Master's thesis

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

Frida Pedersen Eidsnes

Improvement and Development of Collision and Obstacle Avoidance in nRF52850 Mobile SLAM Robots

Master's thesis in Cybernetics and Robotics
Supervisor: Tor Onshus
June 2024

NTNU
Norwegian University of
Science and Technology

Frida Pedersen Eidsnes

Improvement and Development of Collision and Obstacle Avoidance in nRF52850 Mobile SLAM Robots

Master's thesis in Cybernetics and Robotics
Supervisor: Tor Onshus
June 2024

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

Preface

This report is the resulting product of a master's thesis at the Department of Engineering Cybernetics at the Norwegian University of Science and Technology (NTNU). The thesis is part of a larger SLAM robot project led by Tor Onshus, who has been my supervisor and has assisted me throughout the thesis. The SLAM robot project aims to establish multiple cooperative ground robots and drones to map and localize unknown areas. Because all technology development should strive to contribute to making the world a better place for all creatures, a goal is for the project to align with one or more of the United Nations' Sustainable Development Goals (SDGs). Particularly Goal 3: "Good Health and Well-being" and Goal 15: "Life on Land" are aligned in this project. Cooperative SLAM robots can rescue individuals in need in challenging environments where human operation is difficult. Additionally, such a system is valuable for investigating areas in a sustainable manner, supporting research on how to care for and restore nature.

Tor Onshus has provided invaluable guidance in determining the direction of my thesis and has offered advice on the content, level of detail, and structure of the report. The development and writing of the report have been completed solely by me.

Thank you to everyone who has supported me through this master's thesis and through five years of hard work towards this end goal.

Summary and conclusion

This master's thesis is part of a SLAM robot project aimed at developing cooperating robots for mapping and localization of unknown areas. In this thesis, existing mobile ground robots are used. These robots already have significant functionality implemented, so the goal of this thesis is to improve existing functionalities as well as implementing new ones. The thesis primarily focuses on collision and obstacle avoidance for these mobile ground robots. Additionally, several smaller improvements were made. The main files used for collision and obstacle avoidance in the robot code were cleaned up and slightly restructured. This cleanup resulted in the deletion of a significant number of unused code lines, making these files more readable. However, the robot code would still benefit from a more thorough cleanup. Furthermore, a test program was successfully created to verify the functionality of the robots, assisting those who repair the robots when they have malfunctioning hardware. The resulting test program was of great help in robot repairs. Lastly, the work done in this thesis was merged with contributions from other students working on the SLAM robot project, to ensure cohesive progress.

A collision avoidance version was implemented in the specialization project [Eidsnes (2023)] that was written as a build up to this master thesis. This version serves as the initial collision avoidance version in this thesis. Two additional versions were developed, and all three versions were tested in four tests designed to reveal each version's strengths and weaknesses. Each version was scored based on the robot's ability to avoid crashing into obstacles and to reach the target position when no obstacles were present. Version 2 performed the best and received the highest score. Subsequently, the obstacle avoidance functionality was investigated. For the robot to navigate around obstacles, it is crucial that it stops upon approaching one. Since Version 2 of collision avoidance performed the best, it was used as the baseline for implementing the obstacle avoidance algorithms.

In the specialization project [Eidsnes (2023)], two obstacle avoidance algorithms were designed but not implemented. These two algorithms, along with one additional algorithm, were adjusted/developed, implemented, and tested in a different test set designed to reveal strengths and weaknesses in obstacle avoidance.

As mentioned, Version 2 of collision avoidance performed the best overall in the tests and was therefor the baseline for implementing the obstacle avoidance algorithms. However, Version 2 does not perform perfectly. The robot sometimes crashes when approaching small obstacles or when not approaching obstacles head-on. Additionally, it sometimes stops when no obstacles are in front of it. These problems are mainly due challenges with the robot having a turning sensor tower, resulting in blind spots between each sensor. Furthermore, the nRF board that contains the robot software appears to have a capacity problem, which forces a solution where the sensor tower skips a degree each time it turns, to avoid software crashes. This weakens the robot's ability to detect obstacles.

Since the obstacle avoidance algorithms use Version 2 of collision avoidance, the same issues persisted after implementing the obstacle avoidance algorithms. From the testing of the obstacle avoidance algorithms, it was clear that Algorithm 2 most often led the robot to the target. However, there were a few instances where the robot failed to navigate around obstacles and reach the target. Additionally, the algorithm is inefficient as the robot stops every 20 degrees to check if it has turned enough to start moving forward, rather than turning the necessary degrees to start moving along the obstacle. Another issue is that the robot sometimes unexpectedly oscillates back and forth at the same position.

A third algorithm, Algorithm 3, was also created. Algorithm 3 is a further development of Algorithm 2, but testing revealed that it has many unexpected behaviors that cause the robot to crash in situations where it normally would not and generally behaves unpredictably. Algorithm 3 is expected to be more efficient than Algorithm 2 if these issues are resolved.

For future work, Version 2 should be further improved for better collision avoidance. Alternatively, a completely new collision avoidance system could be implemented using different sensors. For the obstacle avoidance algorithms, either Algorithm 2 should be further developed or Algorithm 3 should be fixed before further development. Whatever solution is chosen, the algorithm should be improved to increase efficiency.

Sammendrag og konklusjon

Denne masteroppgaven er skrevet som en del av et ”SLAM-robot-prosjekt” som har som mål å utvikle samarbeidende roboter for bruk til lokalisering og kartlegging av ukjente områder. I denne oppgaven er allerede eksisterende bakkeroboter brukt. Disse robotene har allerede mye funksjonalitet implementert, så målet for denne oppgaven er å forbedre eksisterende funksjonalitet i tillegg til å implementere ny. Hovedfokuset i denne oppgaven er kollisjons- og hindringsunngåelse for robotene. I tillegg er flere små forbedringer gjennomført. Filene i robotkoden som er mest brukt i kollisjons- og hindringsunngåelse ble ryddet opp i og noe omstrukturert. Denne opprydningen resulterte i sletting av et betydelig antall ubrukte kodelinjer, noe som gjorde disse filene mer lesbare. Imidlertid vil robotkoden fortsatt kunne dra nytte av en grundigere opprydning. Videre ble et testprogram laget for å bekrefte at robotene fungerer etter å ha vært på reparasjon på grunn av hardware-feil. Det resulterende test programmet var til stor hjelp ved robotreparasjoner. Til slutt ble resultatene fra denne oppgaven integrert med arbeid utført av andre studenter som også jobber med ”SLAM-robot-prosjektet”. Dette ble gjort for å sikre en helhetlig fremdrift og samarbeid på tvers av prosjektet.

En kollisjonsunngåelsesversjon ble implementert i prosjektoppgaven [Eidsnes (2023)] som ble skrevet som en oppbygning til denne masteroppgaven. Denne versjonen er den første versjonen i denne oppgaven. To nye versjoner ble så utviklet, og alle tre ble testet i et testsett laget for å avdekke styrker og svakheter i hver av versjonene. Hver versjon ble tildelt poeng basert på robotens evne til å unngå å krasje inn i hindringer, i tillegg til evnen til å la være å stoppe når det ikke er hindringer i veien. Versjon 2 gjennomførte testene på best måte totalt sett og ble dermed tildelt flest poeng. Videre ble hindringsunngåelse utviklet. For at roboten skal kunne komme seg rundt hindringer er det vesentlig at den stopper før den krasjer inn i hindringen. Versjon 2 ble derfor startpunktet for hindringsunngåelsesalgoritmene.

I prosjektoppgaven [Eidsnes (2023)] ble to hindringsunngåelsesalgoritmer designet, men ikke implementert. Disse to algoritmene, i tillegg til en nyutviklet algoritme, ble revidert/utviklet, implementert og testet i et annet testsett designet for å avdekke styrker og svakheter i hindringsunngåelse.

Som nevnt, presterte Versjon 2 av kollisjonsunngåelse best totalt sett i testene og ble derfor brukt som grunnlag for implementering av hindringsunngåelsesalgoritmene. Imidlertid er ikke Versjon 2 perfekt. Roboten krasjer noen ganger når den møter små hindringer eller når den møter hindringer på skrå. I tillegg stopper den noen ganger når det ikke er noen hindringer foran den. Disse problemene skyldes hovedsakelig utfordringer med at roboten har et roterende sensortårn, noe som resulterer i blindsoner mellom hver sensor. Videre ser det ut til at nRF-kortet som inneholder robotprogramvaren har et kapasitetsproblem, noe som tvingte frem en løsning hvor sensortårnet hopper over en grad hver gang det roterer,

for å unngå programvarekrasj. Dette svekker robotens evne til å oppdage hindringer.

Siden hindringsunngåelsesalgoritmene brukte Versjon 2 av kollisjonsunngåelse, vedvarte de samme problemene etter implementeringen av hindringsunngåelsesalgoritmene. Fra testingen av hindringsunngåelsesalgoritmene var det klart at Algoritme 2 oftest ledet roboten til målet. Imidlertid var det noen tilfeller hvor roboten ikke klarte å navigere rundt hindringer og nå målet. I tillegg er algoritmen ineffektiv ettersom roboten stopper hver 20. grad for å sjekke om den har rotert nok til å begynne å bevege seg fremover, i stedet for å rotere de nødvendige gradene i én bevegelse før den kan begynne å bevege seg langs hindringen. Et annet problem er at roboten noen ganger uventet svinger frem og tilbake på samme sted.

En tredje algoritme, Algoritme 3, ble også utviklet. Algoritme 3 er en videreutvikling av Algoritme 2, men testing avslørte at den har mye uventet oppførsel som får roboten til å krasje i situasjoner der den normalt ikke ville gjort det og generelt oppfører seg uforutsigbart. Algoritme 3 forventes å være mer effektiv enn Algoritme 2 hvis disse problemene blir løst.

For fremtidig arbeid bør Versjon 2 forbedres ytterligere for bedre kollisjonsunngåelse. Alternativt kan et helt nytt kollisjonsunngåelsessystem implementeres ved hjelp av andre typer sensorer. For hindringsunngåelsesalgoritmene bør enten Algoritme 2 videreutvikles eller Algoritme 3 fikses før videre utvikling. Uansett hvilken løsning som velges, bør algoritmen forbedres for å øke effektiviteten.

Table of Contents

1

Introduction and Previous Work

1.1 Description of the SLAM-robot project

The majority of the following part (1.1 Description of the SLAM-robot project) is directly taken from the specialization project that was written as a build-up to this master's thesis [Eidsnes (2023)]. Only a few adjustments were made.

The SLAM-robot project by Tor Onshus has been going on for many years and many students have done their specialization project and master thesis on this project. The project consists of several robots and six of them have been used, more or less, in this project. The different robots will be referred to as DK 1 to 6, named after the nRF52840 DK development board used in all robots, throughout this project. The oldest robots (DK1, DK2 and DK3) have plastic chassis while the newer generation of the robots has a metal chassis (DK4, DK5 and DK6), but except from that they are approximate copies of the older ones. The robots communicate with a server through the network protocol MQTT. The goal of this SLAM robot project is to have several cooperating autonomous ground robots and drones for localization and mapping of unknown areas. The ground robots are built in two layers for housing all the components. It has two wheels and additional ball casters in front and in the back which keeps the robot balanced while still allowing movement in all horizontal directions. One of the robots used throughout the project is shown in Figure 1.1. The rest of this introduction will describe the SLAM-robot project at a detail-level that covers what is needed to understand the upcoming parts of the report.

1.1.1 Hardware

The main components of the robots are marked in Figure 1.1. The lower layer of the robot mainly contains two batteries and motors for each of the two wheels. The upper layer contains a motor driver, a nrf development board, a hardware shield that provides additional connectors to components and a rotating sensor tower.

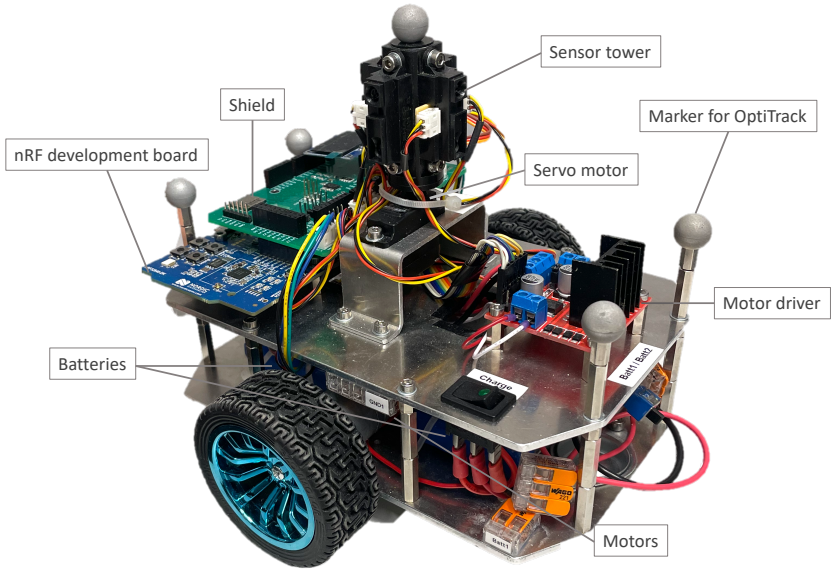


Figure 1.1: One of the robots in the SLAM-robot project.

The nRF52840 DK is a development kit from Nordic Semiconductor and is the brain of the robot. All changes in the robot code are pushed to this board and trough the connectors on the hardware shield stacked on top, it controls all functionality of the robot.

