.1.6 nRF52832 vs nRF52840

The nRF52832 SoC that is on the NRF robot mentioned in section 2.1.2 does not support the IEEE 802.15.4 standard. There is however a new lineup of SoCs which support the IEEE 802.15.4 standard, the nRF52840. This is the newest SoC in the same family as the ones already in use for their Bluetooth technology in the robots for the SLAM project.

---

## 2.2 Testing existing Java system

Getting the server and robot up were an important step to see how the system works and behaves at the start of the thesis.

### 2.2.1 Mapping a circle with Java-application

The Java-application automatically detected the NRF robot with pre-existing code. The robot could be connected to the existing Java-application through the Bluetooth on the nRF52832 system-on-chip. The robot moved strange and the mapping progress were very slow, the robot would often come to a complete stop and would not continue to navigate with the commands sent from the Java-application. I ran a short test with the NRF-robot inside a circle-track found in the office to see how the robot would behave in a more closed environment.



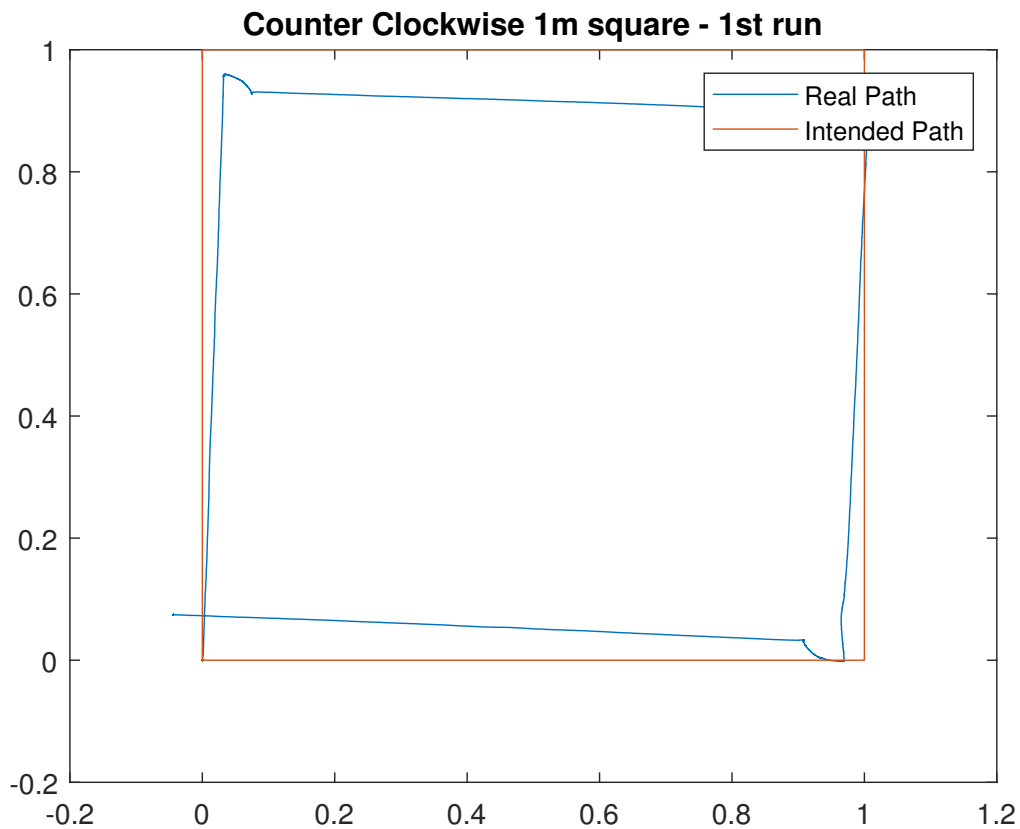
**Figure 2.2:** Mapping a circle

We can see an incomplete map of the circle in fig 2.2 as seen from the Java-application. The only solution to get the robot to move after it had stopped was to command the robot to move to coordinates sent manually from the Java-application.

---

### 2.2.2 Accuracy tracking

The NRF-robot was then equipped with 5 IR-reflectors to use with the tracking of the exact movement with OptiTrack in B333 "Slangelabben". It was quickly noticed that the robot had problems to just drive in a straight line in B333 due to a more slippery surface. The issue became even more apparent when the robot had to rotate in order to change heading. There seem to be a very heavy weight on the trackball that is used to help the rotation of the robot. The weight of the back of the robot caused to the friction of the trackball to be large enough for the trackball to stop rotating and therefore causing the spinning rubber wheels in the front of the robot to increase on the surface of the B333. This issue was mitigated slightly by cleaning the trackball. This resulted in reduced spinning of the wheels while tracking the robot with OptiTrack.



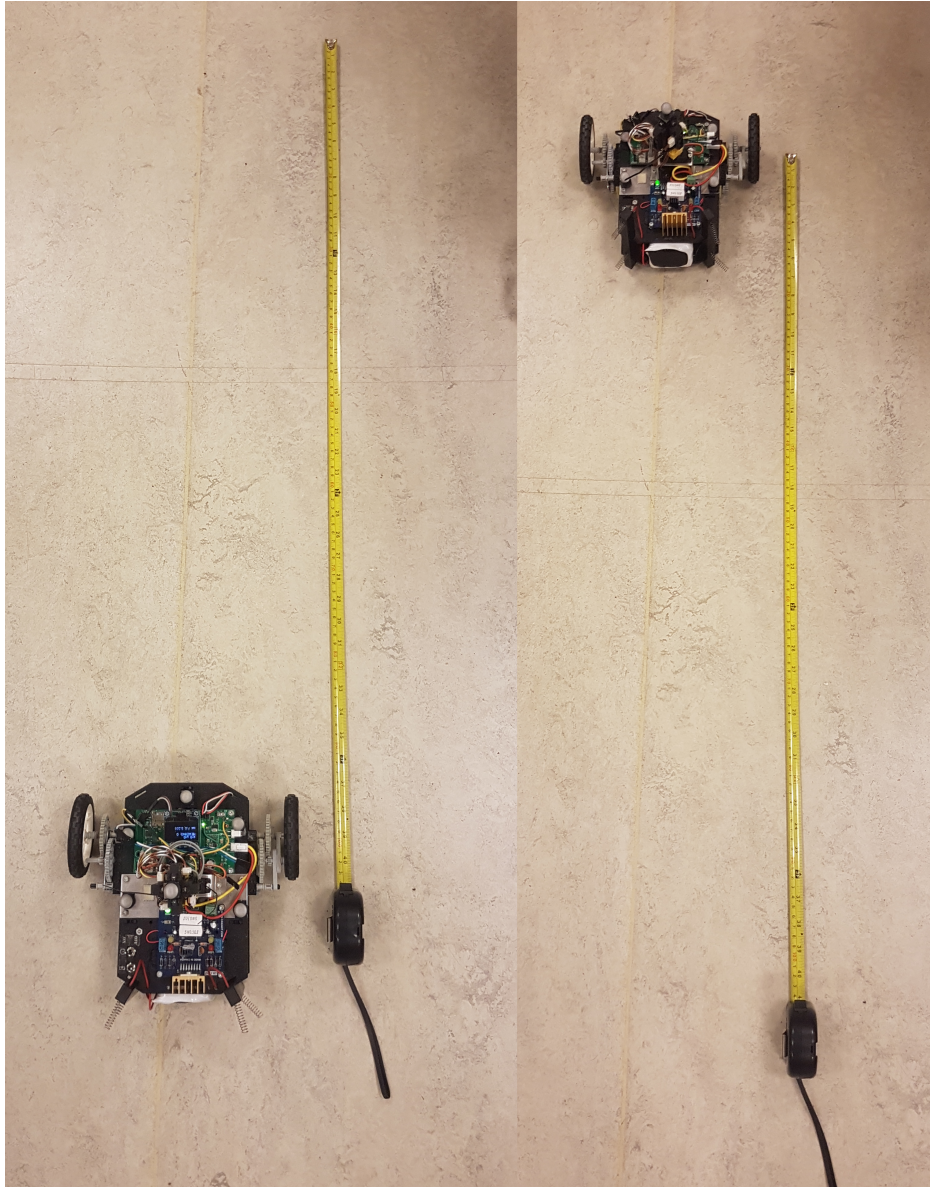
**Figure 2.3:** Matlab plot of NRF

The data from the OptiTrack system was then plotted in Matlab, figure 2.3, and it can be seen that the robot does not reach the destination when trying to manually drive the robot in a 1x1 meter square. There were done several attempts at tracking the robot, where this was the best result.

---

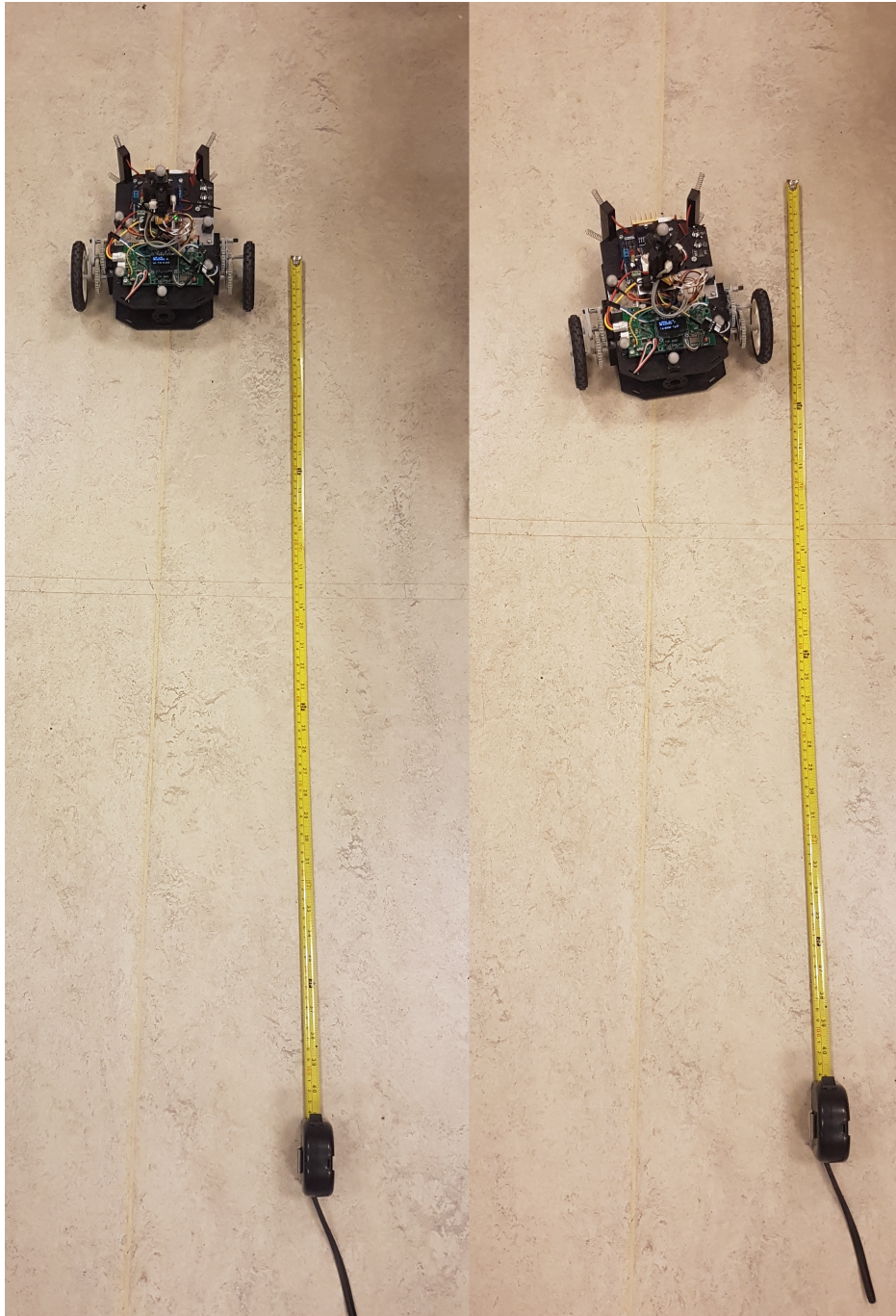
### 2.2.3 Driving robot in a straight line

As mentioned in section 2.2.2, it was noted that the robot did not fully reach the destination. It was then decided to see if the Java-application and the NRF-robot had the same measurements. It was measured a 1 meter straight line in the office to see check if the robot were driving the distance that was specified by the Java-application.