The sensor tower contains four infrared (IR) distance sensors pointing in separate directions, 90 degrees from each other. The sensor tower is mounted on a servo motor allowing the sensor tower to turn around its own axis. SLAM algorithms in the server then process sensor data, from each degree in all 360 degrees around the robot, to create a map.

1.1.2 SLAM

Simultaneous localization and mapping (SLAM) consider the ability of a mobile robot, when placed in an unknown area, to build a consistent map of the environment and simultaneously localize itself within the map. [Khairuddin et al. (2015)] The robots in this project use the sensor data from the four infrared sensors on the sensor tower. The server runs the sensor data through SLAM algorithms and creates a map of the area as it moves thorough.

1.1.3 Server

The robot receives targets (positional coordinates) from the server and then attempts to reach this target. That is, from the server one can give the robot a position (target) where the robot is desired to go to. Additionally, the server is responsible for creating a map to visualize the area which it is located in. Prior to the start of this project, several servers has

been created. One written in Java, one in C++, one simpler server written in Python and a final one in Golang. The one written in Python does not contain mapping functionalities which means it can only be used to give the robot new targets. The C++ server was written to replace the Java server because C++ is a more used programming language among the students participating in this SLAM robot project. Finally, another student working on the SLAM robot project created a new server as his specialization project. This server is written in Golang and requires less installations than the C++ server. Additionally, it can run from macOS in addition to Windows without any hassle. Since the Golang server contains the same functionality as the C++ server and is easier to install and use, it was decided to use the Golang server in this master's thesis. To learn more about the Golang server read [Klose (2023)]. As mentioned, the robots send the IR-data to the server and the server uses this data to create a map.

1.1.4 Communication between robot and server

For communication between the server and the robots, the network protocol MQTT is used. The MQTT protocol is built up of two types of network entities, a message broker and clients. The clients subscribe to the specific topics that they want to receive messages from. The clients publish messages to the MQTT broker. The broker forwards the messages to the clients subscribing to the specific topic. [EMQX-Team (2023)] This means that the broker only provides the robots and the server, which are the clients, messages from topics relevant for them. A Raspberry Pi serves as the MQTT broker. A nRF USB Dongle is connected to the Raspberry Pi which allows communication between the nRF board (robot, client) and the Raspberry Pi (broker) through Thread networking. The server communicates with the broker by connecting to the Raspberry PI WiFi. Figure 1.2 shows two constructed examples of how the clients and the broker communicate using topics in MQTT.

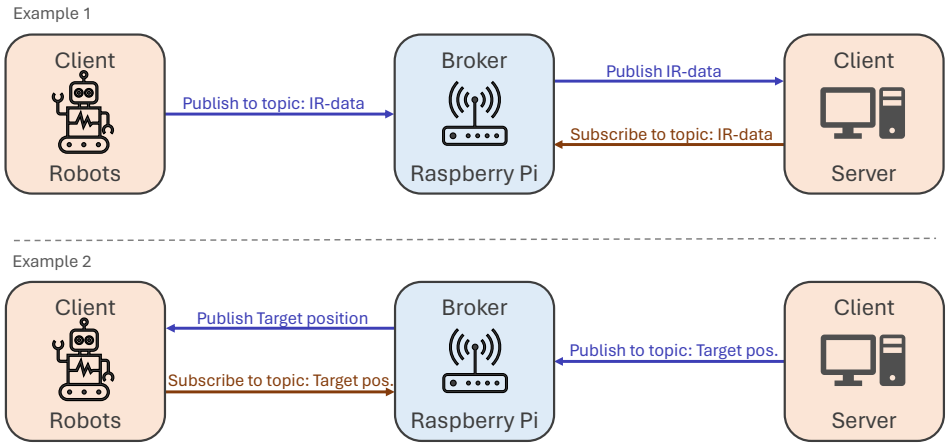


Figure 1.2: Two constructed examples of how the clients and the broker communicates using topics in MQTT.

1.1.5 Robot software and functionality

The robot software is written in C and FreeRTOS is used to run multiple tasks (processes) concurrently. FreeRTOS is an open-source real time operating system made for microcontrollers. FreeRTOS has been adapted to work with more than 40 microcontroller architectures, including the architecture of the nRF52840 DK used in this project [Fre], which allows for real time multi-threading of the task in the robot code. The different tasks with their functionalities are as follows:

- **Position estimation task:** The position estimation task is responsible for estimating the position of the robot in real time. It does so using motor encoders combined with an IMU.
- **Position controller task:** The position controller task is responsible for running a PID controller for each motor and setting a speed reference. The task uses the position estimated in the position estimation task and the target (where the robot is determined to go) to calculate the speed reference for the motors.
- **Motor speed controller task:** The motor speed controller task is responsible for controlling the speed of the motors and does so by using information from the motor encoders in a PID controller to keep the movement of the robot smooth.
- **Sensor tower task:** The sensor tower task is responsible for controlling the position and the sensor readings from the sensor tower. As mentioned earlier, the sensor tower is mounted on a servo motor. At the start of this thesis, the sensor tower operated in the following manner: When the robot was stationary, the tower would rotate one degree at a time, back and forth between 0 and 90 degrees. In Figure 1.3 the four sensors at the sensor tower are numbered. By following sensor one, it is possible to see that sensor one goes from pointing straight forward (0 degrees) to pointing straight to the left (90 degrees) before is starts tuning back. The rotating sensor tower provides the mapping task with IR data from all 360 degrees around the robot. The behavior of the sensor tower was modified during the course of this thesis.

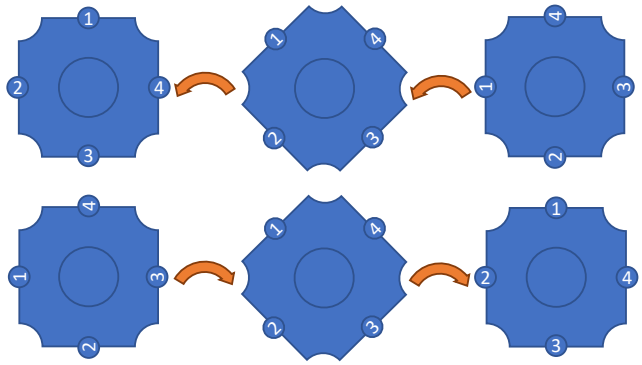


Figure 1.3: The sensor tower is turning back and forth between 0 and 90 degrees.

- **Mapping task:** The mapping task is responsible for creating line segments using the IR data and publishing it for the server to use it to create the map of the environment.

1.2 Previous work: Specialization project

Prior to this master thesis, a specialization project on the same topic, in the same SLAM robot project, was carried out. In the project a collision avoidance version was developed and the report shows the resulting behaviour and addresses challenges with this version. Additionally, two algorithms for obstacle avoidance were designed and discussed (not implemented or tested). This specialization project is the starting point of this master thesis and is briefly described in the following parts. For more details read [Eidsnes (2023)].

1.2.1 Collision avoidance

Prior to the specialization project, the SLAM robots were programmed such that the sensor tower was only spinning while the robot was standing still. When the robot was driving the sensor tower was standing still with one designated sensor pointing straight forward. The idea was to have the sensor pointing forward, detecting upcoming obstacles. This idea was further developed and tested throughout the specialization project, and is the initial collision avoidance version in this master's thesis.

From the specialization project it is clear that this version is limited by the sensor tower standing still while driving. The IR sensors on the sensor tower has a small angular range, meaning that the sensor only detects obstacles straight in front of them. This means that, because of the sensor tower standing still with one sensor pointing forward, only obstacles straight in front of the robot would be detected while driving.

Another problem with the initial version are the cases illustrated in Figure 1.4. This is a result of the sensor tower spinning while the robot standing still and checking for obstacles immediately after receiving a new target. That is, the sensor tower is always spinning while the robot is standing still and as soon as it receives a new target, the sensor tower turns into driving position (one designated sensor pointing forward). Additionally, as soon as the target is received, the robot checks for obstacles. The time of this software process is way shorter than the time it takes for the sensor tower to physically move into position. This results in the two cases:

1. Case 1: The robot has an obstacle at its left and the detecting sensor is (in this moment) pointing to the left when the robot receives a new target. The robot immediately goes into driving state and begins checking for obstacles. This happens way faster than the time it takes for sensor tower to go into position, which results in the robot interpreting the obstacle at its left as an obstacle in the collision sector. The robot therefor refuses to move forward.

2. Case 2: The robot has an obstacle at in front and the detecting sensor is (in this moment) pointing to the left when the robot receives a new target. The robot immediately goes into driving state and begins checking for obstacles. Since the detecting

sensor does not detect any obstacles (since it is pointing to the left) the robot starts moving forward and crashes into the obstacle.

These two cases are illustrated in Figure 1.4. Read [Eidsnes (2023)] for further explanation.

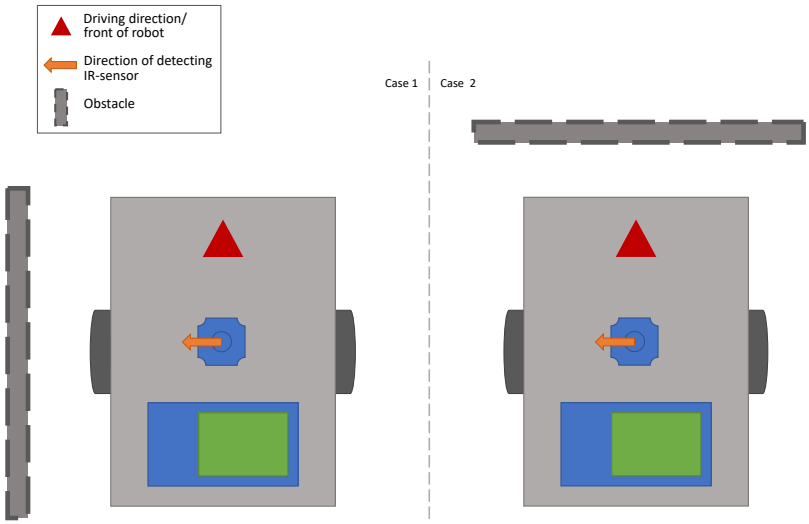


Figure 1.4: In case 1 the detecting IR-sensor is pointing to the left towards an obstacle and the robot is therefor told that it will crash if it starts moving forward even though it has free lane ahead. In case 2 the detecting IR-sensor is pointing to the left where there are no obstacles, and the robot is therefor told that it has free lane even though it does not. *Figure and caption is retrieved from [Eidsnes (2023)].*

The conclusion from the Specialization project [Eidsnes (2023)] was as follows: For the robot to be able to also detect obstacles that are not straight in front of the robot/the detecting sensor, and additionally to avoid the two cases described above, the sensor tower needs to turn constantly. The information about the surroundings should be stored such that the collision avoidance does not depend on only the current sensor readings which are limited to a very narrow area.

1.2.2 Obstacle avoidance algorithms

In addition to development and testing of the collision avoidance method, two algorithms for obstacle avoidance were designed. The first algorithm, Algorithm 1, is shown in Figure 1.5. This algorithm is inspired by [Baras et al. (2019)]. In this paper Baras et al. uses a simple decision algorithm for the robot to decide when to stop and which direction to turn when approaching an obstacle. The algorithm presented in Figure 1.5 is similar, but does not chose whether to turn left or right, it always turns left. The robot turns 90 degrees left if it approaches an obstacle and moves in this direction until there are no obstacle in the

direction of the target, then the robot turns and moves in the direction of the target until it reaches the target or it approaches a new obstacle. To sum up; after the robot approaches an obstacle and has turned to the left and started moving, it will always either turn left, if it meets a new obstacle, or if there are no obstacles visible in the direction of the target, it will turn towards the target. Otherwise it will continue to drive straight forward. (*The figure retrieved directly from the specialization project contained errors/shortcomings: it did not show what happens if the robot approaches a new obstacle after it has turned 90 degrees to the left. Therefor Figure 1.5 is a revised version of the original figure. This revised figure corresponds more correctly to the description of the algorithm and the expected path determined in the specialization project.*)

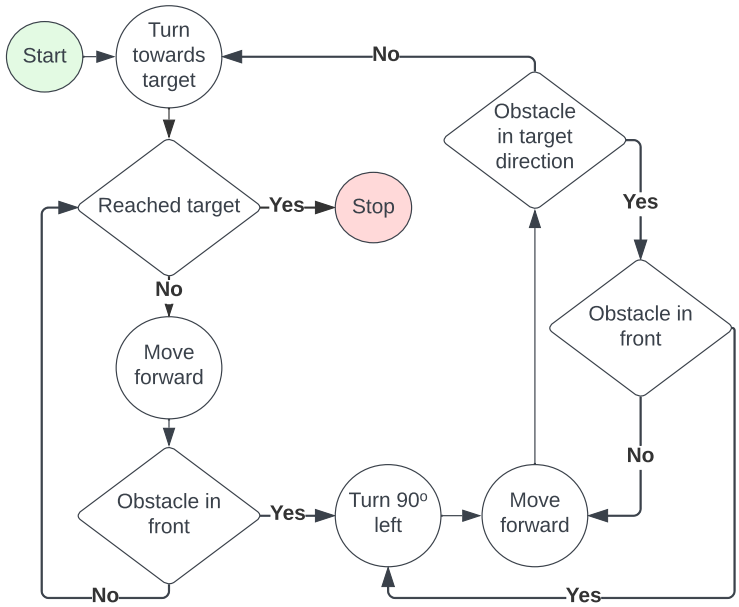


Figure 1.5: Obstacle avoidance algorithm. Algorithm 1. *Retrieved from Specialization project, but an error is resolved. [Eidsnes (2023)]*

The path of the robot is expected to be as shown in Figure 1.6 after implementing Algorithm 1. For some obstacles, this simple algorithm is expected to work, but in other cases not. Case 2 shows an example of a scenario where this algorithm is expected to not lead the robot to the target. The robot drives towards the target until it meets the obstacle, it then turns to the left and moves forward until it meets the other wall of the obstacle. It then turns to its left again and starts moving. After moving for a while in this direction the robot no longer detects any obstacles in the target direction and therefor turns towards the target. The robot will then meet the target again and will be stuck in the corner like case 2 in Figure 1.6 shows.

Because of the shortcoming revealed in case 2 in Figure 1.6, another, more complex algo-

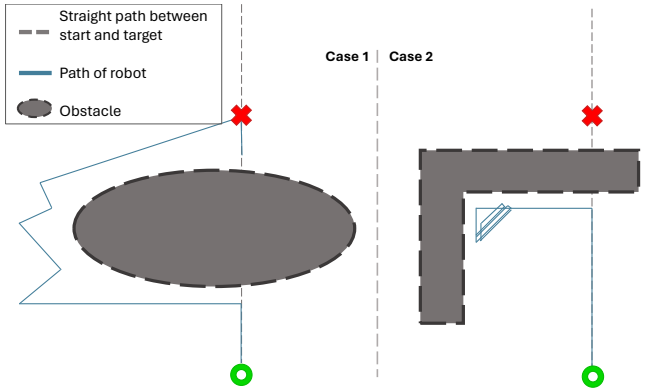


Figure 1.6: Expected path for Algorithm 1.

rithm was designed. This algorithm, Algorithm 2, is shown in Figure 1.7. As the figure shows, the robot moves towards the target until it either reaches it or approaches an obstacle. If it approaches an obstacle, it turns 90 degrees either to the right or to the left and continues to move in the new direction until it either approaches a new obstacle or there are no obstacle 90 degrees back in the direction it drove before it approached the obstacle. For example, if the robot approaches an obstacle and it is possible to turn left, it turns 90 degrees left. Then it starts moving forward and does so until the robot either has no obstacle on its right or until it approaches a new obstacle. In the case where it drives forward until there are no obstacle on the robot's right, the robot turns right and once again keeps moving in this direction until it either approaches an obstacle or has a free lane to its right. On the other hand, if it approaches a new obstacle (or the same, but at a different place) it turns to the left. The robot moves in this pattern until it reaches the straight path between the starting point and the target, but now further away from the starting point than when it first approached the obstacle. This behaviour is visualized in Figure 1.8 which shows the expected path of the robot with Algorithm 2 implemented.

Remember, the algorithms were not implemented or tested in the specialization project and Figure 1.6 and 1.8 only shows the expected behaviour and not actual robot behavior.

The results and conclusion from the specialization project makes the starting point for this master thesis.

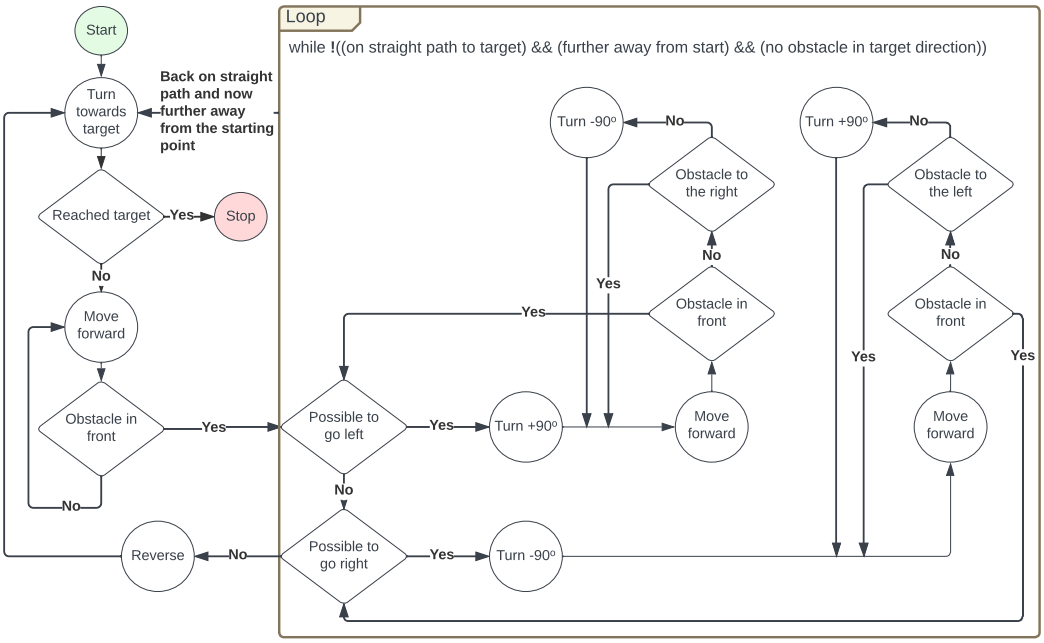


Figure 1.7: Algorithm 2. *Retrieved from Specialization project [Eidsnes (2023)]*

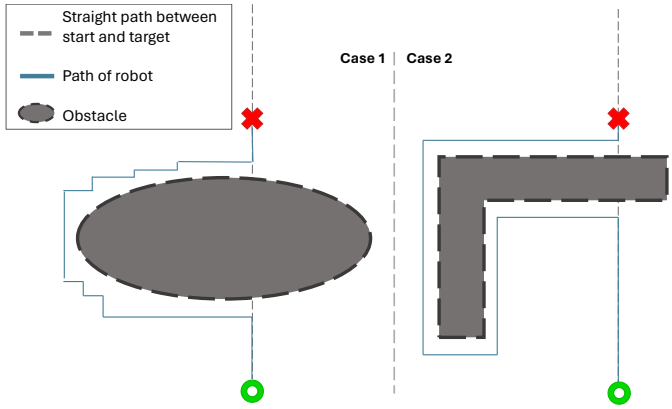


Figure 1.8: Expected path for Algorithm 2.

2

Problem Description

From the work carried out in the specialization project described in Chapter 1.2 it was concluded that the collision avoidance of the SLAM robots needed further improvements. In addition there is a wish for the robots to be able to drive around an obstacle to reach a target given behind it. Furthermore, there was a need for a simple test program to verify the robot's functionality after repairs. Therefore, the problem description for this master's thesis is as follows:

- **Clean up and restructure the files relevant for collision and obstacle avoidance in the robot code to make implementation easier.**
- **Create a simple test program for use when the robots are repaired from damage at a workshop. The test program should be easy to manage for people with limited knowledge about the SLAM robot project and should give an indication on whether the robot is functioning or not.**
- **Improve the SLAM robots' collision avoidance functionality such that the robot stops when it approaches obstacles of different shapes and sizes.**
- **Implement the two algorithms designed in the specialization project. Use the results to further improve the obstacle avoidance functionality.**
- **Merge work done on the SLAM robot project by different students and gather the newest and relevant software for the SLAM robot project such that it is easier to navigate through the project for coming students.**

3

Project Overview and Chapter Outline

To give the reader an overview and to make it easier to follow along when reading the report, this overview has been created. Below follow a short summary of the work done during this master's thesis and then an overview of the coming chapters.

3.1 Project overview: Short summary of what has been done

1. Code in relevant files has been cleaned up and restructured.
2. A test program for use after hardware changes/reparations, has been created.
3. Improvement of collision avoidance by developing two new versions.
4. Obstacle avoidance algorithms designed in the specialization project has been adjusted/revised.
5. These two obstacle avoidance algorithms has been implemented in addition to development and implementation of a third algorithm.
6. Tests of the initial and the two new collision avoidance versions has been conducted.
7. All three obstacle avoidance algorithms has been tested.
8. The resulting robot behaviour has ben visualized in plots.
9. The work done has been merged with the work done of other students working on the SLAM-robot project.
10. The work done has been documented by writing this report.

3.2 Chapter outline for the rest of the report

- **Chapter 4 Method and Tools:**
  This chapter describes the execution of this master's thesis and details how the results are carried out. The software tools used in this master's thesis, both for implementing functionalities in the SLAM robots, but also to present the work in the report, are described. In addition, the test strategy used to carry out results are presented here.

- **Chapter 5 Minor Improvements:**
  The main goal for the master's thesis is to improve collision avoidance and implement obstacle avoidance, but as described in the problem description (Chapter 2), there are some additional improvements that have been implemented. The development and implementation of these additional improvements are described in this chapter.

- **Chapter 6 Collision Avoidance:**
  This chapter contains all the documentation of one of the main goals which is to improve the collision avoidance functionality. The chapter first presents brief theory exploration of the topic, collision avoidance and examining its relevance to the project at hand. The chapter is then divided into sections separated by the three different collision avoidance versions. For each version it is further divided this way:

  - Development and Implementation:
    Describes the software development of the collision avoidance version and how these software changes is intended to give new/changed functionalities to the robots.
  - Results:
    Presents the resulting behaviour of the robots for the version, after it has been implemented. The version is systematically tested by conducting the tests described in Chapter 4. That is, the plots presented in this part shows how the robot behaves, when the version described in Development and Implementation is uploaded to the nrf-board, and the robot is undergoing the tests described in Chapter 4.

  Finally, the chapter contains a discussion part that discusses the results presented.

- **Chapter 7 Obstacle Avoidance:**
  This chapter follows the exact same structure as the previous chapter (Chapter 6). In this chapter, theory on obstacle avoidance is examined before development and implementation, results and discussion on the three different obstacle avoidance algorithms, are presented.

- **Chapter 8 Further work:**
  This final chapter discusses what can be done to further improve collision and obstacle avoidance on the SLAM robots.

4

Method and Tools

Parts of this chapter (4.1, 4.2, 4.5 and 4.6.) are mainly retrieved directly from the specialization project that was written as a build-up to this master's thesis [Eidsnes (2023)]. Some small adjustments were made and additional sentences were added.

4.1 Software changes using SEGGER

The changes in the robot code was done in SEGGER Embedded Studio which is an integrated development environment for microcontrollers using C or C++ [SEG]. SEGGER provides simple uploading of new code to the nRF board by USB connection.

4.2 Tracking robot movement using OptiTrack:

In Figure 1.1 in Chapter 1.1.1 there is an object marked "Marker for OptiTrack" in the picture of the robot. Each robot carries five of these markers and the motion capture system OptiTrack uses them to track the robots' movement. Multiple cameras positioned in a square above the floor capture data by tracking these markers. This setup accurately records the robot's position within the camera-visible area. The position data can be exported in several formats. In this project the position data is exported to a MATLAB file.

4.3 Test strategy for collision avoidance

Four tests were developed to show the resulting behavior of the robots after implementing new versions of collision avoidance. The goal was to develop different tests to reveal the robot behavior in various scenarios. For all four tests the robot was placed approximately at (x=0,y=0) and one or more obstacles were placed at approximately (x=150, y=0). The robot was given a target 150cm straight ahead. For three out of four tests, one or more boxes were placed in front of the robot. For the remanding test the obstacle was a cylinder

with a diameter of approximately 10cm. The robot was initialized between each attempt, meaning that the starting point of each attempt is (x=0, y=0) internally in the robot. This was done to make it easier to drive it 150cm straight forward. Since the precision of the robot's position estimation might drift when it is turning around, it would be challenging to give it the wanted target without initializing between each attempt. The four tests are illustrated in Figure 4.1 and described below.

- **Test 1**: The first test shows the behaviour when the robot drives "straight" towards a box.
- **Test 2**: The second test shows when the robot drives towards a angled box, meaning that the wall which the robot approaches, reaches diagonally approximately between (x=800,y=0) and (x=1200, y=60).
- **Test 3**: For the third test the robot drives toward a significantly smaller obstacle, the cylinder.
- **Test 4**: For the last test, the robot drives between two boxes. For the three first tests the goal is to test the ability of the robot to stop and hence avoid crashing into the obstacle. For the last test, on the other hand, the goal is to test the ability to ignore obstacles that are not in the robots collision sector and hence drive between the obstacles all the way to the given target position. When the robot is within a radius of 15cm around the target, the target is reached. All the attempts showed for Test 4 in the results parts in Chapter 6 are attempts where the robot had a clear path to the target.

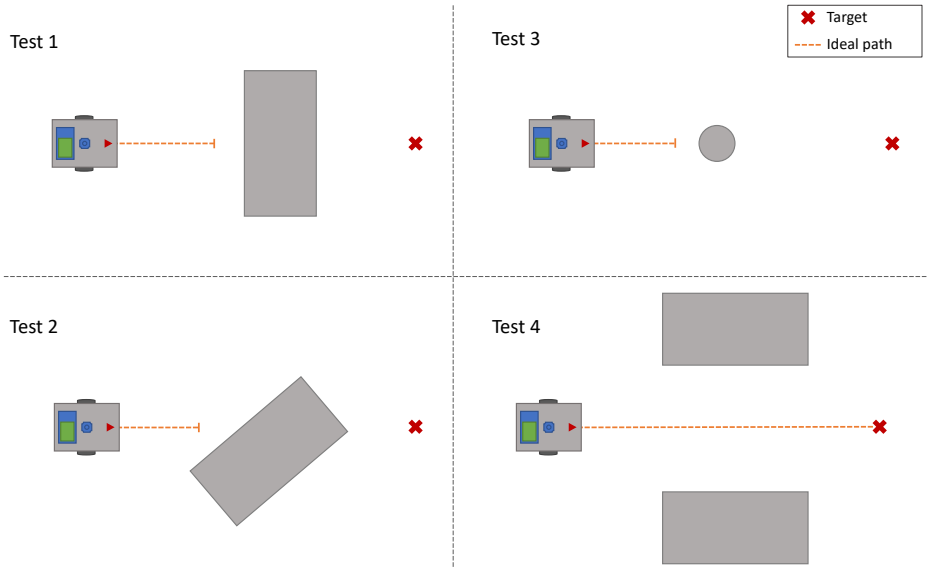


Figure 4.1: The four tests developed to test the robots' collision avoidance functionality.