**Figure 2.4:** Robot reach destination

The robot managed to reach the target coordinates set by the Java-application as seen in figure 2.4. There were done several tests in the office with the same results. However there was some outliers where the robot were nowhere close to reaching the target coordinates.



**Figure 2.5:** Robot does not reach destination

We can see from figure 2.5 that the robot did not always reach the target coordinates. This was caused by high friction on the trackball on the back of the robot, as in this case it acted like a brake and caused the rubber wheels on the robot to spin and set the robot of course. The robot came to a stop when it thought it had reached the target coordinates, yet it is clear that this was not the case.

---

## 2.3 Running robots with C++

It was not possible to run any of the robots with the C++ application that Grindvik[6] had started on. None of the robots are compatible with the C++ application without adding new hardware and there was not a running Thread network for communication. In practice this means that the real robots had no way to communicate with the C++ application.



# 3 JAVA APPLICATION

The current robots use a Java application to navigate and draw a map of the surroundings. The communication protocol with the Java application is Bluetooth Smart(BLE).

## 3.1 Toolchain

### 3.1.1 JDK 8

The Java application use JavaFX for the graphical user interface and the latest version of Java (JDK12) does not come with the JavaFX library set. The Java development kit 8(JDK8) does however include JavaFX and I found it easier to use the JDK8 instead of adding the JavaFX library set manually.

### 3.1.2 Netbeans IDE

Netbeans integrated development environment was chosen because of familiarity and that it can be used with JDK8.

## 3.2 SSNAR

System of Self-Navigation Robots(SSNAR) is the Java server application used with the thesis. The Java application have been the main server with the current Bluetooth robots for many years. There seem to be several different versions of this server.

### 3.2.1 Simulation

The simulation of the robots in the Java application was developed by Thon in 2016. The simulator is able to simulate robots. This simulator have later been improved upon and is working as intended with simulating a real robot.

---

### 3.2.2 SLAM

Simultaneous localization and mapping

### 3.2.3 Navigation

The A\* algorithm were used for path-planning on the Java application, where the A\* algorithm is a search-algorithm that utilize the principle of best-first. The A\* algorithm is similar to dijkstra algorithm where it use an open set and a closed set where new nodes are inserted into the open set. The algorithm computes a cost for each node and at each iteration it selects the node with the lowest cost from the open-set. The algorithm were implemented by Thon so that it can traverse both directly and diagonally in the map. The current Java application use cartesian coordinates.

### 3.2.4 Communication

The current BLE communication use a nRF51-server dongle for communication with the server and a nRF51-peripheral dongle on the robots. Some of the current robots use a nRF52832-dongle for BLE instead, which serve the same purpose as the nRF51-dongle. The message protocol can be summarised by Table 3.1 and Table 3.2.

Robot	
Message type	Parameters
Handshake	Message type Robot width and length Tower offset Axle offset Sensor offset Initial sensor heading
Update	Message type Robot position (x,y) Orientation Tower heading Sensor values
Status	Message type Idle

**Table 3.1:** Message list from the robot

---

Server	
Message type	Parameters
Handshake	Message type Orientation Distance
Update	Message type Handshake confirmed Pause robot Unpause robot Robot finished

**Table 3.2:** Message list from the server

# 4 C++ APPLICATION

Several tools and various hardware need to be installed and configured to be able to start working with the new C++ application.

## 4.1 Toolchain

There are many different tools and software available for developing the new C++ application.

### 4.1.1 Visual Studio

Some of the required libraries for the C++17 application does not work as intended with 64-bit. The C++ application is therefore built in a 32-bit environment. Visual Studio was chosen as a development tool because of familiarity with the environment.

### 4.1.2 vcpkg

vcpkg is a command-line package manager that simplifies the acquisition and installation of third-party libraries. The libraries in the vcpkg Windows catalog have been tested for compatibility with Visual Studio 2015/2017/2019.

**Listing 4.1:** install vcpkg

```
1 git clone https://github.com/microsoft/vcpkg.git
2 cd vcpkg
3 bootstrap-vcpkg.bat
```

## 4.2 Third-party libraries

Various third-party libraries have been used and are mandatory for the C++ application to run.

---

## 4.2.1 boost

Boost provides a open-source and peer-reviewed portable C++ source libraries. Boost works on almost any modern operating system, like UNIX and Windows variants.[2]

**Listing 4.2:** install boost

```
1 .\vcpkg install boost
```

### boost Graph

Graphs are mathematical abstractions that are useful for solving many types of problems in computer science. Part of the Boost Graph Library is a generic interface that allows access to a graph's structure, but hides the details of the implementation. The Boost Graph Library algorithms consist of a core set of algorithm patterns and a larger set of graph algorithms. The core algorithm patterns are breadth first search, depth first search and uniform cost search.[3] The graph algorithms in the Boost Graph Library currently include

- Dijkstra's Shortest Paths
- Bellman-Ford Shortest Paths
- Johnson's All-Pairs Shortest Paths
- Kruskal's Minimum Spanning Tree
- Prim's Minimum Spanning Tree
- Connected Components
- Strongly Connected Components
- Dynamic Connected Components (using Disjoint Sets)
- Topological Sort
- Transpose
- Reverse Cuthill Mckee Ordering
- Smallest Last Vertex Ordering
- Sequential Vertex Coloring

---

## boost Fiber

Fiber provides a framework for micro-/userland-threads (fibers) scheduled cooperatively. Each fiber has its own stack.

### 4.2.2 sfml

Simple and fast multimedia library (sfml) provides a sample interface to various components of the PC.[14] A multi-platform multimedia library composed of five modules: system, window, graphics, audio and network.[14]

**Listing 4.3:** install sfml

```
1 .\vcpkg install sfml
```

### 4.2.3 imgui

Interface framework for graphical applications.[4] Well suited for creating user interfaces.

**Listing 4.4:** install imgui

```
1 .\vcpkg install imgui
```

### 4.2.4 paho-mqtt

A library which enables C++11 applications to connect to a MQTT broker, publish messages to the broker, and to subscribe to topics and receive published messages.[5] Paho-mqtt did not integrate well into the C++ application and had to be added manually in Visual Studio.

**Listing 4.5:** install paho-mqtt

```
1 .\vcpkg install paho-mqtt
```

### 4.2.5 thor

Multi-platform, open source C++ library which provides several extensions to SFML with higher-level features.[7]

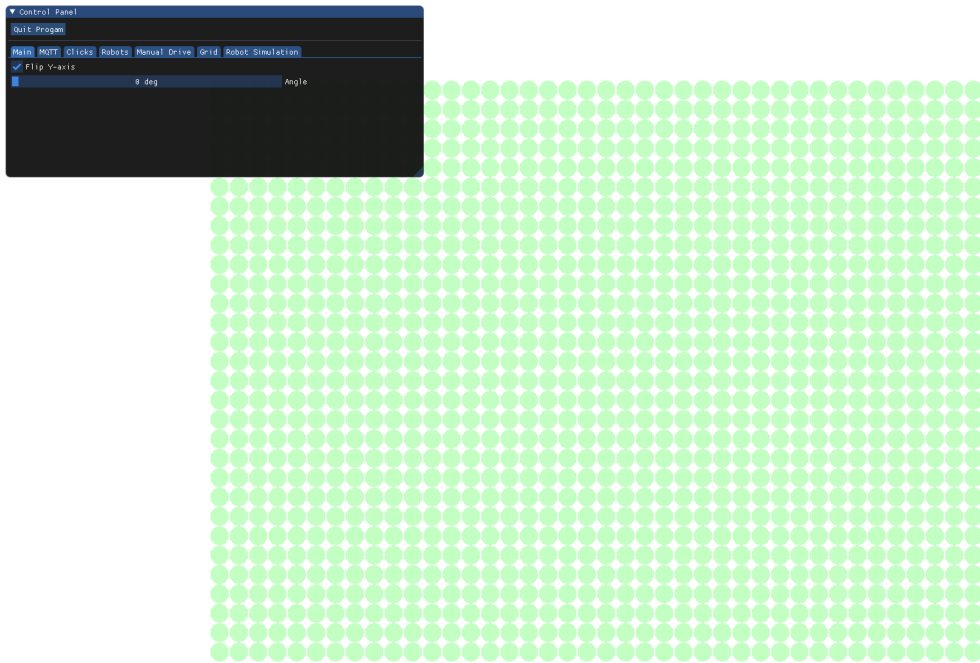
**Listing 4.6:** install thor

```
1 .\vcpkg install thor
```

---

## 4.3 Server running

After the mandatory libraries were installed, using x86 and using C++17 it was possible to get the C++ application to properly build.



**Figure 4.1:** Server running

A graphical overview of the running C++ application with nothing connected can be seen from figure 4.1.

## 4.4 Namespace

The previous authors of the Java application have separated their contributions by prefixing their classes and functions by short keywords. Some of these have been combined and made into namespace in the C++ application. The prefix TG used by Grindvik have been changed to the more generic name NTNU and some new namespace have been added where it was necessary. The new code have been merged into existing functions and classes instead of separating the contribution like it was done in the Java application to hopefully make the code easier to work with in the future. Most of this is unchanged and can also be found in chapter 7.2 and 7.3 in Grindvik thesis[6].

---

### 4.4.1 NTNU::application

For uses directly concerned with the actual application.

#### SLAM::message

Messages from robots. Messages include robot position and obstacles.

#### SLAM::robot

Abstraction of robots. They are drawable and intended to be shown graphically.

#### SLAM::robots

Messages are fed to this robots class, where they are parsed. A new robot might then be found at any time, and this class adds robots dynamically. A robot is defined by the topic to which it publishes messages to. There are some events put out by this class. These include `ROBOT_MOVED`, `ROBOT_IDLE`, `ROBOT_FOUND_OBSTACLE` and `ROBOT_CLEARED_POINTS`. These can be assigned as callbacks.

#### SLAM::robot\_simulation

This class simulates a robot by publishing messages. The simulated robot move around the map by randomising small changes of position and randomising obstacles to publish a mqtt message to `v1/robot/simulated/adv`. It is possible to simulated the path-planning with the simulated robot but there is however not currently possible to have the simulated robot move to a specific coordinate.

#### SLAM::grid

High-level grid used in the application. It combines `NTNU::graph::grid::obstructable_grid` and `NTNU::gui::elements::circle_grid`. The result is a grid which can be shown and represented graphically, and also be obstructed and navigated via pathfinding.

#### SLAM::utility

Collection of various free utility functions. Functions include converting to and from the global coordinate system and the rows and columns found in the application grid drawn in the application, getting random numbers, random colours and converting from bytes to `int16_t`.

### 4.4.2 NTNU::graph

Deals with graphs, in the sense of nodes, vertices and algorithms using graph constructs



---

## **base::grid**

Base grid abstraction in the context of graphs. Can be constructed from a given number of rows and columns.

## **grid::obstructable\_grid**

Inherits from base grid to allow to mark nodes as obstructed. It is possible to query whether a node is obstructed.

## **grid::utility**

Contains a utility function for converting integer row and columns accesses into vertex descriptors as used internally by Boost.graph library

## **grid::vertex\_hash**

Boost.graph requires a hashing function for placing vertices into an unordered map, which is done as part of the pathfinding algorithm.

## **pathfinding**

Have two functions. First function is solve, which takes a grid graph and two points in a grid and return a vector of points which represents the points in the solved path between the two points given. The next function is reduce, which takes a vector of points and reduce it to lines.