All the plots presented in the results part for each collision avoidance version in Chapter 6, are from the robots undergoing the four tests described above.

4.4 Test strategy for obstacle avoidance

To test the obstacle avoidance functionality, four new tests were developed. Similarly as for collision avoidance, the goal was also here to develop tests that could reveal the robots behaviour for varying scenarios. In these four tests, the robot was positioned and initialized at approximately at (0,0), with the target set at 170-200cm straight ahead. For all four tests a successful attempt is if the robot reaches the target (15cm radius around the target). The tests are illustrated in Figure 4.2 and described below.

- **Test 1**: The first test reveals the behaviour of the robot when there is a rectangular obstacle between the starting point and the target.
- **Test 2**: The second tests reveals the behaviour of the robot when it approaches an obstacle shaped as the letter ”L”, but rotated and flipped (see Figure 4.2). The robot drives ”into the L” such that is has the obstacle both in front, but also on its left.
- **Test 3**: In the third test the robot is given a target behind two obstacles, one circular obstacle with a diameter of approximately 29cm and wall with a length of approximately 60cm.
- **Test 4**: The last test reveals the behaviour of the robot when it has to drive past and between several obstacles of different shapes and sizes. See Figure 4.2.

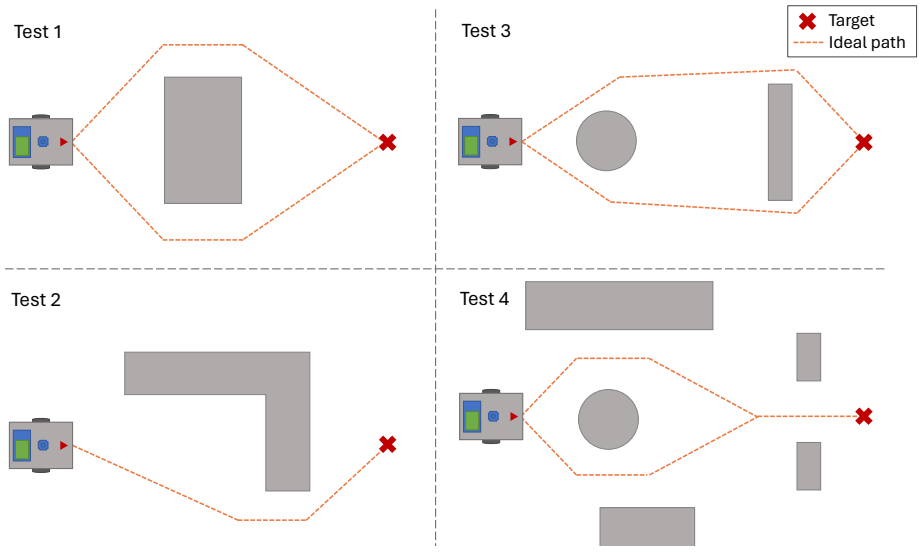


Figure 4.2: The four tests developed to test the robots' obstacle avoidance functionality.

All the plots presented in the results part for each of the algorithms in Chapter 7, are from the robots executing the four tests described above. For all the tests for both collision and obstacle avoidance, the robots named DK4, DK5 and DK6 were used.

In addition to testing the ability of the robot reaching the target, it would be interesting to evaluate the efficiency of the different algorithms. This could have been done measuring the time the robot uses to reach the target in each successful attempt, but because the robot has major problems with spinning, this would not provide any useful information. That is, the robot sometimes stops moving forward or turning around while the wheels are still spinning. By using time as a measurement of the efficiency of the algorithms would therefor depend on the robot not spinning. Therefor the efficiency of the algorithms will be evaluated by looking at the robot paths. This means that this will not be a analytic result, but will only be based on the impression from looking at the different robot paths in the plots.

As mentioned, the robot was placed approximately at (x=0,y=0) in both of the test sets. However, the placement of the robot, was purposely random inside a reasonable range around the origin and not at the exact (0,0) position. This was done to see how it behaves when it approaches the different obstacles from different angles. This means that the results from the different test can not be compared one to one, but should be viewed as an indication. For more valid results, a determined number of positions should have been chosen and for each version test the robots would have started from these determined positions. However, this would be extremely time consuming considering that the robots would have to be moved manually to the exact position between each run and additionally the OptiTrack would have to be calibrated often. Therefor it was concluded that to run the robot from random positions close to (0,0) would give sufficient results to see a path for this master's thesis.

4.5 Visualizing position data using MATLAB

The position data exported from OptiTrack is plotted using MATLAB. All figures in the results parts in both Chapter 6 and 7 with labeled x- and y-axis in mm precision, contain position data from OptiTrack and are plotted using MATLAB. Only the lines labeled "Robot path" are position data while all the other elements are added onto the plots to be able to interpret the result. The size of the obstacles might seem to change in size from one to another plot. This is partly because two different boxes of slightly different sizes are used, but the main reason is because the plots are scaled differently. Additionally, in some of these plots it looks like the robot has driven through an obstacle. This is caused by the robot moving the obstacle after crashing into it, while the obstacle is only plotted at its initial position. Note that the path shown is the middle of the front of the robot, meaning that the width of the robot is not directly visible in the plots. The robot is approximately 15cm wide and since the plot is only from a point on the front of the robot, it can be hard to see if the robot crashed or not. Therefor all plots are commented such that it is clear if the robot crashed or not, especially in cases where it is hard to understand from the plot. Note that, because the OptiTrack system has not been calibrated during the testing. Some

of the plotted trajectories were shifted, such that they did not correspond correctly to the other plots or the obstacles plotted into the figures. Because of this, a note from every attempt was written down during the testing. This way it was discovered if the plot did not correspond to the behaviour noted during testing. In such cases the plots were adjusted. This means that some of the trajectories in the plots are adjusted due to lack of calibration, which might be an error source.

4.6 Creating illustrations

Understanding the robots and their functionalities can be hard for anyone that has not been working with the system. Therefor it has been a great focus on creating illustration for the reader to better understand the issues discussed in this report. All the figures, excluding the MATLAB plots, are created using either Lucidchart, or different Microsoft Office software. Some of the figures include photos. These are captured with a mobile phone during the project.

- **Lucidchart:**
  The figures shaped as diagrams are created in the free version of the diagram application Lucidchart.
- **Microsoft Office software:**
  The figures with illustrations specific for this master's thesis are created using the Microsoft Office software, mainly PowerPoint.

4.7 ChatGPT

The free AI system, ChatGPT is used as a tool for both development and report writing. The use is described below:

- **Development**: Throughout the development process, when the robot acted unexpectedly or encountered issues like code crashes or compilation errors, ChatGPT was sometimes consulted for assistance. Sometimes, specific code snippets were provided for analysis, with a desire to get help finding the underlying problem. Other times, broader questions were given, seeking advice on debugging strategies and best practices.
- **Visualizing data**: When plotting position data using MATLAB, ChatGPT was used to help writing syntactically correct code to ensure the desired format of the plots.
- **Report writing**: After completing a chapter of the report, ChatGPT was used as a helpful tool for rewriting poorly formulated sentences or clarifying messages that were difficult to understand.

It is important to note that ChatGPT did not directly contribute to the development work. Instead, it was only used as a tool to facilitate the implementation of developed logic and functionality, as well as to assist in data visualization and report writing. ChatGPT

helped making sure code was syntactically correct for data plotting and provided assistance in refining already-written sentences for the report. However, it did not independently generate any code or text.

5

Minor Improvements

5.1 Code clean up

From the specialization project [Eidsnes (2023)] described in Chapter 1.2, it was concluded that the code needed a clean up to make it easier to develop new functionality. The code is very much characterized by the fact that many different people have worked on it over many years. Several different outdated servers and position estimation methods were still present in the code. For this master's thesis there are mainly two files relevant and therefor these were prioritized in the clean up. The clean up was mostly removing unused variables and if states that was no longer relevant. An example of unused code is a piece from the file *task_pose_controller*. Before the clean up this piece was as shown below:

```
        if (idleSendt == false) {
            NRF_LOG_INFO("controller sending idle");
            //TODO: send_idle();
            idleSendt = true;
          }

          if (USE_SPEED_CONTROLLER) {
            set_motor_speed_reference(0, 0);
          } else {
            motor_brake();
          }
          lastMovement = MOVE_STOP;
          xQueueSend(qScanStatus, & lastMovement, 0); // Send the current
    movement to the scan task
          //display_text_on_line(4,"Reached target");
        }

        /*********************************************
         * Set output
         *********************************************/
        if (PUBLISH_POSITION_CONTROLLER) {
```

```
21          double timeSinceStartup = ticksSinceStartup * 1.0 /
    configTICK_RATE_HZ;
22          controllerMsg.time = timeSinceStartup;
23          controllerMsg.x = xhat;
24          controllerMsg.y = yhat;
25          controllerMsg.theta = thetahat;
26          controllerMsg.uLeft = uLeft;
27          controllerMsg.uRight = uRight;
28          publish("v2/robot/NRF_5/controller", &controllerMsg, sizeof(
    controllerMsg), 0, 0);
29        }
30
31        if (USE_SPEED_CONTROLLER) {
32          set_motor_speed_reference(uLeft, uRight);
33        } else {
34          motor_movement_switch(uLeft, uRight);
35        }
36
```

After the clean up the resulting code was as follows:

```
1           set_motor_speed_reference(0, 0);
2           lastMovement = MOVE_STOP;
3           xQueueSend(qScanStatus, & lastMovement, 0); // Send the current
    movement to the scan task
4         }
5
6         /*********************************************
7          * Set output
8          *********************************************/
9         set_motor_speed_reference(uLeft, uRight);
10
```

As shown, this piece of code was reduced from 36 lines to 10, by just deleting lines and it did not change the functionality of the robot. Such deletions were done in the files relevant for collision and obstacle avoidance, which resulted in more readable and manageable code.

5.2 Test program

The robots used in the SLAM robot project are rather delicate, and due to their regular use by new individuals each year, they are prone to occasional damage. To prevent excessive time spent on repairs by students using the robots for their projects and theses, the Department of Engineering Cybernetics provides workshop assistance for this purpose. However, since the workshop personnel have limited knowledge about the SLAM robots, a test program was developed. The aim of this test program is to provide a straightforward code to upload to the nRF-board, which can determine if the hardware is functioning properly or not. Another crucial aspect is that this program does not necessitate a connection with the server, which means that the workshop does not have to set up the server or have the broker available to run this program. The program works as follows:

1. The sensor tower turns 90 degrees

2. The robot drives 50 cm straight forward
3. The sensor tower turns back to initial position

With this program, workshop personnel can upload it to the nRF-board after hardware repairs to verify if the robot is now operational. The program has been successfully used at the workshop several times after robot repatriations throughout the duration of this thesis.

5.3 Merging work

Other students have also contributed to the SLAM robot project. To ensure continuous progress, the work completed by different students needed to be integrated, ensuring that the project moves forward cohesively across all the directions explored in the various theses. The objective is to provide future students with a clear path to continue the work, avoiding divergent threads leading in different directions. This task involved merging changes made in the robot code, gathering improvements and implementing new functionalities into a unified codebase. Additionally, efforts were made to ensure that the Golang server functions seamlessly for both ground robots and drones.

6

Collision Avoidance

In this chapter the developed collision avoidance functionality is presented. Based on the theory presented below (6.1) three versions of collision avoidance has been developed, implemented and tested. For each of the versions, the development and implementation is described before the resulting robot behaviour is presented.

To show the resulting robot behavior from the collision avoidance implementation, four different tests are done for each version. The test method and strategy are explained in Chapter 4. For easier interpretation of the results, a score from 0 to 10 is given for each test. This means that, in all four tests, the ten attempts are either successful or failed, and the number of successful attempts gives the score. The requirements for a successful attempt depends on the test.

Remember that, as discussed in Chapter 4, when interpreting the plots in the results section for all versions, it is crucial to bear in mind that the robot is approximately 15 cm wide, while the robot path shown in the figures corresponds to a point in the middle of the front of the robot. Consequently, while the robot path in the figures may appear clear of obstacles, it is important to consider the robot's width, as it may have collided despite the seemingly clear path. Each attempt is commented to provide clarity on whether the robot crashed, especially if it's not directly visible from the figures. Additionally, there are instances where it appears as though the robot passed through an obstacle. This occurs because the robot sometimes moves the obstacle when it crashes into it, while it is not visible in the plot that the obstacle has been moved.

6.1 Theory

For unmanned automated vehicles, such as the robots in the SLAM robot project, sensors registering the environment are crucial. Different types of sensors are used for obstacle detection, including radar, ultrasonic, cameras, LIDAR and IR. Each sensor type has its own advantages and disadvantages. Radar lacks fine resolution but performs well in chal-

lenging visibility conditions, such as bad weather. Ultrasonic sensors are cost-effective and good in short-range detection. Cameras provide a comprehensive view of the environment but are limited by visibility conditions. LIDAR offers high-resolution, long-distance, 360-degree detection but struggles in harsh environments, particularly with low-reflective obstacles. [Yu and Marinov (2020)]

The robots in the SLAM robot project are equipped with a sensor tower of four infrared (IR) distance sensors, as described in Chapter 1.1. Distance measurement with the IR sensors is therefor chosen as the solution used for obstacle detection and collision avoidance in this project. The advantage of using IR sensors is that they work in the dark, provide high resolution, and are low cost. However, IR sensors are limited by their poor light tolerance and the fact that the range of the area the sensors detect is limited, creating blind spots between the sensors. [Yu and Marinov (2020)] [Ismail et al. (2016)]

For the collision avoidance versions presented in the upcoming parts the turning sensor tower is used to measure the distance between the robot and obstacles around it. The already implemented *position controller task* stops the motor when the *sensor tower task* (see Chapter 1.1.5) decides that the robot is too close to an obstacle.

6.2 Initial version

6.2.1 Implementation and development

Chapter 1.2 describes the collision avoidance version developed in the specialization project, which is the initial version in this thesis. This version has a static sensor tower while driving.That is, the sensor tower is positioned into the *driving position* with one designated sensor pointing straight forward. When this sensor detects an obstacle closer to the robot than the allowed collision threshold, the robot stops. The development and implementation of this version is described in [Eidsnes (2023)].

6.2.2 Results

To be able to evaluate the collision avoidance versions developed and implemented as part of this master's thesis, this initial collision avoidance version is tested the same way as the versions developed in this thesis. The following figures show the behaviour of the robot with the initial collision avoidance version in the four different tests.

The first figure, Figure 6.1, illustrates the resulting paths for the ten attempts in Test 1. The robot is positioned at a reasonable radius from (x=0, y=0) and given a target 150 cm straight ahead. In this test, an attempt is considered successful if the robot does not crash. Upon examining the robot paths in Figure 6.1, it's evident that the robot moves towards the obstacle and stops when it comes close to it, except for attempts 1, 4, and 10. In these instances, the robot stops early, despite being distant from the obstacle. For attempts 2, 3, 5, 7, and 9, the robot does not stop until it is very close to the obstacle. Considering the width of the robot when it approaches the obstacle in attempts 9, it is clear that the robot

was dangerously close to crashing with its front left corner. Since the sole criterion for success in this test was for the robot not to crash, the initial version is awarded a score of 10 points in Test 1.

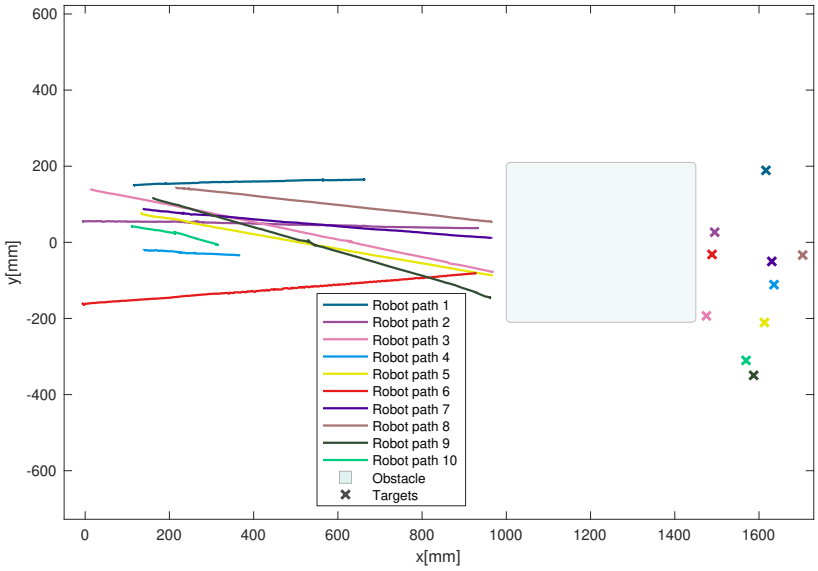


Figure 6.1: Test 1 for Initial Version: 10 attempts where the robot drives more or less head-on towards a box.

Figure 6.2 shows the paths of the robot for Test 2. In this test, the robot advances towards a diagonally placed box. Only two of the attempts resulted in the robot avoiding a collision in this scenario. In attempt 10, the robot stops after traveling approximately 30 cm instead of approaching the obstacle closely before stopping. It's noteworthy that the only attempt where the robot drives all the way up to the obstacle but stops in time is attempt 7. In this case, the robot approaches the obstacle at a corner of the box, indicating that it detects the part of the obstacle closest to it, directly in front. Consequently, the initial version is assigned a score of 2 points in Test 2.

In Test 3 the robot drives towards a cylinder with a diameter of approximately 10cm. Figure 6.3 shows the path of the robot for all 10 attempts. In this test the robot stops early in attempt 1 and stops close to the obstacle in attempt 6. Robot path 6, which is the only attempt it does not crash or stop early, shows that the robot approaches the obstacle more straight on than many of the other attempts. Similar to Test 1, the best attempt is a path where the robot approaches head-on to the part of the obstacle that is closest to the robot. For the rest of the attempts the robot crashes. The initial version receives a score of 2 points in Test 3.

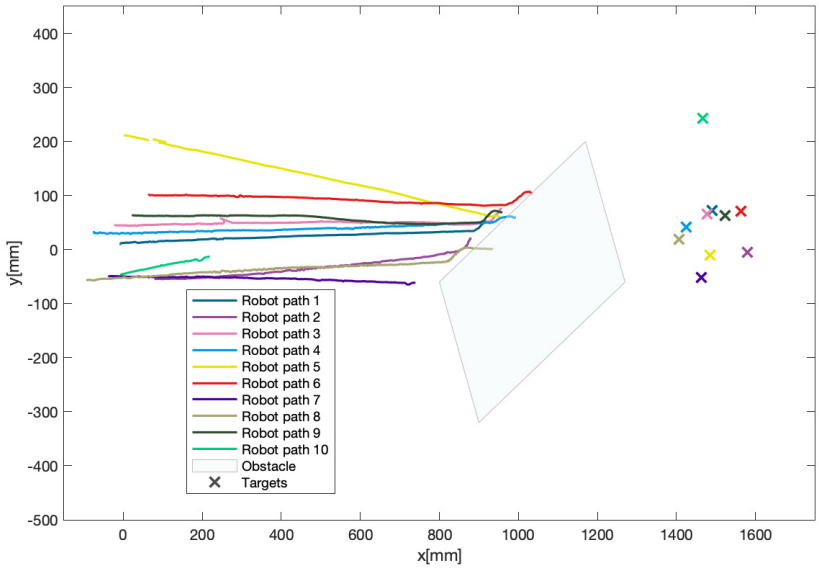


Figure 6.2: Test 2 for Initial Version: 10 attempts where the robot drives towards an angled box.

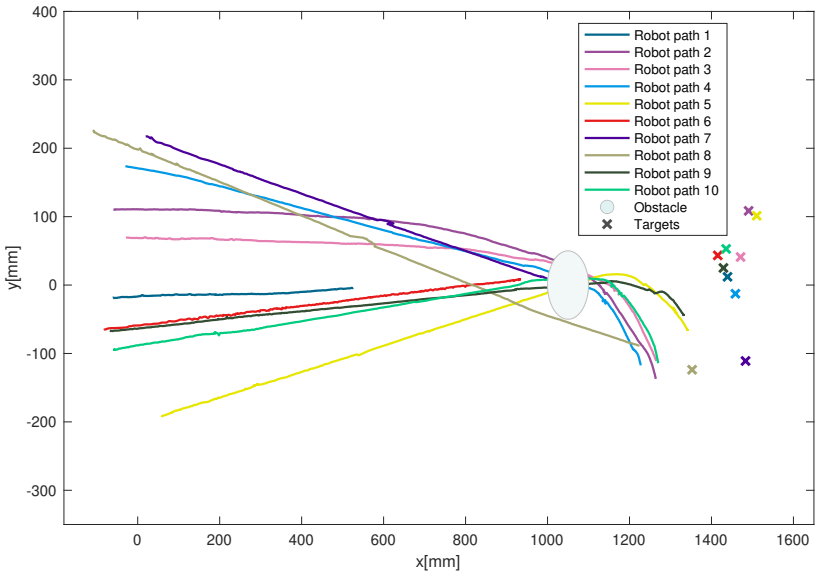


Figure 6.3: Test 3 for Initial Version: 10 attempts where the robot drives towards a small cylinder.

In the fourth and last test, the robot was to drive between two boxes. Figure 6.4 shows the paths for all 10 attempts. The criteria for success in this test was for the robot to reach the target (radius of 15cm). From the figure it is clear that most of the attempts were successful, but for attempt 4, 6 and 10 the robot stopped early. This gives the initial version a score of 7 in Test 4.

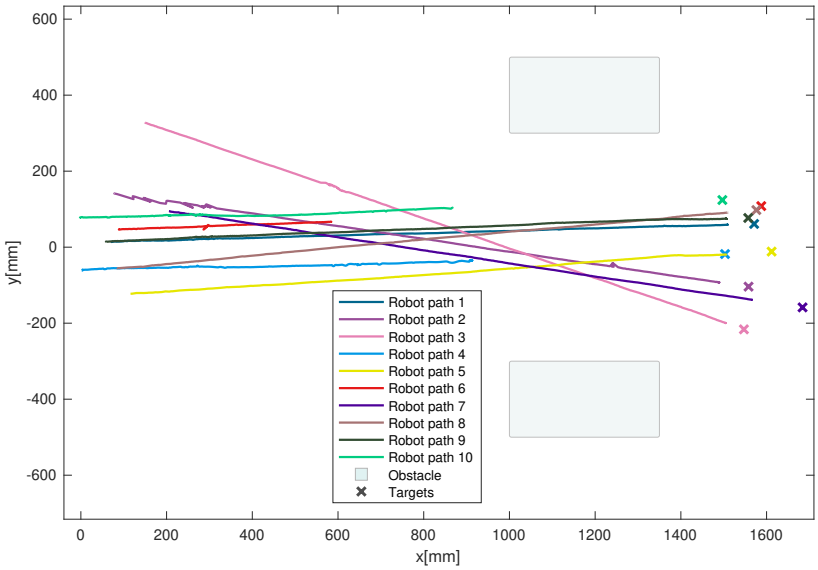


Figure 6.4: Test 4 for Initial Version: 10 attempts where the robot is to drive between two boxes.

6.3 Version 1

6.3.1 Implementation and development

As the results from the initial version shows, the robot often crashes into the obstacle when the obstacle is small or when it does not approach the obstacle straight on. It also stops early in many attempt even though there are no obstacles in front of the robot. Additionally, it was concluded in [Eidsnes (2023)] that the sensor tower needs to turn constantly to be able to improve the collision avoidance functionality. A new version was therefor developed. The main steps of the development and the functionality of collision avoidance Version 1 are described below:

1. The sensor tower was set to always turning.
2. The function checking for obstacles, *check_for_collision*, was modified so that, instead of only checking one specific sensor, it takes in the sensor number as input and checks if there is an obstacle within the the determined threshold or not.

3. An array of 360 boolean elements, as exemplified in Table 6.1, was created. The sensor tower has four sensors, each with a limited angular range, resulting in blind spots between sensors. During each iteration of the sensor tower task, the updated *check_for_collision* function is executed for all four sensors. This process is used to determine if any sensor detects an obstacle. The robot's current angle, combined with the sensor tower's current angle and the sensor number, is used to calculate the angular position of the obstacle. If an obstacle is detected, the element corresponding to that specific degree of the obstacle's position is set to true.

true_0	true_1	true_2	...	false_{359}

Table 6.1: An example of the array in Version 1 that keeps obstacle information.

4. In each iteration, the sensor tower task checks the elements in the array within a 60-degree range in front of the robot (-30 to 30 degrees relative to the robot's current position). If any of these elements in the array is set to true, it indicates that there is an obstacle in the collision sector.

5. In addition to the sensor tower task, the position controller task also plays a role in collision avoidance. When the position controller receives a new target, it first checks if the robot needs to rotate to align itself towards the target. If rotation is required, it notifies the sensor tower task that it is about to start turning. During this rotation, the sensor tower task temporarily stops registering obstacles. This is due to the robot's position estimation not being sufficiently accurate. Once the robot completes its rotation, it notifies the sensor tower task and requests confirmation on whether it is safe to start forward movement. The sensor tower task then checks the array for any obstacles in the path and communicates the result back to the position controller task. If it is not safe to proceed, the robot remains stationary and awaits a new target. Conversely, if it is safe, the position controller allows the robot to start moving forward and instructs the sensor tower task to resume monitoring for obstacles and notify the position controller task if an obstacle appears in the collision sector.

6. The sensor tower task then updates and examines the collision sector for obstacles in each iteration. If the array storing obstacle information registers an obstacle (true) at two or more elements within the collision sector, the position controller task is alerted. Subsequently, the position controller stops the robot. The necessity for multiple "true" readings within the collision sector comes from the occurrence of false positive readings by the sensors, where they detect obstacles that are not present. Without this redundancy, the robot would be prone to stopping early, as observed in the initial version, where the robot stopped early in many instances.

7. Finally, the sleep time between each iteration in the sensor tower task was reduced, resulting in a faster rotation of the sensor tower. Additionally, the maximum speed of the robot was decreased. These adjustments aimed to reduce the likelihood of the robot entering the collision area (closer to the obstacle than the chosen threshold)

while the obstacle is in a blind spot between the sensors. This issue is discussed further in Chapter 6.6.3.

6.3.2 Results

The upcoming figures show the results for the same four tests as when testing the initial version, now with Version 1 uploaded to the nrf board.