### **4.4.3 NTNU::gui**

Everything related to graphics is sorted here, including drawing of all primitives and all handling of panels.

## **base::path**

This class is graphically drawable. The path is composed of lines (via NTNU::gui::collections::lines) which connects waypoints which are circles (via NTNU::gui::collections::circles). It's intention is to graphically show a path.

## **base::updatable**

Header-only class. It is a simple abstract class with a single virtual function that must be defined in child classes. This function is given a time delta since the last time it was called. By inheriting from this interface, any class can add itself as an updatable via the NTNU::gui::base::window class.

---

### **base::window**

Provides the actual window in which all the graphics are shown, including primitives and panels. It allows the user to add panels, drawables, and updatables to the window via polymorphism. If any of these are added via a single function call, the given element will be shown as part of the window.

### **collections::circles**

This class is a helper class wrapping a collection of the basic drawable circle. It has some functionality for acting on the whole collection, such as setting colours and alpha values, which apply to the whole collection. Attributes such as radius and point resolution of circles are settable.

### **collections::lines**

Wraps a collection of lines, similar to `NTNU::gui::collections::circles`.

### **elements::circle\_grid**

This class represents a drawable grid of circles. It allows creating a grid with the amount of rows and columns wanted, and a given separation. Separation indicates the space between rows and columns, essentially being the diameter of a given circle in the grid. The class can also have customisable colouring and alpha values.

### **panel::clicks**

Control what mouse clicks do during the runtime of the application. It generate events which are generated when the left-click or right-click mode changes.

### **panel::panel**

Acts as a parent class for all panels, allowing them to be grouped together via polymorphism. If the panel class is constructed with a title, it will become a stand-alone panel with it's own window. If no title is given, it is expected to be embedded within another panel.

### **panel::control\_panel**

Holds other panels. Main feature of this class is that when a panel is added to it, the child panel is automatically placed within a new tab in the control panel.

---

### **panel::simulation\_panel**

Panel used for simulation of a robot by running the task from `NTNU::application::SLAM::robot_simulation`

### **panel::target\_panel**

Panel that is used to manually drive the robot, either to a certain coordinate or a set of pre-determined coordinates that form a square.

### **panel::MQTT\_panel**

Panel used as a helper for the MQTT communication in the application. It allows manually publishing messages to arbitrary topics, manually subscribing to topics and monitoring the flow of in- and outbound messages.

## **4.4.4 NTNU::networking**

All protocols and their usages are sorted under this namespace.

### **protocols::MQTT::MQTT**

Implements all the MQTT functionality. The class wraps an asynchronous MQTT C library. It exposes a number of events which can be used with callbacks: `connect_response`, `disconnect_response`, `publish_response`, `connection_lost`, `message_arrived` and `delivery_complete`. An important note of this class is that the underlying C library uses threads to achieve asynchronicity. All callbacks are given in a multi-threaded context, which means special care has to be taken when using these events.

### **protocols::MQTT::utility**

Introduce a structure representing an MQTT topic. This application uses topics which are composed of a version, a sender, an ID and a command. This structure is arbitrary and need not be used in the future if another scheme is desired.

## **4.4.5 NTNU::utility**

Utility functions and classes of higher abstractions-not sorting under any other namespace-sort here

### **callbacks**

Enabling class which provides a simple shortcut to allow any class to use the pattern of event-callback.

---

## 4.5 Changes and additions to the application

NTNU::application	SLAM::robot SLAM::robots	modified modified
NTNU::gui	panel::target_panel panel::simulation_panel	new new
NTNU::graph		unchanged
NTNU::networking		unchanged
NTNU::utility		unchanged

**Table 4.1:** Changes to namespace

Table 4.1 shows where changes were made and which namespace that were added to the C++ application.

# 5 COMMUNICATION WITH C++

## APPLICATION

Grindvik did a case study on an alternative method of communication in chapter 5 of his thesis. [6] The newer C++ server is using Thread for communication instead of BLE. This mesh network requires new hardware and software that were not available and had to be set up again from scratch.

### 5.1 Thread network

Thread is a low-power wireless mesh networking protocol based on Internet Protocol (IP). Thread enables device-to-device and device-to-cloud communications. The network has no single point of failure and can self-heal and reconfigure when a device is added or removed which means that devices in a Thread network will automatically change roles if a device disconnects from the network.

#### 5.1.1 OpenThread

OpenThread from Google is an open-source implementation of Thread.

#### 5.1.2 MQTT

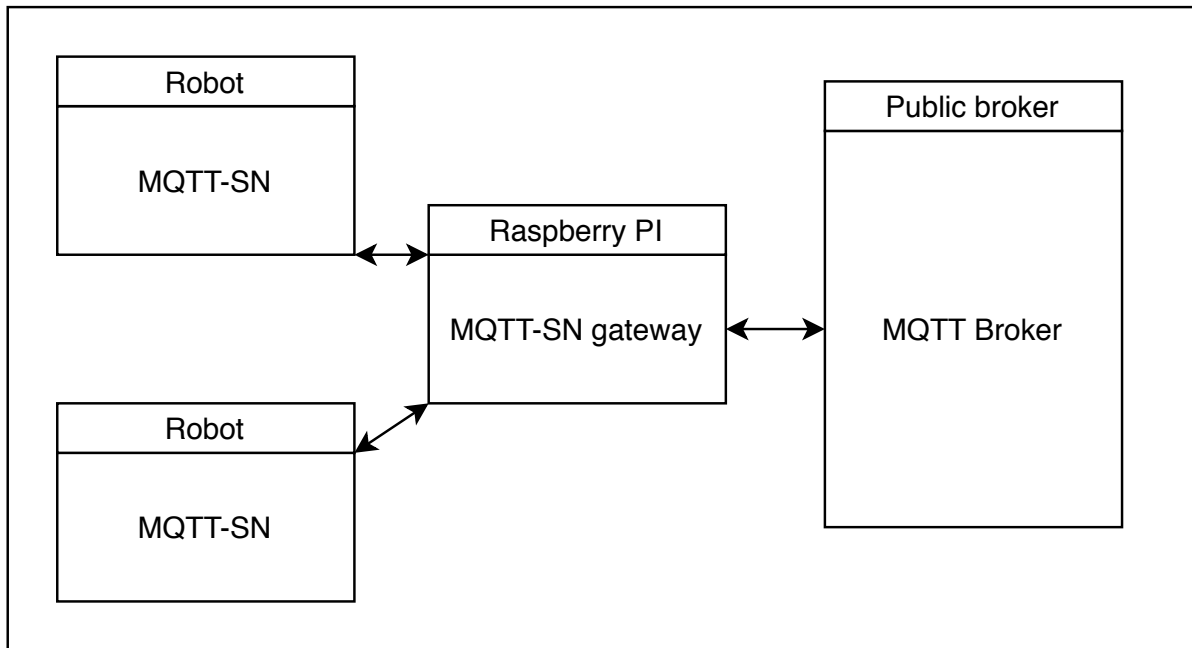
MQTT is a machine-to-machine messaging transport protocol. It is a publish/subscribe messaging protocol, which means that it will let clients subscribe to topics and then receive messages if other clients publish to such a topic.[10] The protocol usually runs over TCP/IP. MQTT requires a broker, which is responsible for receiving all messages, filtering the messages, determining who is subscribed and sending the messages to the subscribed clients.

An example of a topic which is used in the C++ application is `v1/robot/name_of_robot/adv`, where the robots publish their current position and coordinates to detected obstacles in a coordinate system.

---

### 5.1.3 MQTT-SN

MQTT-SN(MQTT for Sensor Networks) is designed to mostly work the same way as MQTT. It also use a publish/subscribe model and can be considered as a version of MQTT. The main difference is to reduce the size of the message payload and to remove a need for a permanent connection by using UDP instead of TCP.



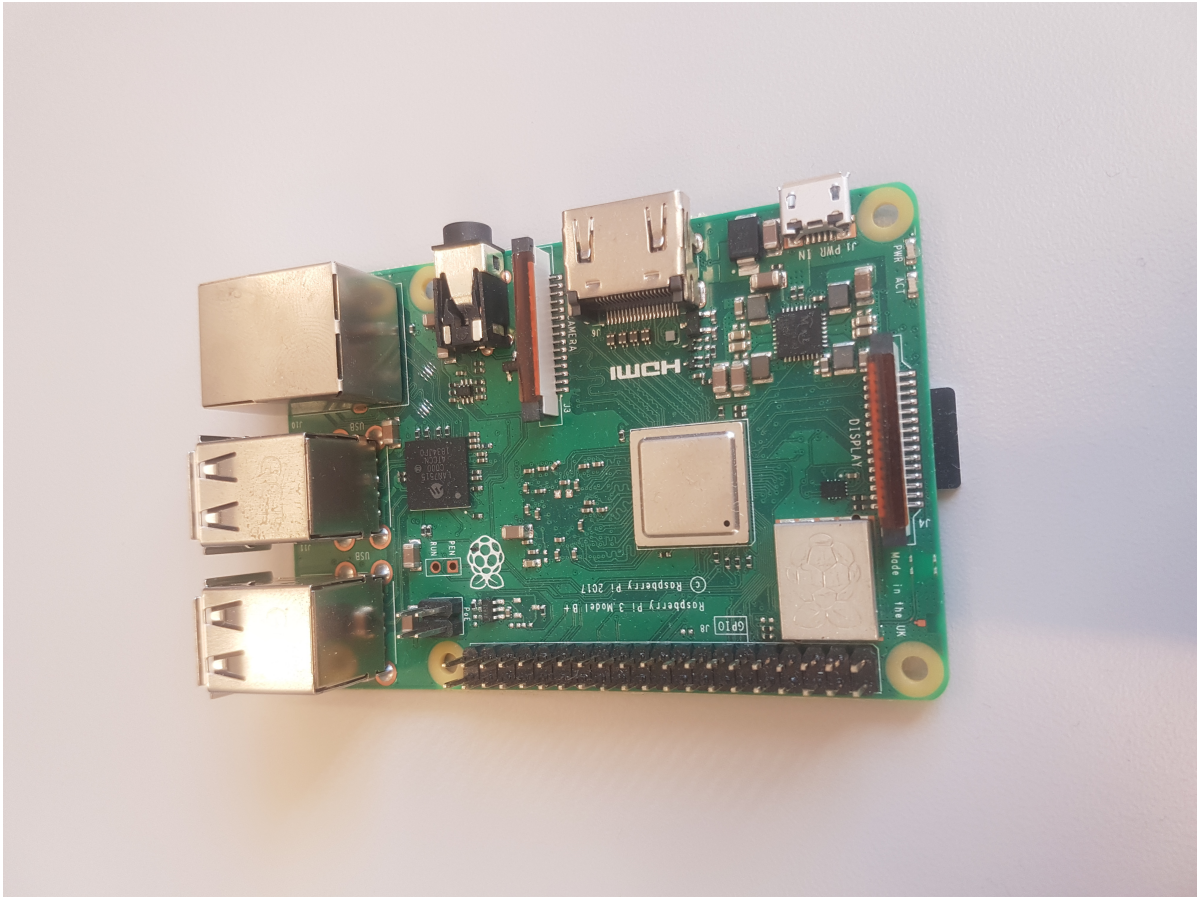
**Figure 5.1:** MQTT-SN gateway illustration

As seen from the illustration in figure 5.1, a MQTT-SN gateway is required between the MQTT-SN client and the MQTT broker.

### 5.1.4 Raspberry PI as Border Router

The C++ application use the MQTT messaging protocol for communication. A border Router and a MQTT-SN gateway was required for the robots to communicate with the C++ application. A border Router is a device that can forward information between a Thread network and a non-Thread network. The OpenThread border router software supports any Linux machine.

A Raspberry Pi 3 Model B+ was set up with an image of a linux installation from Nordic Semiconductor which include the OpenThread border router and a MQTT-SN gateway.



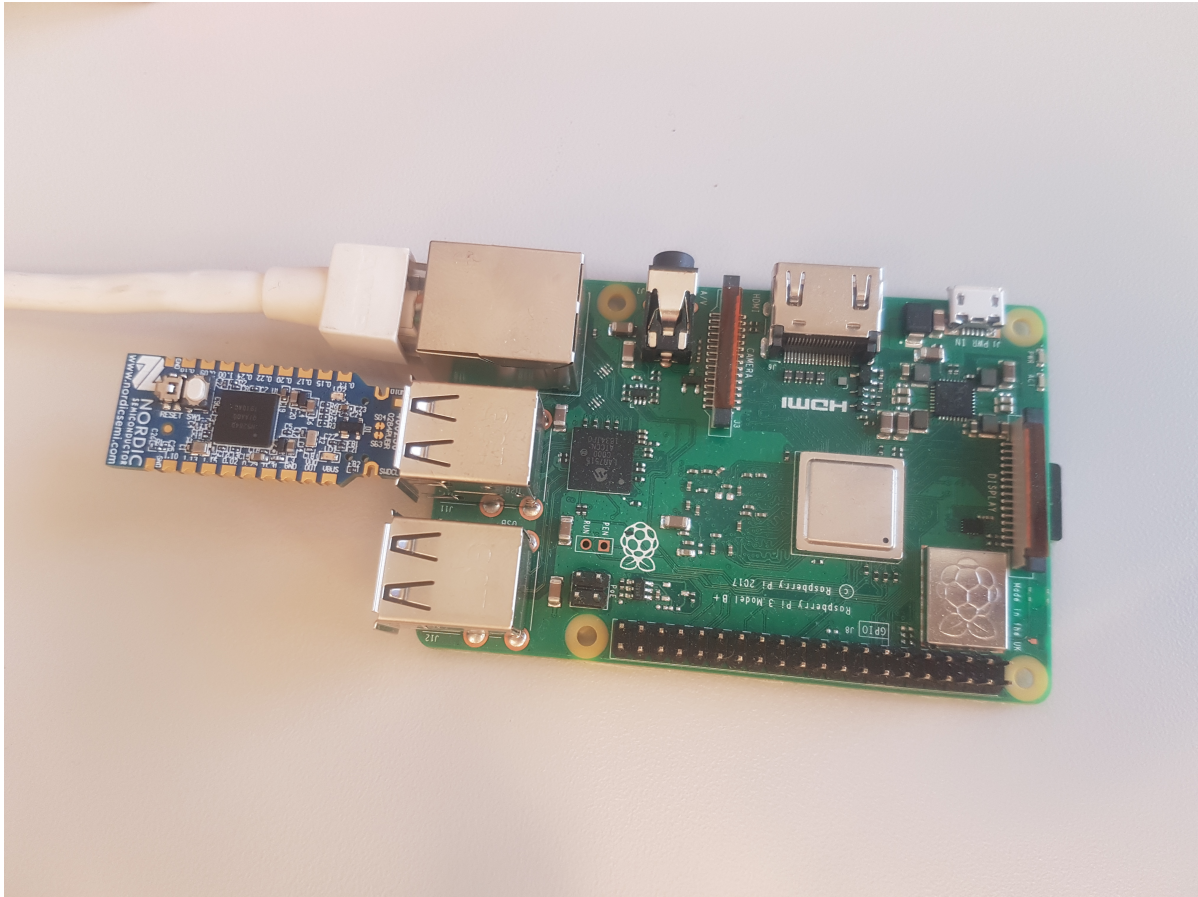
**Figure 5.2:** Raspberry Pi 3 Model B+

The Raspberry Pi were also configured as an access point with a static local IPv4 for wireless connection to ease the process of finding the external IPv4 whenever the external IPv4 changed.

The MQTT-SN gateway on the Raspberry PI use a public MQTT broker and this can easily be changed by setting up and configuring a MQTT broker locally instead.

### **5.1.5 nRF52840 as the dongle for RPi**

The Raspberry Pi 3 Model B+ does not have the IEEE 802.15.4 standard that is required for Thread. Thus the Raspberry PI would require a network co-processor(NCP) in the application layer to be able to communicate with OpenThread devices through a serial interface over the Spinel protocol, where Spinel is a general management protocol initially designed to support Thread-based NCPs.



**Figure 5.3:** Raspberry Pi 3 Model B+ with NCP dongle

An nRF52840 dongle was required to be able to connect the Raspberry Pi to the Thread network. The Thread NCP example code from Nordic Semiconductor (nRF5 SDK for Thread and Zigbee v3.2.0) was used and flashed on to the usb dongle.

### 5.1.6 Ready for Thread

With the NCP setup and OpenThread running on the Raspberry Pi it was possible to get the wireless personal area network up and running.

**Listing 5.1:** Wireless Personal Area Network Status

```
1 pi@raspberrypi:~ $ sudo wpanctl status
2 wpan0 => [
3     "NCP:State" => "associated"
4     "Daemon:Enabled" => true
5     "NCP:Version" => "OPENTHREAD/20180926-00632-g2279ef61; NRF52840"
6     "Daemon:Version" => "0.08.00d (; Jun 11 2019 09:10:53)"
7     "Config:NCP:DriverName" => "spinel"
8     "NCP:HardwareAddress" => [F4CE36B7BCC5EB54]
```



---

```
9      "NCP:Channel" => 11
10     "Network:NodeType" => "router"
11     "Network:Name" => "NordicOpenThread"
12     "Network:XPANID" => 0xDEAD00BEEF00CAFE
13     "Network:PANID" => 0xABCD
14     "IPv6:LinkLocalAddress" => "fe80::da:8385:d6d1:2ca9"
15     "IPv6:MeshLocalAddress" => "fdde:ad00:beef:0:e93c:64c1:87d8:7e9a"
16     "IPv6:MeshLocalPrefix" => "fdde:ad00:beef::/64"
17     "com.nestlabs.internal:Network:AllowingJoin" => false
```

We can see the status of the Thread network from listing 5.1.

## 5.2 Thread with existing robots

None of the current robots have the hardware to support Thread. Grindvik made a legacy layer in order to bridge the gap between the new C++ application using Thread and the older BLE communication used by the robots. The legacy layer software was available, but the new hardware had to be added to the robot.

### 5.2.1 Toolchain

There are many different software and tools available that can be used, and there are many different ways for developing and flashing software to the robots and the development kits. It is possible to use other tools than those that are suggested here.

#### SEGGER

SEGGER is software development tool for embedded systems (SES) which have a free license for development with nRF system-on-chip, which made using SES an easy choice when working with the nRF52832 system-on-chip on the robot. SES can also be used to develop software for the nRF-dongles and all of the example-code from the software development kits from Nordic Semiconductor contains a SES project file.

#### nRF connect

nRF Connect with the programmer utility from Nordic Semiconductor is designed to be used with nRF51, nRF52 and nRF53 development kits and dongles. It made the flashing of development kit and the dongles easier as long as you had compiled a \*.hex file and used a SoftDevice, where a SoftDevice is a precompiled and linked binary software developed by Nordic Semiconductor.

---

## nRF52840 development kit

The nRF52840-DK is a Bluetooth 5, Bluetooth mesh, Thread and Zigbee development kit for the nRF52840 system-on-chip. Having a development kit is not required but it is highly recommended when developing software for the nRF-dongles since the nRF-dongles does not have a real debugging option.

## 5.3 nRF52840 dongle for robot

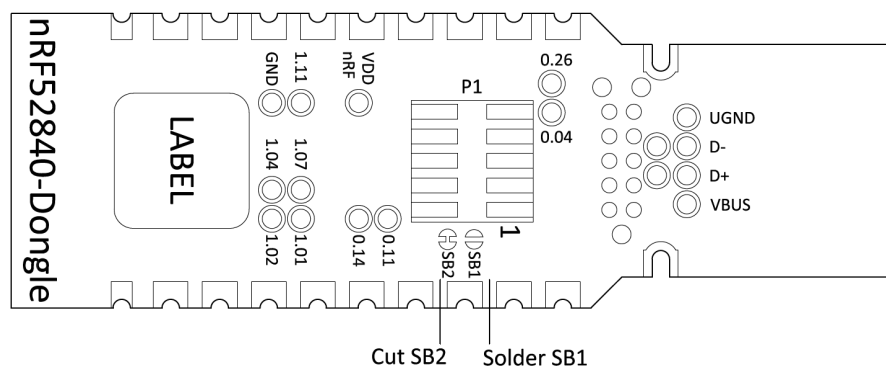
The nRF52840 dongle can be powered from different sources and would have to be modified to be able to use an external power source.

### 5.3.1 Internal regulated source

The default power supply of the nRF52840 dongle is the USB interface. The USB interface supplies power to the on-chip high voltage regulator of the nRF52840 system-on-chip.[11]

### 5.3.2 External regulated source

The nRF52840 dongle can be configured to be supplied from an external regulated 1.8-3.6 V source through the VDD OUT connection point. [11]



**Figure 5.4:** SB2 cut, SB1 soldered

As seen from figure 5.4 SB2 must be cut and SB1 must be soldered in order to use an external power source. The USB will no longer provide power which means that the dongle can no longer be flashed via USB without reversing this process. An alternative option in order to be able to flash the dongle without USB is to solder a SWD interface directly on the dongle.

---

### 5.3.3 Legacy layer on the NRF robot

The legacy layer use the I2C communication bus that was available on the NRF robot, where the nRF52840-dongle acts as a I2C slave. The first coordinate pair the robot sends to the server is the position of the robot where the rest of the coordinate pairs are obstacles. The only thing the robot receive is waypoints, e.g. where the server wish the robot to go.

**Listing 5.2:** Legacy Layer main.c

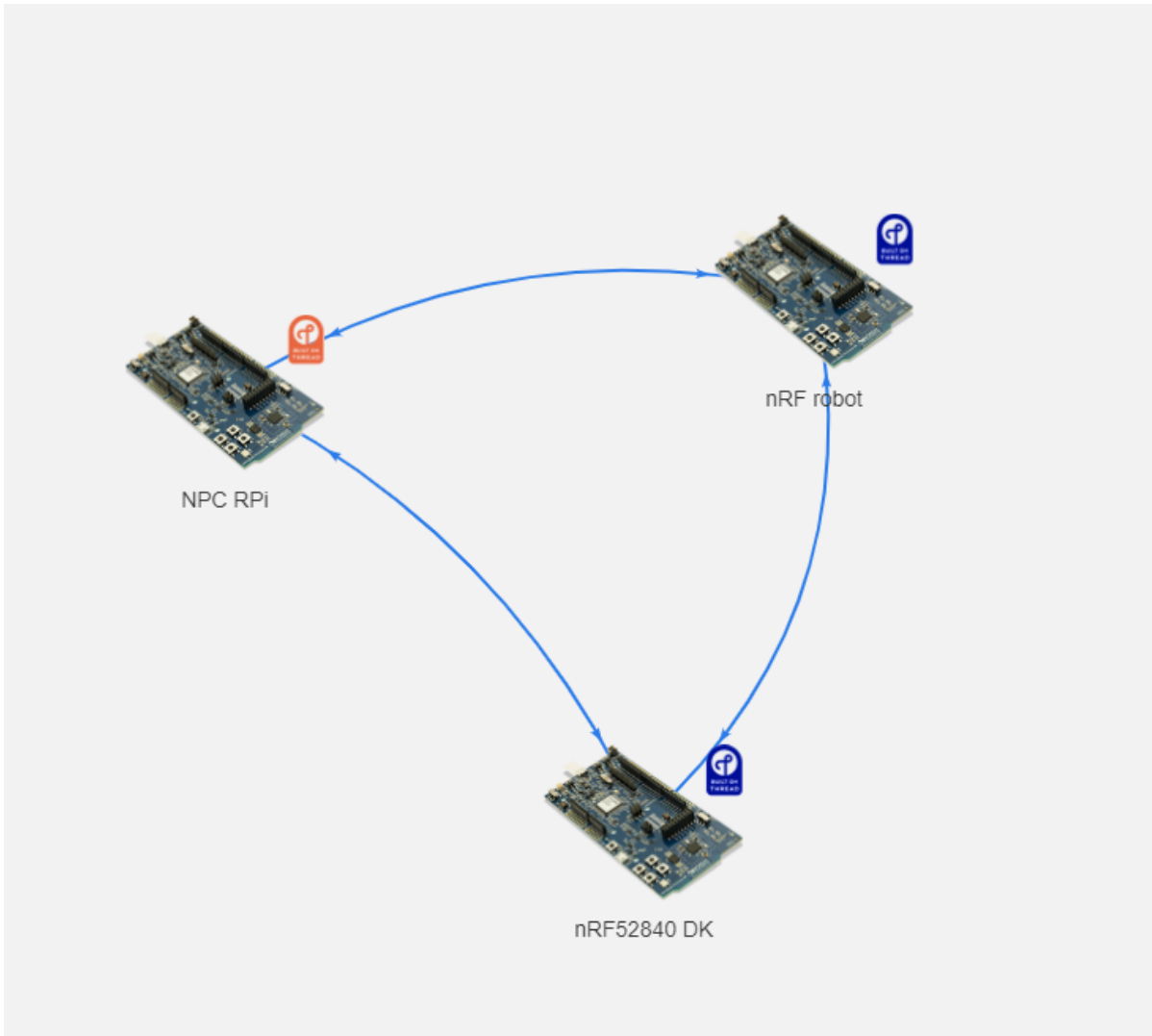
```
1 static nrfx_twis_config_t m_twis_config = {
2     .addr = {0x72,
3             0x73},
4     .scl = NRF_GPIO_PIN_MAP(0, 31),
5     .scl_pull = (nrf_gpio_pin_pull_t)NRF_X_TWIS_DEFAULT_CONFIG_SCL_PULL,
6     .sda = NRF_GPIO_PIN_MAP(0, 29),
7     .sda_pull = (nrf_gpio_pin_pull_t)NRF_X_TWIS_DEFAULT_CONFIG_SDA_PULL,
8     .interrupt_priority = NRF_X_TWIS_DEFAULT_CONFIG_IRQ_PRIORITY};
```