The resulting robot paths for Test 1 are shown in Figure 6.5. Attempts 7, 9, and 10 result in crashes, while for the remaining attempts, the robot avoids crashing into the obstacle. Hence, Version 1 receives a score of 7 points in Test 1.

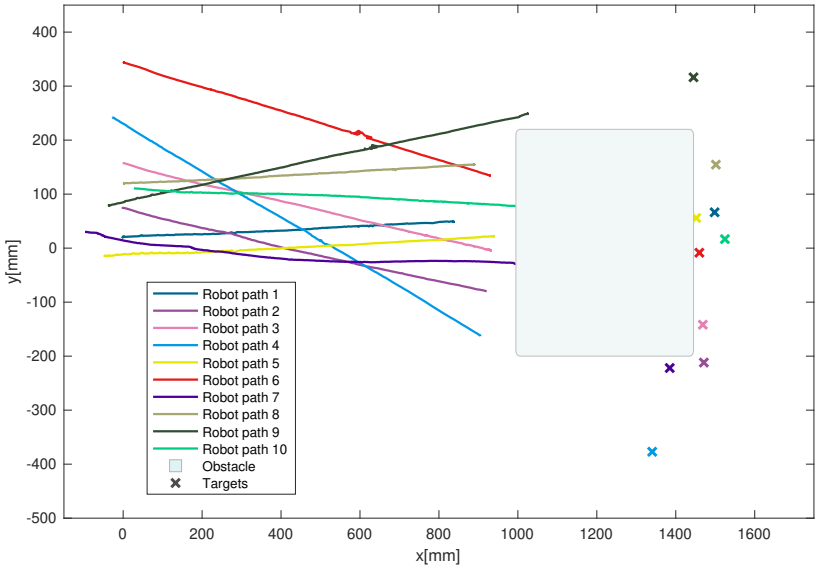


Figure 6.5: Test 1 for Version 1: 10 attempts where the robot drives more or less head-on towards a box.

The attempts in Test 2 are shown in Figure 6.6. For attempt 1, 5, 7, and 10, the robot stops successfully, while the rest of the attempts end in a crash. Note that most of the failed attempts occur when the robot approaches the obstacle at an angle, as seen in path 2, 3, 4, and 8. Conversely, the successful attempts 5 and 10 approach the obstacle almost straight on. Version 1 receives a score of 4 on Test 2.

For Test 3, 3 of the attempts end in success. Figure 6.7 shows that all robot paths except from path 1, 5 and 9, end in the robot crashing into the cylinder. On Test 3, Version 1 therefor receives a score of 2 points.

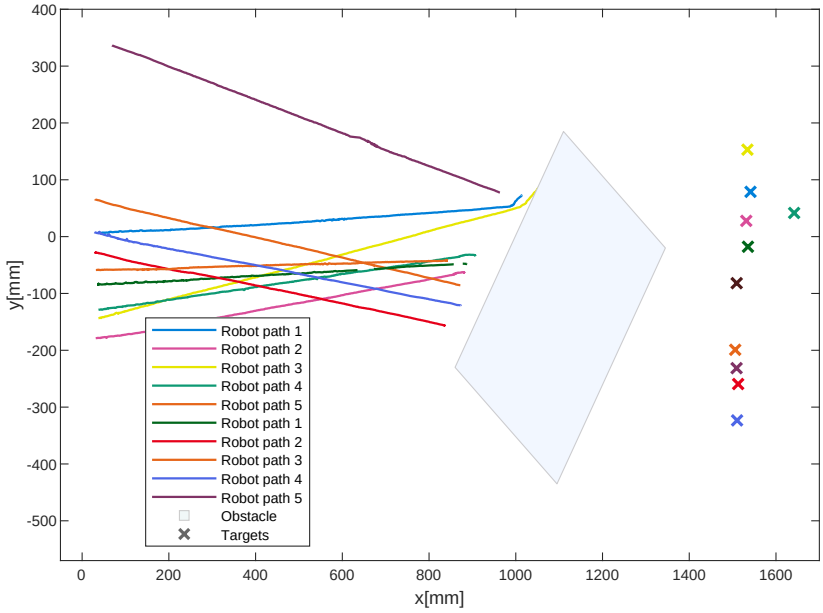


Figure 6.6: Test 2 for Version 1: 10 attempts where the robot drives towards an angled box.

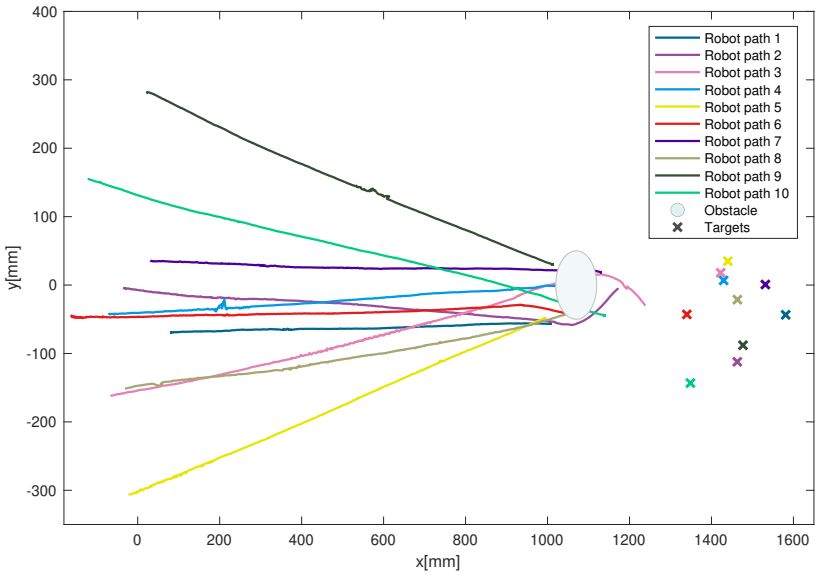


Figure 6.7: Test 3 for Version 1: 10 attempts where the robot drives towards a small cylinder.

In the last test for Version 1, the robot reaches the target for all 10 attempts. Figure 6.8 shows that all the robot paths leads within the radius of 15cm around the corresponding target. Hence, Version 1 scores 10 points in Test 4.

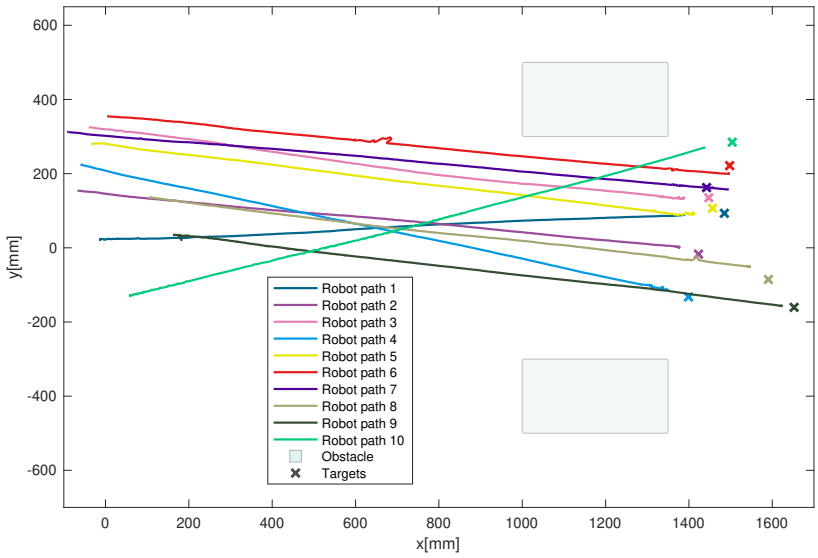


Figure 6.8: Test 4 for Version 1: 10 attempts where the robot is to drive between two boxes.

Note that none of the attempts in all four test for Version 1, results in the robot stopping early.

6.4 Version 2

6.4.1 Development and implementation

From the tests, it became evident that Version 1 had significant shortcomings. The primary issue with Version 1 is its tendency to collide with obstacles located within blind spots, where they are not visible to the sensors when they enter the collision threshold (too close). This occurs despite increasing the speed of the sensor tower and decreasing the robot's speed in an attempt to avoid this problem. Unfortunately, the delay between iterations of the sensor tower task could not be further reduced without risking software crashes. Moreover, the robot's speed was reduced until its movement was no longer smooth. Despite these adjustments, the robot still occasionally crashed due to blind spots between the sensors. As mentioned, this problem is further examined in Chapter 6.6.3 and illustrated in Figure 6.16.

It's clear that despite efforts to address the issue in Version 1, the problem is still present. Therefor a new version, depending less on the speed of the sensor tower, was developed. Below are the main steps and functionality of collision avoidance Version 2:

1. The sensor tower was set to move 2 degrees in each iteration instead of just 1. This way the turning speed of the sensor was increased while still allowing the program the required delay.

2. A new 2D array to store obstacle information was created. Instead of only keeping a true/false values in the array, the position (x,y) of the obstacle observed is stored. The new 2D array contains of 60 elements, meaning that element 0 represents 0, 2 and 4 degrees, element 1 represents 6, 8 and 10 (because the sensor tower moves 2 degrees in each iteration) and so on. Table 6.2 shows the structure of the array. The reason it does not contain 360 elements, is due to capacity challenges. When one of the sensors detects an obstacle, the sensor tower task calculates the x and y position of the obstacle using the IR distance measurement, combined with the robot position estimate.

xhatObstacle_0	yhatObstacle_0
xhatObstacle_1	yhatObstacle_1
...	...
xhatObstacle_{59}	yhatObstacle_{59}

Table 6.2: The 2D array used to store obstacle information in Version 2.

3. If there is already an obstacle position registered at the array index corresponding to the angle where the obstacle is observed, the existing reading is replaced by the new one. If no obstacles are observed for five readings in a row, the element at the index corresponding to the angle from two iterations ago is deleted. Having five false readings in a row guarantees that no obstacles are registered at the array index corresponding to the current angle minus twice the degree increment (since the sensor tower skips a degree in each iteration). To better understand this deletion process, see the illustration in Figure 6.9. The deletion mechanism ensures that incorrect readings are removed and eliminates obstacle positions that no longer correctly correspond to the robot's estimated position due to drift in the robot's position estimation.

4. Finally the redundancy from Version 1 was changed. Instead of requiring two or more of the elements within the collision sector to be within the collision threshold, for the robot to stop, it now does at least two IR-readings in each iteration (each servo motor position) and if these two are consistent (differ very little), the distance to the obstacle is calculated to be the average between these two reading. This average distance is then used to calculate the position of the obstacle. If the two readings are not consistent (significant deviation between the two readings), another IR-reading is done. New readings are done until there are two matching readings in a row.

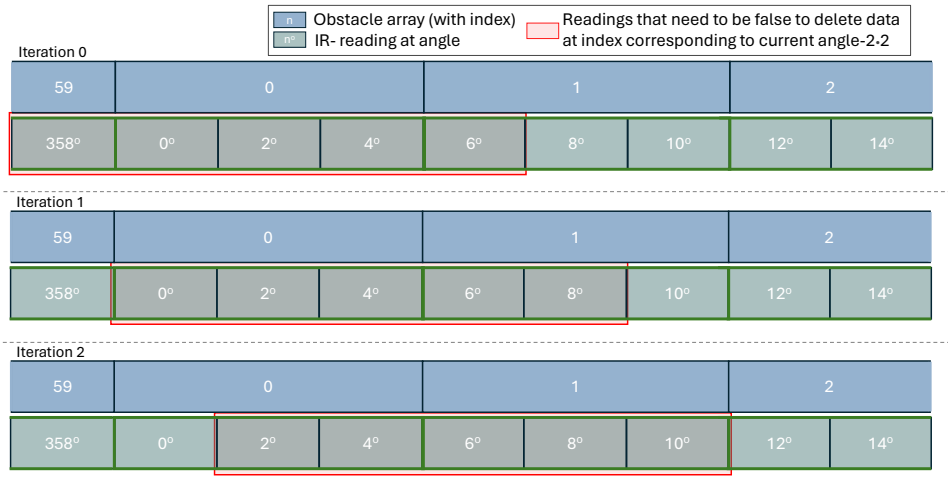


Figure 6.9: An illustration of the deletion process. The red area shows the five last IR-readings. If all of these false, it its guaranteed to be safe to delete the stored obstacle position at index corresponding to the current angle (the angle furthest to the right in the red area) minus 2. For iteration 1 it is safe to delete the data at the index corresponding to 8^{o}- 4^{o}= 4^{o}, which is index 0. For Iteration 2, the data should be deleted at index 1 if five false readings in row has occurred.

6.4.2 Results

The upcoming figures show the resulting robot paths when the four tests described in Chapter 4.3 (the same as for the previous versions) were carried out with Version 2 uploaded to the nRF-board.

Figure 6.10 shows the robot paths for the 10 attempts in Test 1. Robot path 10 shows that the robot is extremely close to crashing, but stops just in time to avoid it. Considered the width of the robot and the angle it approaches the obstacle in attempt 2 and 7, the robot is also here close to crashing with its front right corner. For all attempts except from attempt number 3, the robot avoids crashing into the obstacle. Robot path 3 shows that the robot in this case hits the corner of the obstacle with its front right corner. Hence, Version 2 receives a score of 9 points in Test 1.