We can see from a snippet of the legacy layer from listing 5.2 that pin 31 and pin 29 on the nRF52840 dongle is used for the I2C communication with the NRF robot.

The reason as to why the NRF robot were chosen with the new Thread communication was because the robot already contained a way to read or write on the I2C bus. This will make it possible to send a MQTT message to a topic the robot subscribe to and the nRF52840 dongle will request to read or write on the I2C bus.

### 5.3.4 Thread topology with the nRF robot

The nRF52840 development kit were flashed with a similar legacy layer that were used on the nRF52840-dongle to simulate having more than one robot in the Thread network. The development kit would not send any messages over the Thread network, but would instead be used to listen and analyse how the network would connect and self-heal if a device were to be disconnected from the network.



**Figure 5.5:** Thread topology

A clear real-time image of the Thread network can be seen from figure 5.5 by using the Thread Topology Monitor(TTM) from Nordic Semiconductor.

# 6 C++ APPLICATION WITH A REAL ROBOT

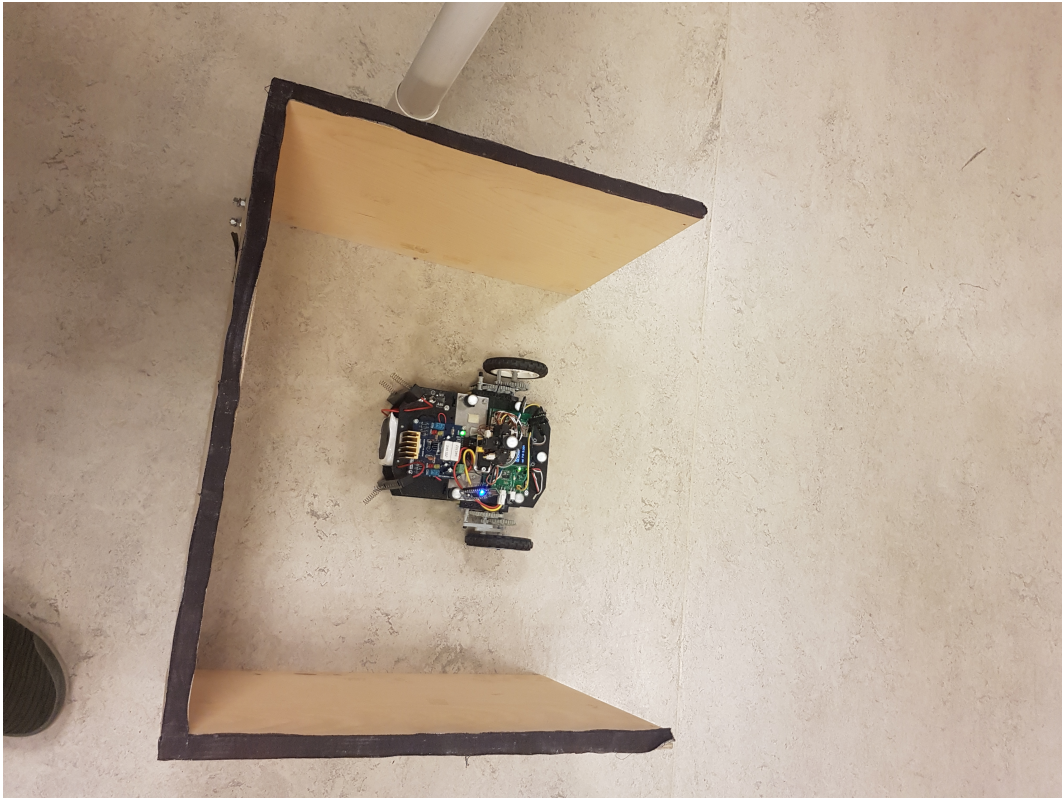
With the Thread network, the border Router, the MQTT-gateway and the legacy layer set up it was possible to run the C++ application with a real robot in the real-world.

## 6.1 Testing with the NRF robot

The NRF robot would connect to the Thread network through the nRF52840-dongle, where the Raspberry PI would act as a MQTT-SN gateway for publishing to the public MQTT broker used in this thesis. The C++ application would subscribe to the topic published by the NRF robot and read the feed that was handled by the public MQTT broker.

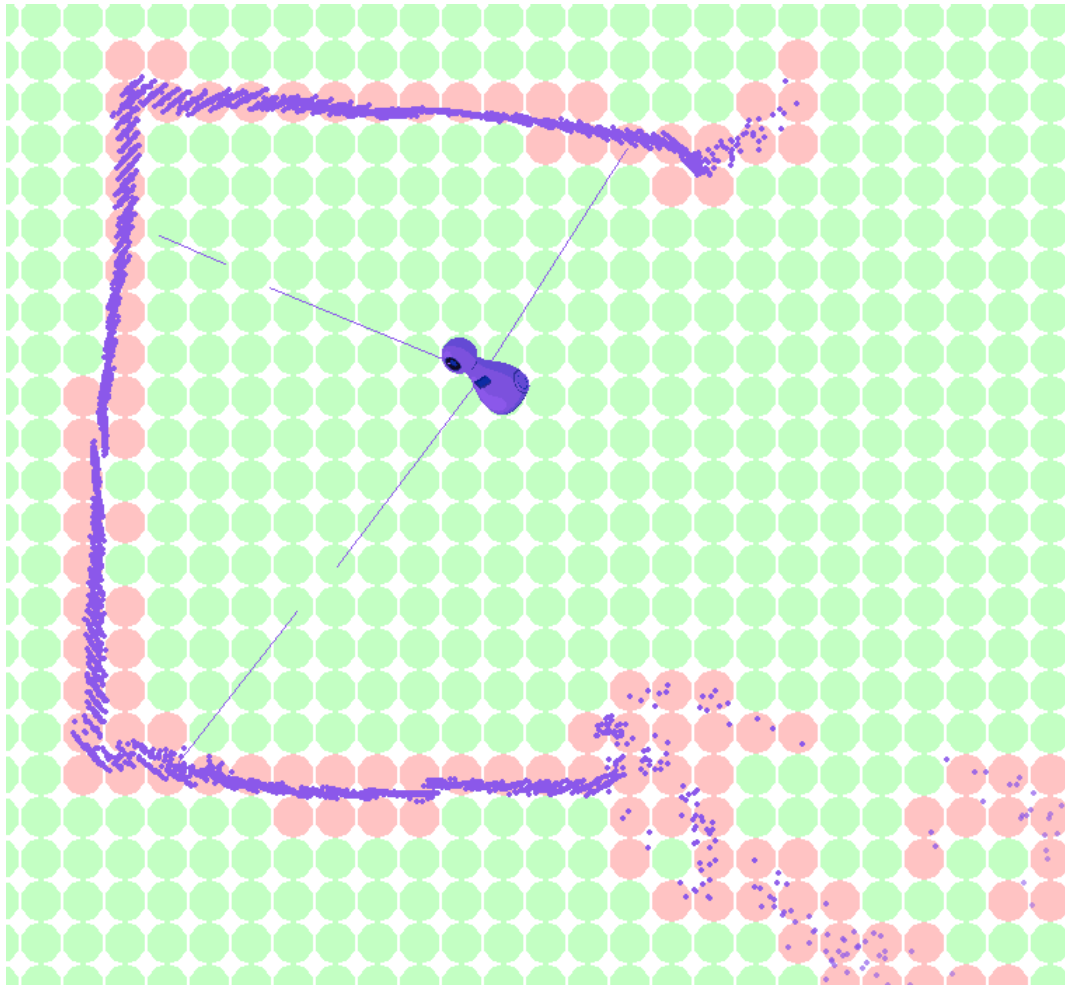
### 6.1.1 Testing messages sent by the robot

The charging station for the real robots were set up to see how well the real robot would detect obstacles in the real-world. The robot would stand still and only the measurement messages from the infrared tower would be used. This would give a good indication to see if the obstacles detected by the infrared sensor tower on the robot and the messages published by the robot were the same.



**Figure 6.1:** charging station

Figure 6.1 show how the charging station looks like in the real-world and how the NRF robot were positioned in comparison with the charging station.

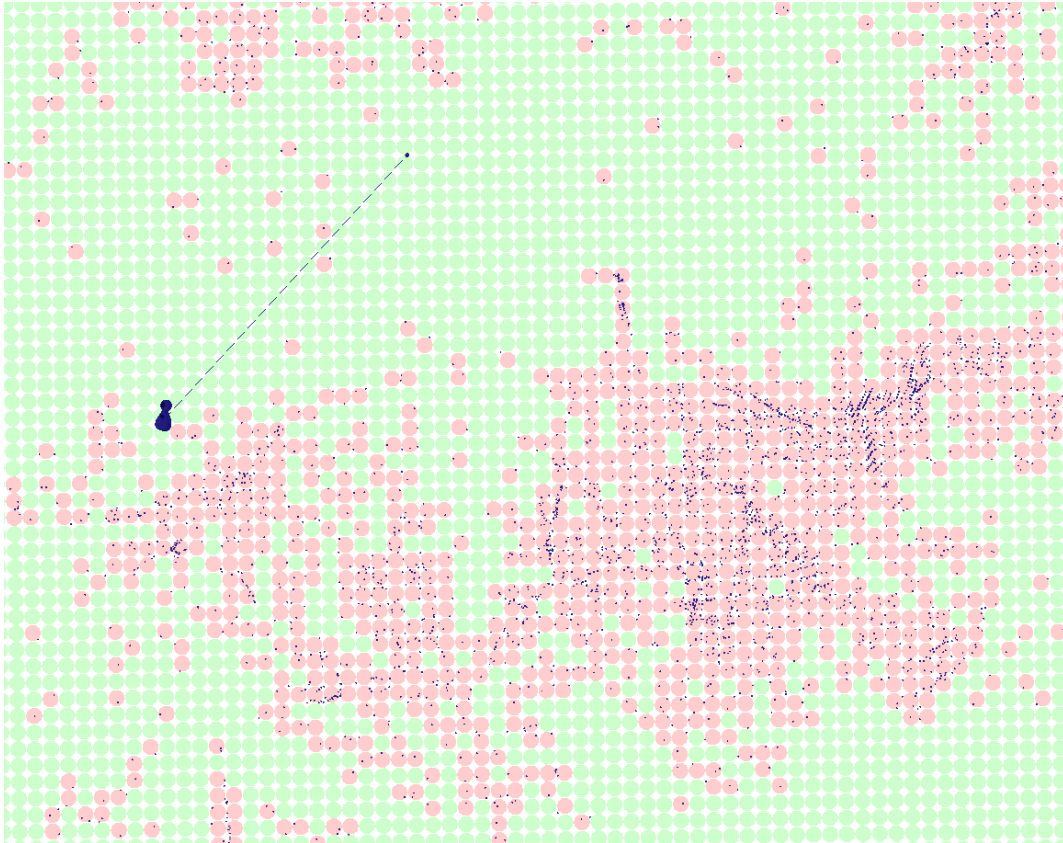


**Figure 6.2:** charging station2

As can be seen from figure 6.2 the robot is able to detect obstacles and place them in the grid of the server. The message protocol is different from the Java application as in the robot will only transmit the current position of the robot and the coordinates to an obstacle, whereas the old BLE communication protocol would update the server with the current status of the robot, e.g. if the robot was moving or idle.