For Test 2 of Version 2, two attempts end in a crash and for the rest the robot stops before hitting the obstacle. By studying the angle the robot approaching the obstacle in attempt 5, one can see that the front right corner of the robot is very close to crashing, but it stays clear. For attempt 8 on the other hand, the same corner of the robot, crashes into the top corner of the obstacle, while for attempt 4 it crashes into the wall of the obstacle. Unlike the initial version, Version 2 prevents crashes in most attempts, even when the robot approaches the obstacle at varying angles. Robot path 2, 5 and 10 are examples of the robot approaching the obstacle less orthogonal, but still manage to stop in time. Version 2 therefor scores 8 points on Test 2.

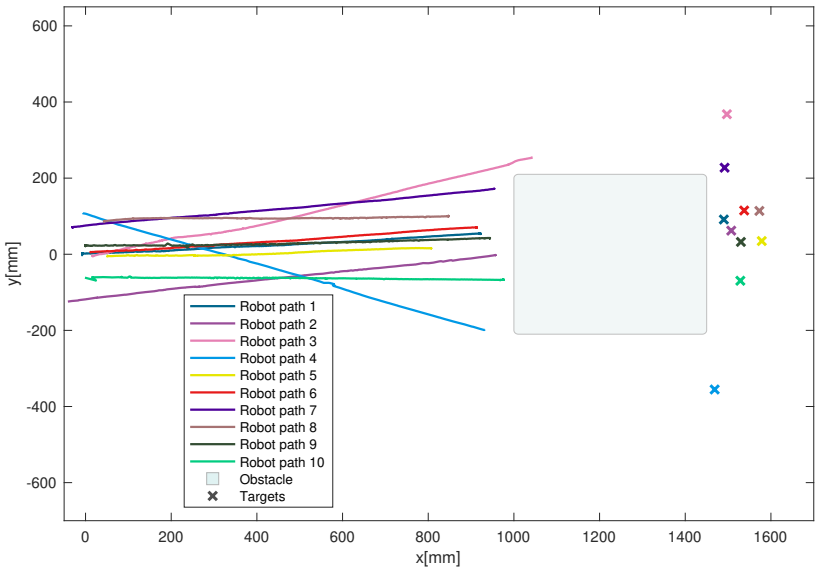


Figure 6.10: Test 1 for Version 2: 10 attempts where the robot drives more or less head-on towards a box.

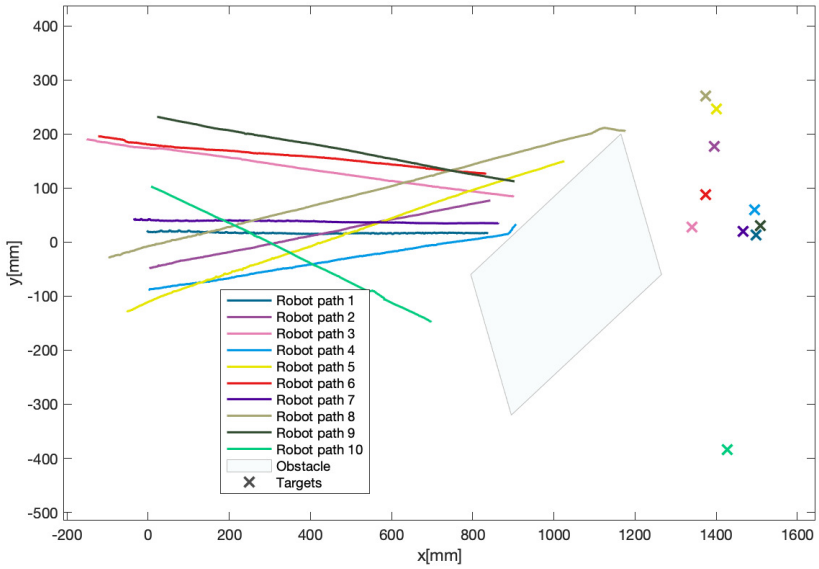


Figure 6.11: Test 2 for Version 2: 10 attempts where the robot drives towards an angled box.

Figure 6.12 shows the resulting paths for the attempts with the small cylinder, Test 3. The figure clearly shows that attempt 2, 4 and 7 end in a crash, but additionally by studying

robot path 3 it is also possible to see that this attempt also ends in failure considering the width of the robot. For the remanding attempts the robot manages to avoid crashing. It's worth noting that in several successful attempts, the robot's path directs it towards the edges of the obstacle, which contrasts with the behavior observed in the previous versions. Robot path 5, 6 and 8 are such examples. Test 3 therefor results in 7 points.

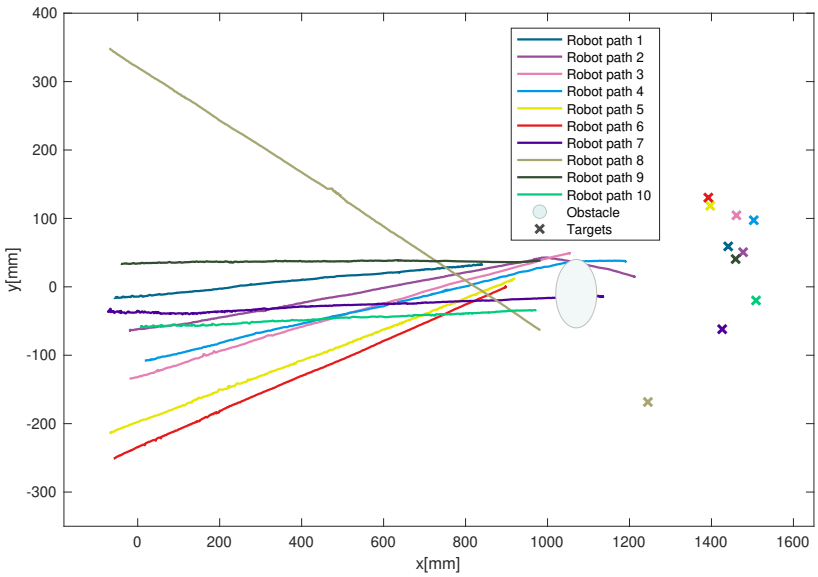


Figure 6.12: Test 3 for Version 2: 10 attempts where the robot drives towards a small cylinder.

For the final test, Test 4, all attempts except attempt 4, 5 and 7, ends in success. Figure 6.13 shows that robot path 5 and 7 ends far away from the target, while path 4 are closer to its target, but just outside the 15cm radius. The remaining paths lead all the way into the 15cm radius of their targets. Hence, Test 4 results in 7 points for Version 2.

6.5 Summary and comparison

As described, the three different versions are rated by a score from 0 to 10, in each of the four tests. Figure 6.14 shows the scores for all tests gathered in a diagram. The initial version scores the best in Test 1 with 10 points, meaning that this version did not crash into the wall orthogonal to the robots angle (approximately). On the other hand, it got the worst score in both Test 2 and 3. The robot only avoided crashing twice when approaching the diagonal wall and the small cylinder, while for Test 4 it shares the worst score with Version 2. The "winner" of Test 4 is Version 1 as it gets full score on this particular test, meaning that it ignores the obstacles outside the collision sector in all the attempts. For

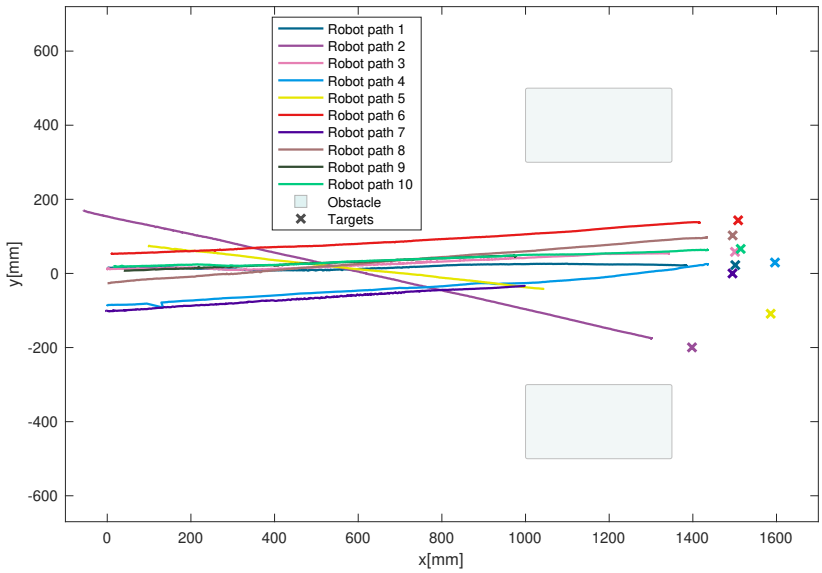


Figure 6.13: Test 4 for Version 2: 10 attempts where the robot is to drive between two boxes.

Test 2 and 3, Version 1 crashes less often than the initial version, but still more often than Version 2. Version 2 is the best at avoiding crashing into both the diagonal box and the small cylinder, but does not get the highest score for neither Test 1 or 4.

The test scores for the different versions shows that they have different strengths and weaknesses, but by adding the scores from each test together, Version 2 receives a significantly higher total score than the two previous version. Figure 6.15 shows that the initial version receives a total score of 21 points, Version 1 receives a score of 24, while Version 2 scores 30 out of 40 possible points.

In addition to the tests there are a few problems related to the different versions that were hard to generalize in a test, but still should be considered when choosing which version is the best. One of the main problems with the initial version is illustrated in Figure 1.4 in Chapter 1.2. If the sensor was pointing to the left when receiving a target straight ahead, the robot could either refuse to drive forward because of an obstacle to the left or it could drive and crash because into an obstacle in front of it because the sensor was pointing to the left where there was a free lane. This problem was resolved with Version 1 by having the tower turning constantly and saving information for all 360 degrees around the robot. Instead of relaying the one current reading from a determined sensor, it relies on readings from readings from all 360 degrees around the robot.

From the results on the tests and from the additional improvements described, Version 2 was chosen to serve as the baseline for the Obstacle avoidance development.

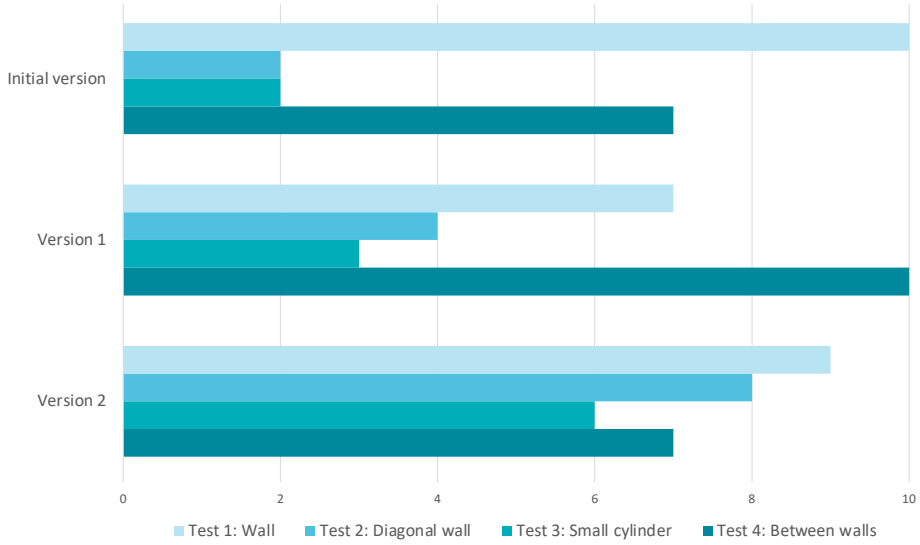


Figure 6.14: The figure shows the score for each version in all four tests.

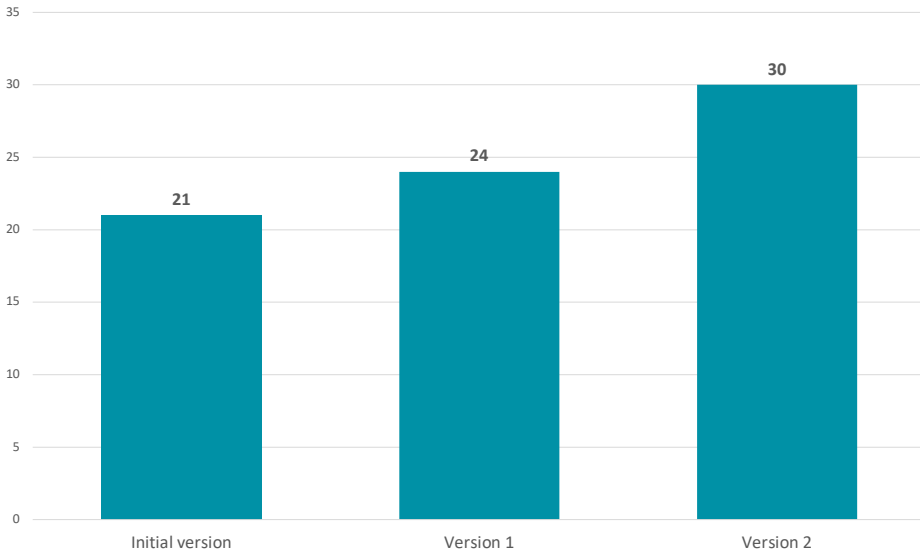


Figure 6.15: The figure shows the total score each version received throughout the testing.

6.6 Discussion

6.6.1 Error sources

As mentioned in Chapter 4 the results should be viewed as an indication instead of a final conclusion. There are several issues that prevents the results from being fully polite.

- The placement of robot during testing, as mentioned in Chapter 4, was random, but close to the origin. Some attempts might benefit from fortunate placement, giving the robot a higher chance of success if it starts from specific positions. Additionally, only 10 attempts in each test were conducted. To ensure accuracy, additional tests should be conducted. More systematic testing is necessary, with the robot starting from the same positions for each version/algorithm to validate the results.