### **6.1.2 Testing messages sent by C++ application**

An attempt was made to try and drive the robot around the office. This was accomplished by manually send a coordinate pair with MQTT to the robot that would act as a waypoint in which the robot would drive to. This would give a good indication to see if the waypoints published by the C++ application would be properly received by the NRF robot and if the NRF robot would physically move in the real-world.



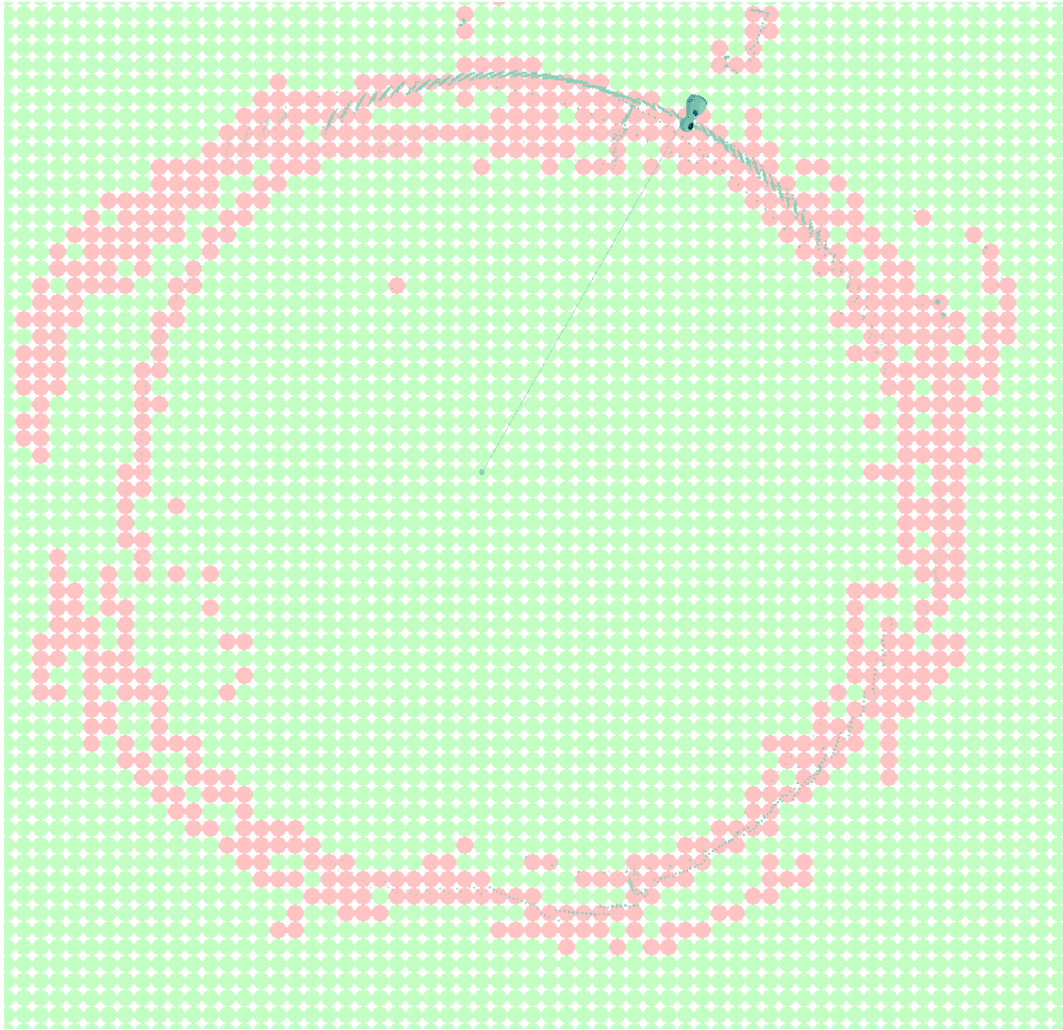
**Figure 6.3:** drive office01

Driving the robot manually was successful as can be seen from figure 6.3. The robot would drive to the waypoint it was assigned and the infrared tower would find obstacles. The robot would however drive blindly and crash into obstacles instead of stopping and requesting a new waypoint even if there is supposed to be software on the robot that should handle and detect if the robot is about to crash.

### **6.1.3 Testing movement and sensor in a controlled space**

There is a large circular track with walls in the office that is often used to run real robots in the real-world. This would give a chance to see how the server would handle the measurements sent by the sensor tower and how the C++ application would place obstacles in a more controlled space.





**Figure 6.4:** drive circle

As can be seen from the figure 6.4, the robot is able to draw a fairly accurate map of the circle. Since the coordinates to the obstacle is in relation to the robot current position, then the C++ application is able to place these obstacles fairly accurate when the robot is moving.

One of the reason as to why this circle is not as accurate as it should be is because the right motor of the NRF-robot began to show signs of wear and tear. The motor would not always turn as it should and therefore the robot would physically be at a different coordinate then it was supposed to be even if the robot reported back the current position, and therefore the server would place the obstacles wrongly.

## 6.2 Improving

Some improvements had to be made after testing to see how the C++ application would handle a real robot in the real world.

---

## 6.2.1 New robot event

The C++ application had no indication if a robot ever reached the destination given by a waypoint and would therefore keep publishing new waypoints through MQTT at very fast intervals. The old BLE communication used status parameters which the robot would send to the server to avoid getting spammed with new commands. Since this does not currently exist in the new communication protocol it was decided that the server would handle this instead.

A new event called `ROBOT_IDLE` were created which compared the messages read through the MQTT feed. The `ROBOT_IDLE` event would then compare the robots last position with the current position by reading the current topic of the MQTT feed and will give the robot an idle status if those coordinates are the same. In practice this means that other classes will be able to listen to the `ROBOT_IDLE` event by using a callback function.

**Listing 6.1:** `ROBOT_IDLE` callback

```
1 robots.enable_callback(NTNU::application::SLAM::robots_events::ROBOT_IDLE,
2                               [&](std::any context) {
3     try {
4       auto [id, x, y] = std::any_cast<std::tuple<std::string,
5                                       int16_t,
6                                       int16_t>>(context);
7       std::cout << "Robot [" << id << "] idle at {" << x << ", " << y << "}\n";
8     }
9   }
10  catch (const std::bad_any_cast& e) { std::cout << e.what(); }
11 });
```

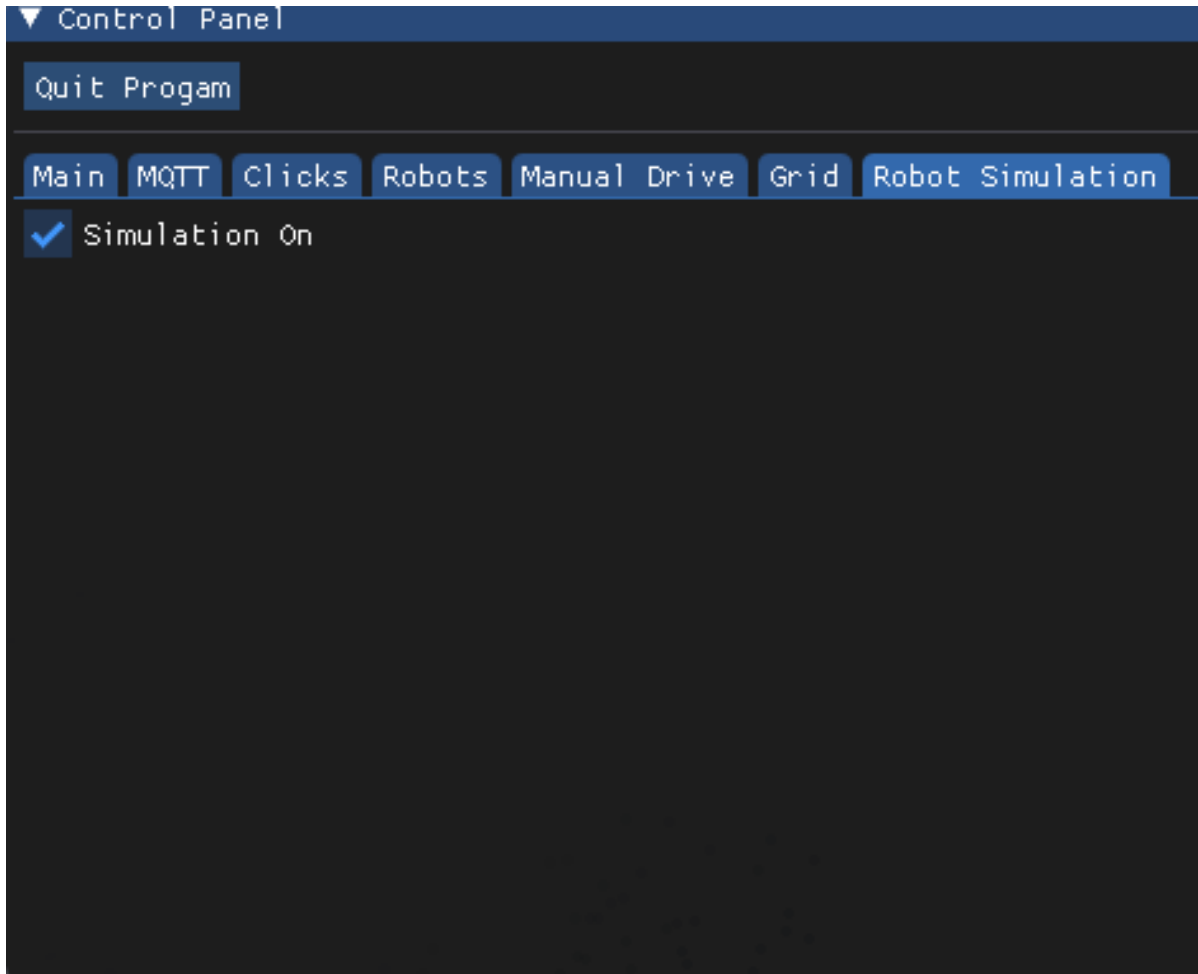
Listing 6.1 show how to implement the callback function for `ROBOT_IDLE` in the C++ application.

## 6.3 User interface

Some changes were made to the user interface. Two new classes for the panel were added and an existing panel got some additional functionality.

### 6.3.1 New simulation panel

A new class was made for the simulation panel, which inherits from the panel class. The class use the same `NTNU::application::SLAM::robot_simulation` as earlier for a randomized simulated robot.

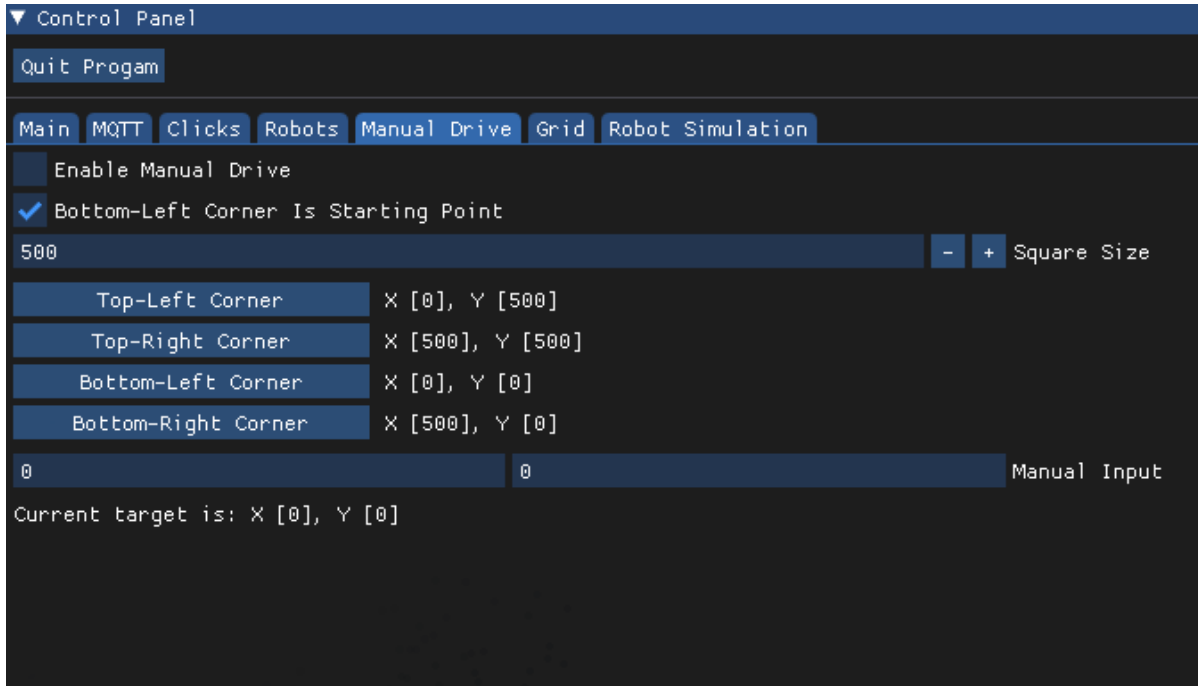


**Figure 6.5:** gui simulation

From figure 6.5 it can be seen that the panel use the same design as the previously existing simulation panel.

### **6.3.2 New manual panel**

A new class was also made for the manual drive panel. This panel did not previously exist as a class of its own, but were instead part of the main class.

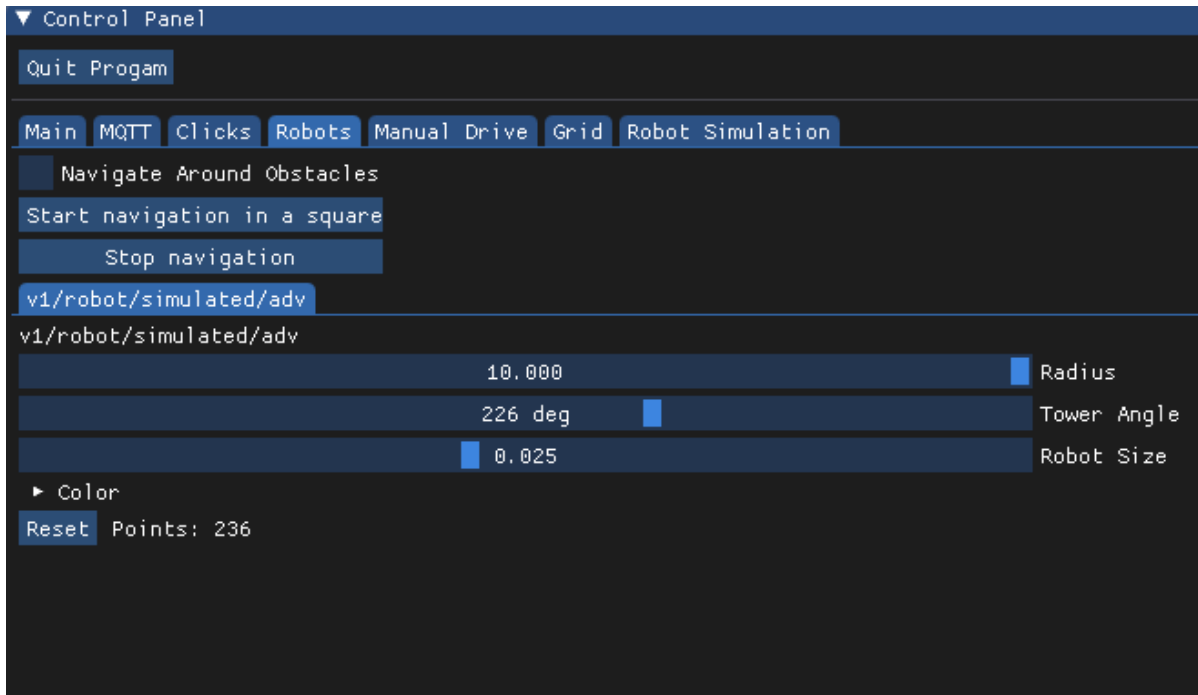


**Figure 6.6:** manual

Figure 6.6 shows the manual drive tab. No changes to the functionality or design the panel.

### 6.3.3 Changes to robots panel

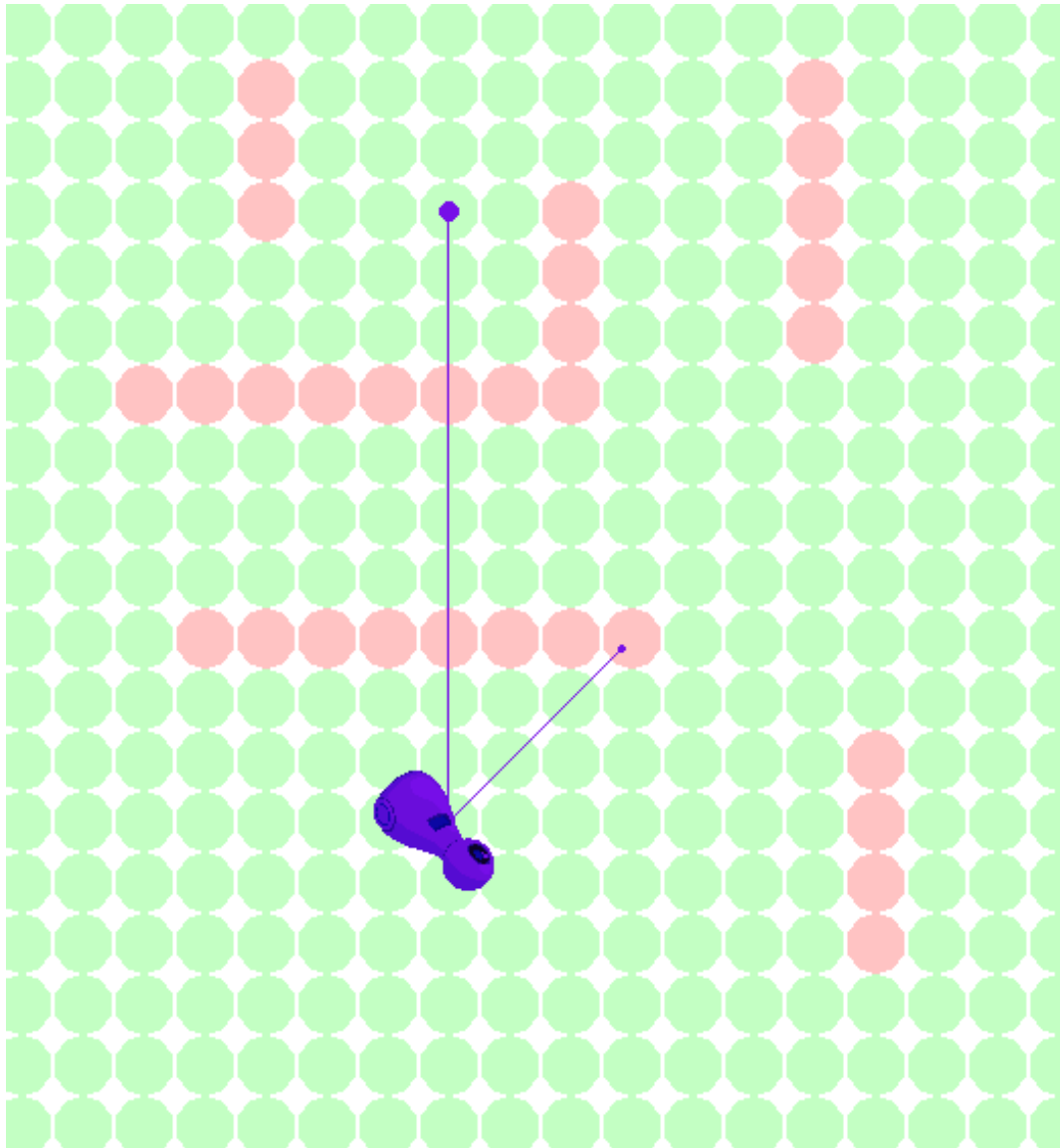
A new navigation panel were added but it was later decided to merge it with the robot panel instead to avoid too many options or confusion with the user interface.



**Figure 6.7:** gui robot

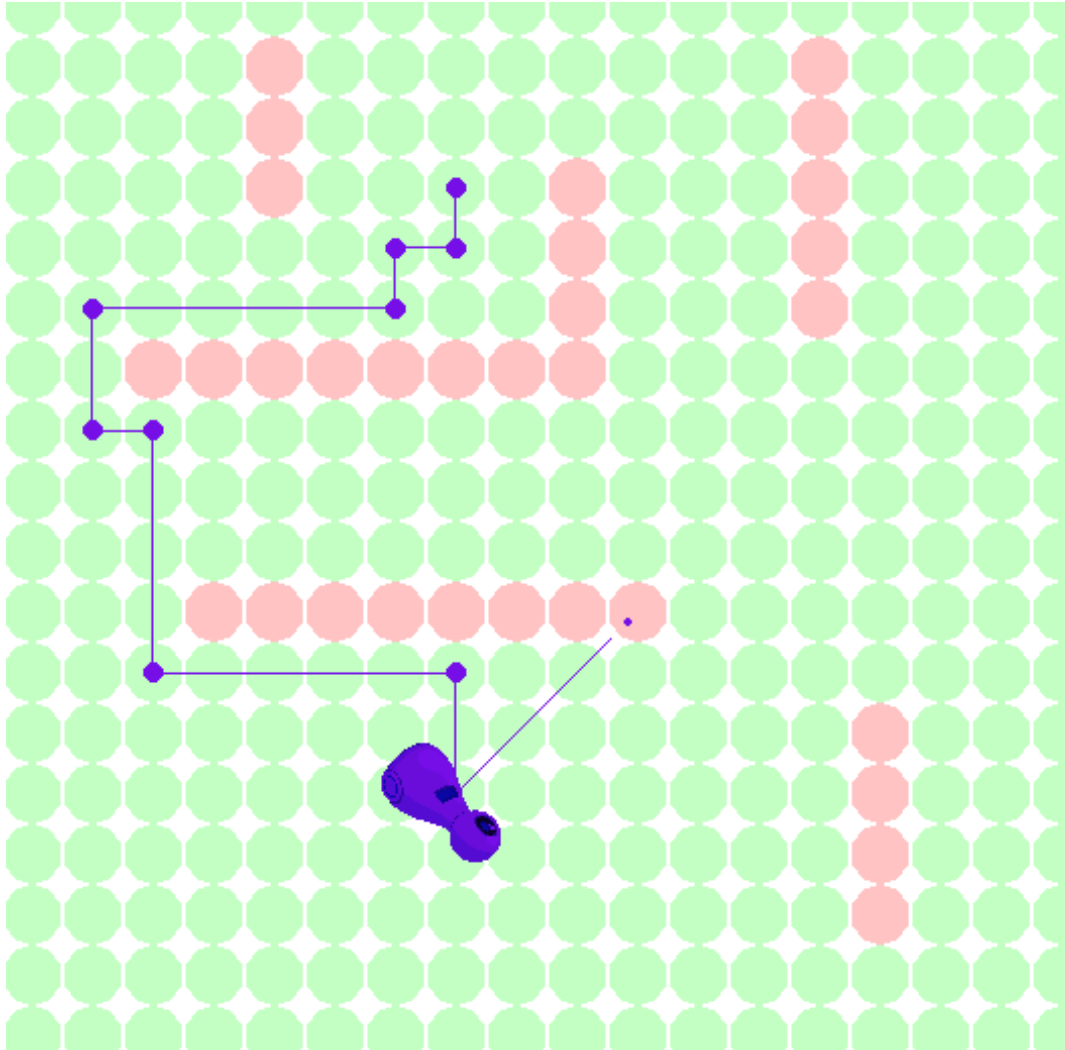
The robots panel got two new functionalities as can be seen from figure 6.7. The "Start navigation in a square" button will look at the current position of the robot and give a new waypoint relative to the current position of the robot whenever the robot would stop moving, e.g. ROBOT\_IDLE event. The waypoints published by the C++ application will resemble a square, hence the name "navigation in a square".

The stop navigation button will publish a waypoint to the current position of the robot and the robot would therefore stop navigation since it had already reached the destination.



**Figure 6.8:** navigate off

We can see a simulated world with simulated obstacles from figure 6.8. The C++ application would publish a new waypoint relative to the current robot position. This was also confirmed by debugging and looking at the current target on the real-robot. A new waypoint would be published when the simulated robot reached the current target waypoint. This was also confirmed by looking that the current target of the real robot would update with the new waypoint.



**Figure 6.9:** navigate on

The navigation also includes the use of pathfinding through the A\* search algorithm from the Boosts.graph library. In practice this means that if there are obstacles between the target waypoint and the robot, then the algorithm will find the shortest path between the nodes that are not obstructed. This function do not however handle corner cases well, where for example it will detect obstacles and change the pathing between the final waypoint and the robot, but it would not handle if the final waypoint is placed on an already detect obstacle.

### **navigate\_square**

The problem with the faulty motor of the NRF robot became a bigger issue and caused the robot to never reach the waypoint it was supposed to. This caused the navigation software on the robot to try and compensate for a missing motor by rotating the one working motor back and forth. This caused the server to never receive the ROBOT\_IDLE event, which resulted in the C++ application to never give out a new waypoint.

---

**Listing 6.2:** navigate\_square

```
1 auto navigate_square = [&](const std::string& robot, bool state) {
2     std::pair<int, int> target;
3     auto get_pos = robots.position(robot);
4     if (!get_pos)
5         return;
6     auto source = get_pos.value();
7     auto result = NTNU::application::SLAM::utility::coord_to_row_col(grid,
8                                                                     source.first,
9                                                                     source.second);
10    if (!result)
11        return;
12    auto [robo_row, robo_col] = result.value();
13    if (state) {
14        auto coords = robots.get_square_values(robo_row, robo_col);
15        for (int i = 0; i < coords.size(); i++) {
16            target = { coords[i].first, coords[i].second };
17            update_path_for_robot(robot, target);
18            auto next_time = system_clock::now() + milliseconds(1000);
19            sleep_until(next_time);
20        }
21    }if(!state)
22    {
23        target = { robo_row, robo_col };
24        update_path_for_robot(robot, target);
25    }
26 };
```

We can see that a timer was added on line 18 and line 19 from listing 6.2. This line was added because of the problems with one of the motor of the NRF robot when trying to navigate in the real-world. This timer was added to simulate the ROBOT\_IDLE event that would have occurred when the robot reached the destination. This will work in practice with a simulated robot to go through a sequence of waypoints but will however not work with a real robot in the real world. This should be replaced with a callback to ROBOT\_IDLE when using a real robot.



# 7 DISCUSSION AND FURTHER

## WORK

### 7.1 Discussion

#### 7.1.1 MQTT versus BLE communication

One of the biggest current challenges with the MQTT protocol is that there are currently only one robot that have the hardware and software to support it via Thread. The current version of the message protocol used in this thesis is very simple with only two message pairs, current robot coordinates and detected obstacles coordinates.

#### 7.1.2 C++ application versus Java application

The Java application have been through several development cycles and have had added functionality and improvement through many years. It is therefore natural that the Java application currently contains more features when it comes to algorithms that are designed to better control how the robots will explore and map a room. The downside to having been through many development cycles and many different developers is that the consistency of the code itself varies. Several classes in the Java applications have a single responsibility, while other classes have a large number of dependencies and too much functionality.

The C++ application can give the robot a manual target, navigate with a set of predetermined waypoints and click targeting. The C++ server do however lack algorithms for steering the robots in an intelligent manner. The C++ application also lack a good simulation of how a real robot would act in a simulated environment. One of the goals of Grindvik when he designed the new C++ application was that the new classes should have a single responsibility and achieve decoupling through events.

---

## **7.2 Further work**

Contains suggestions that need to be looked closer at for continuation of the thesis for the future.

## **7.3 NRF robot**

The NRF robot are not in a good state and there need to be made improvements for future work with the overall project.

### **7.3.1 Right motor**

The right motor on the NRF robot started to act strange early in the thesis. The motor stopped working completely towards the end of the thesis and it is required to either fix the faulty motor or give the robot two entire new motors.

### **7.3.2 Trackball**

The NRF robot use a trackball in the back in order to turn. This trackball is under heavy weight and would become dirty after some use. This caused the friction of the trackball to increase and the robot would have difficulty turning, or in some cases it would have a difficult time just driving in a straight line. This might be mitigated some with a larger trackball, or look at the weight distribution of the robot.

### **7.3.3 Anti-collision**

The NRF robot should have some anti-collision measurements that is not related to the server. The anti-collision did not seem to work at all though and should be looked at.

### **7.3.4 Robot initialization**

There is a problem with the NRF robots initialization process when used with the nRF52840-dongle. It will look like the robot will do the initialisation process normally and the infrared sensor tower would start rotating and detecting obstacles. The robot would not however always initialise the dongle and therefore it would not be able to communicate with the Thread network, e.g. the C++ application would not get any messages sent by the robot.

### **7.3.5 Gear on the wheels**

There is excessive gear ratios on from the motor to the wheels. This gear ratio causes the NRF robot to have a reduced speed and the robot need a longer axle to the wheels. This also causes

---

the wheels to be further away from the robot and in practice this makes the robot wider than it has to be.

With the already faulty motor of the NRF robot, it should be considered if it might be better to just change the type of motor to possibly avoid the excessive gear ratios in the future.

### **7.3.6 Upgrade SoC**

The NRF robot uses a nRF52832 system-on-chip which does not support Thread. There is however a newer nRF52840 system-on-chip that supports Thread. With a new system-on-chip it will be possible to avoid using an external dongle for the Thread communication. It would not be any major changes to the source code when changing the system-on-chip, but there will however be a need to modify the current printable circuit board for the change, since the newer system-on-chip is of a different physical design.

## **7.4 C++ application**

The C++ application has a good frame for further developing the application. The C++ application still misses many of the features that were in the Java application.

### **7.4.1 SLAM**

SLAM algorithms need to be added to the C++ application.

### **7.4.2 navigate\_square**

The function `navigate_square` in `main.cpp` must be changed in order to use it with a real robot. The function currently has a timer in it to simulate that a robot would reach the destination provided by waypoints. Replace the timer and add a callback to the `ROBOT_IDLE` state when using the C++ application with a real robot.

### **7.4.3 Robot status**

The status of the real robots are currently handled with events by the server. The events are set by reading the MQTT feed sent by the various robots and compared by previous messages from the MQTT feed. This works well in theory but in practice this could slow down the C++ application by having to compare every message from the MQTT feed. The status of the robots were previously handled by the robots themselves and the Java application would only get a status message if the status changed.

---

It should be looked into if it might be better to change the communication stack for the MQTT feed to include the current state of the robot.

#### **7.4.4 Better support for multiple robots**

The C++ application is able to identify the robots by the topic published by the robot. The C++ application will however publish the same MQTT message to all the topics and all connected robots will therefore receive the same waypoints.

This have been a very low priority since there is only one robot that is currently able to work together with the C++ application.

### **7.5 General suggestions to the project**

These are more general suggestions and ideas on how to possibly improve the overall project.

#### **7.5.1 Run robot with nRF52840 DK**

It should be considered at looking at the possibility to use the nRF52840 development kit directly on the robot since the development kit already supports Thread, and it have general purpose input/output(GPIO).

#### **7.5.2 Local MQTT Broker**

MQTT need a MQTT Broker to handle the feed. This is currently done with a public MQTT Broker, which in practice means that that anyone can subscribe or publish to the topics used in this thesis. Another thing to point out is that the public MQTT broker may suffer to high traffic and cause the MQTT feed to slow down. A solution to this is to set up a local MQTT broker on either the PC running the C++ application or run it from the Raspberry PI.

##### **Pros to a public broker**

Everyone can get access to it through the internet, which in practice means that it is possible to run the C++ application in one part of the country and run the robot in another.

##### **Cons to a public broker**

Since everyone have access to it, it may be susceptible to abuse and the MQTT feed could be contaminated with false messages which would in practice be able to control the robot with the correct message format.

---

# Appendix

## .1 Files overview

Shows a simple directory tree of the contents delivered in a zip-file alongside the thesis.

```
master_thesis_marius_blom.zip
├── cpp
│   ├── exe
│   │   └── Release.zip
│   └── src
│       └── SLAM-application.zip
├── document
│   └── thesis.pdf
├── Java
│   └── java_nilssen_2018.zip
├── NRF_robot
│   ├── legacy_layer
│   │   └── legacy_layer.zip
│   └── slam_application
│       └── src.zip
├── quickstart_guide
│   ├── cpp_application.pdf
│   ├── NRF_segger.pdf
│   └── Raspberry_PI.pdf
└── Raspberry_PI
    └── paho-mqtt-sn-gateway.conf
```

# BIBLIOGRAPHY

- [1] Ian Arnes. *What is the OSI model?* URL: <https://medium.com/software-engineering-roundup/the-osi-model-87e5adf35e10>. (accessed 26.09.2019).
- [2] boost. *boost c++ libraries*. URL: <https://www.boost.org/>. (2019).
- [3] boost. *The Boost Graph Library*. URL: [https://www.boost.org/doc/libs/1\\_66\\_0/libs/graph/doc/index.html](https://www.boost.org/doc/libs/1_66_0/libs/graph/doc/index.html). (2019).
- [4] Elias Daler. *Dear ImGui SFML Binding*. URL: <https://github.com/eliasdaler/imgui-sfml>. (2019).
- [5] Eclipse. *paho-mqtt*. URL: [find%20url](#). (2019).
- [6] Torstein Grindvik. “Master thesis”. In: (2019).
- [7] Jan Haller. *Thor C++ library*. URL: <http://www.bromeon.ch/libraries/thor>. (2019).
- [8] Johan Korsnes. “Development of a Real-Time Embedded Control System for SLAM robots”. In: (2018).
- [9] Endre Leithe. “Embedded nRF52 robot”. In: (2019).
- [10] *MQ Telemetry Transport*. URL: <http://mqtt.org/>. (2019).
- [11] *nRF52840 Dongle*. PCA10059 v1.0.0. v1.1. Nordic Semiconductor. Jan. 2019.
- [12] Mats Gjerset Rødseth and Thor Eiving Svergja Andersen. “System for Self-Navigating Autonomous Robots”. In: (2016).
- [13] Nordic Semiconductor. URL: <https://www.nordicsemi.com/>. (2019).
- [14] *Simple and Fast Multimedia Library*. URL: <https://www.sfml-dev.org/>. (2019).