- The real time motion capture system, OptiTrack, that was used to document the robot behaviour was never calibrated during the testing. When plotting the resulting robot paths, it was discovered that some of the plots appeared to be shifted out of the correct position. Some of the trajectories were therefor moved to fit the experienced behaviour. This is a major error source. The OptiTrack system should have been calibrated before the testing to avoid this error source. Additionally, it would have been better if also the obstacles were tracked by OptiTrack than having them plotted manually into the plots. It was hard to measure the exact position of the obstacles in the testing area, which means that the position of the obstacles in the plot does not have millimeter precision. Additionally, when the robot crashed into the obstacle, it was difficult to move the obstacle back at the exact position it had before the crash occurred.

- Precision of the robot estimate drifts off, which makes it hard to evaluate some of the attempts (this is even more relevant in the obstacle avoidance tests). To test and evaluate the collision and obstacle avoidance algorithms isolated, it would be beneficial to have the underlying robot functionality to work as desired. When that is not the case, it might be hard to know if it is the algorithm that is failing or if it is other underlying issues. This is further discussed in the upcoming parts and also in the obstacle avoidance discussion (Chapter 7.6).

6.6.2 Initial version

Starting out with the initial version developed in [Eidsnes (2023)], there is no doubt that this suffers from having a stationary sensor tower. For Test 1, where the robot drives straight towards a box, this version has no problems, but as soon as the obstacle changes into either a smaller object or that it does not approach it straight on, the initial version is in big trouble. Since the one detecting sensor is pointing straight forward at all times when the robot is moving, it cannot detect obstacles outside of the detecting area of this single sensor. This is why the robot crashes in most of the attempts in Test 3. However, in Test 2 the sensor detects the obstacle, but the measured distance is not to the point closest to the robot. This results in the robot believing that the obstacle is further away than it is at the point closest to the robot. Additionally, as illustrated in Figure 1.4 in Chapter 1.2, the

initial version sometimes behaves problematic when the robot receives a new target while the detecting sensor is pointing to the left. This might lead the robot to reject moving forward when it should move or that it starts moving forward even though there is an obstacle in front of it, this is due to the detecting sensor pointing in a different direction than the driving direction.

Another weaknesses of the initial version is that it stops when it is has a clear path towards the target. For all of the tests, except Test 3 (small cylinder), the robot has one or more cases where the robot stops without approaching obstacles. For Test 2 it does so on one attempt, while for Test 2 and 4 it does so on three of the attempts. The version is implemented such that when the detecting sensor detects an obstacle that is closer to the robot than the *collision threshold* allows, the robot stops. This means that if the sensor does one misreading, this can lead to the robot stopping even though it should not.

6.6.3 Version 1

For Version 1, the results from Test 1 was worsened. The robot only manages to stop in time in 7 out of 10 attempts. For the three attempts that ended in a crash, the problem illustrated in Figure 6.16 occurred. Since this version only stores either true or false for each degree around the robot, the robot sometimes ends up in the situation where one sensor passes the obstacle while the distance is still larger than the *collision threshold*, but after this sensor passes, the obstacle ends up in a blind spot between the sensors, and when the next sensor detects the obstacle, the robot has already passed the threshold and might crash due to that. The same problem appears in Test 2 and 3. The robot crashes into the obstacle when the robot crosses the threshold while the obstacle is in a sensor blind spot. As discussed in Chapter 6.3.1, efforts were made to avoid this issue during implementation by increasing the speed of the sensor tower and reducing the speed of the robot. The intention was to prevent the robot from being able to enter the collision threshold area while the obstacle was in the blind spot. However, as demonstrated, the problem persisted.

An alternative solution that was not attempted, is adjusting the collision threshold variable. By increasing this variable, the robot would not be allowed to approach the obstacle as closely as it currently does. Consequently, this adjustment would reduce the likelihood of the robot crashing while the obstacle is in the blind spot, as the distance to the obstacle would be greater when it enters the blind spot. However, it is advantageous for the robot to be able to maneuver close to obstacles, particularly for obstacle avoidance, but also to be able to navigate through narrow passages. This aspect will be further discussed in the discussion part on obstacle avoidance (Chapter 7.6).

On the other hand, Version 1 had some successful attempts in both Test 2 and 3, and in oppose to the initial version, some of these occurred when the robot was not approaching the obstacle straight on. Because the sensor tower is no longer static while driving, the robot is able to register more obstacles than the ones straight in front of the robot.

Additionally, for Version 1, redundancy was implemented. At least two readings within the collision sector had to be true for the robot to stop. This resulted in Version 1 getting

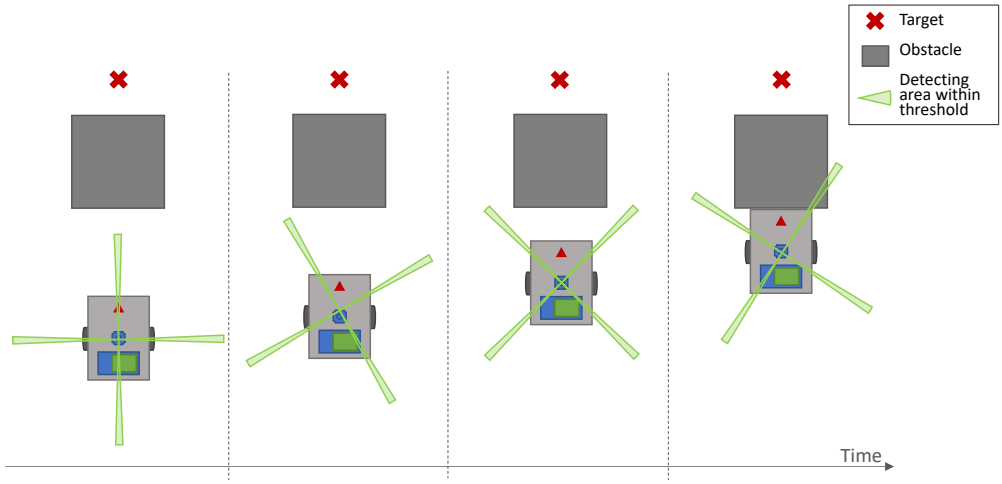


Figure 6.16: The blind spot problem. If the robot approaches the obstacle such as shown in the figure, the obstacle ends up in a blind spot as the robot surpasses the collision threshold where it should stop. Note that this is only an illustration and the dimensions in the figure are not precise.

full score in Test 4, which means that the robot reached the target for all 10 attempts when it had a free lane towards it. However, the redundancy might also have negative impact on the results in Test 3. When the robot approaches the small cylinder, the number of sensor readings of the cylinder will be limited simply by the size of it. This means that if the obstacle is only visible for the sensor at two degrees, there is no room for misreadings. The redundancy might therefor have a negative effect here.

Overall, Version 1 scores better than the initial version due to the non stationary sensor tower, but at the same time, because the tower is constantly turning, it has a problem with blind spots between each sensor which results in a number of crashes.

6.6.4 Version 2

Version 2 gets the highest total score in the collision avoidance tests. By storing the position of the obstacles registered, the robot is less prone to the blind spot problem shown in Figure 6.16. The robot is no longer dependent on the reading to be within the collision threshold to execute on it. If the IR-sensor does a reading that is outside the collision threshold and then moves such that the obstacle goes into the blind spot, the robot is able to stop in time because it now checks if the position of the obstacle is within the distance of the collision threshold from the current robot position. This has great impact on the results in Test 1, 2 and 3. Additionally, it's noteworthy that even more successful attempts (compared to Version 1) occur when the robot is not approaching the obstacle head-on. This version is the best at stopping for obstacles even when the robot's approach is not straight on.

However, Version 2 still seems to encounter the blind spot problem a few times. The reason might be that the IR-readings are less exact the longer the distance to the obstacle is. This has not been verified, but from observations this might seem to be the reason. A solution to the blind spot problem for Version 2 (that stores the position of the obstacle) and that would still allow keeping the collision threshold small, could be to have the speed controller to slow down the speed of the robot as it gets closer to an obstacle. This would increase the number of readings to verify the position of the obstacle, but it still requires the readings from further away to be roughly correct for the speed controller to know that the robot should slow down.

The initial version is still the best in Test 1 where the robot approaches a box straight on. The initial version scored 10 points while Version 2 scored 9 point in Test 1. It should be noted that the one crash for Version 2 in Test 1 is when the robot drives towards the upper corner of the box, while for the initial version non of the robot paths leads towards any of the corners. The attempt for Version 2, where the robot drives towards the corner, might be more like the attempts in Test 3 (small cylinder) since the robot can only register the obstacle to its right. The fortunate starting positions in Test 1 for the initial version, therefor might be the reason for the top score.

In Test 4, Version 1 scores better than Version 2. Version 1 has no problem reaching any of the targets when it drives between the obstacles in Test 4, while Version 2 stops before reaching the target in 3 attempts. The good results for Version 1 in Test 4 was probably a result of the redundancy implemented. Because of frequent software crashes, changes had to be done in Version 2 to be able to store the position of the obstacles instead of only boolean values. One of the changes was that the sensor tower was set to move 2 degrees in each iterations. This implies that the robot now checks for obstacles at every second degree. Additionally, the array storing obstacle information had to be decreased to only storing 60 positions instead of 360. This reduction implies that if Version 2 were to use the same redundancy method as Version 1, it would require two obstacle positions stored in the array to be within the collision sector and within the collision threshold. Since each element in the array is separated by a minimum of 6 degrees, the two positions would also be separated by 6 degrees. Consequently, the robot would never stop for an obstacle that covers less than six degrees of the sensor view. Hence, this redundancy method could no longer be used and therefor a new one was implemented. The new redundancy compares two IR-reading in the same iterations. Apparently this method led to more early stops than the redundancy in Version 1.

6.6.5 Sensor tower and nrf-capacity

In general, the sensor tower system makes collision avoidance challenging. The fact that the robot can only ”see” a fraction of the environment at a time, means that good accuracy in the sensor readings is required to avoid collisions. It needs to be guaranteed that every time one sensor does a reading inside the collision sector, there is no possibility that the robot will crash into an obstacle at this angle while this angle is in the blind spot. This is huge challenge and this requirement is not fully fulfilled in any of the collision avoidance versions presented in this report. Additionally, the nrf-board seem to have limited capacity.

When trying to store positions for all 360 degrees around the robot in Version 2, software crashes occurred frequently. Even when removing all calculations and only adding 0's as both x and y values into the array, the code seem to crash. A lot of experimenting with different solutions were done, but none of them allowed more than 60 elements in the array storing obstacle positions. There is no guarantee that these problems do not come from software/implementation errors, but from the debugging done, no implementation errors causing this issue was found. The full robot code uploaded to the nrf-board is huge and relatively unorganized. There might be unknown real-time dependencies in the code causing the problem. A thorough clean up of the robot code could possibly solve several of the problems with software crashes and might allow more advanced collision avoidance.

Another possibility to improve the collision avoidance functionality is to use a camera instead if IR-sensors. This would require a study of methods using camera to detect obstacles and a completely new implementation.

7

Obstacle Avoidance

In this chapter the developed obstacle avoidance functionality is presented. The three algorithms presented are based on the theory presented below (7.1). The same way as for the collision avoidance versions in Chapter 6, this chapter is divided such that for each obstacle avoidance algorithm, the development and implementation is described before the resulting robot behaviour is presented.

The robot behaviour for the three upcoming algorithms is revealed from the four tests illustrated in Figure 4.2 and described in Chapter 4. The resulting robot paths are shown in the upcoming results parts for each Algorithm. In oppose to the tests for collision avoidance, the robot executes each test five times instead of ten. This was decided because each test takes longer than for collision avoidance, and additionally the plots are more messy which makes it harder to follow each path if there are many paths in each plot. The three algorithms will be rated with a score from 0 to 5 depending on how many of the attempts result in success. All three algorithms are using collision avoidance Version 2 to detect obstacles, meaning that the limitations of Version 2 is still present in the testing of all three algorithms. Since the collision avoidance results are already evaluated, a successful attempt in the upcoming results is therefor only dependent on the robot reaching its target and any collisions are ignored. The robot has reached the target if it stops within a radius of 15 cm around the given target.

7.1 Theory

The subject of ”Obstacle avoidance for mobile robots” is a broad field, with many different approaches used to address the problem in various scenarios. The methods used to achieve effective obstacle avoidance depend on the robot’s hardware, software capabilities, and anatomy. The robots in the SLAM robot project and their equipment are described in Chapter 1.1.

One possible approach to obstacle avoidance is to roughly divide the field into two different techniques: path planning and reactive behavior. [Kunchev et al. (2006)]

- **Path planning** techniques uses the information available to plan a route for the robot. [Kunchev et al. (2006)]
- **Reactive behaviour**, on the other hand, use real time sensor data to make choices on the fly. [Kunchev et al. (2006)]

A path planning method requires information about the environment to work. This information is often limited. The purpose of the SLAM-robot project is to develop mobile robots that can explore an unknown area and create a map of it. Since path planning algorithms uses information about the environment, the method to be used in this project can not rely heavily on path planning techniques. Additionally, the robot is equipped with relatively simple hardware and from experience, it is know that the capacity of the nRF-board is limited. Hence, the obstacle avoidance algorithm needs to be simple enough for it to work with the robots.

For a pure reactive method, on the other hand, the robot is unlikely to reach the desired target as it does not plan where to go, but only reacts to sensor information. This is why most obstacle avoidance methods are a hybrid between path planning and reactive techniques. [Kunchev et al. (2006)]

The collision avoidance functionality (Version 2) described in the previous chapter (Chapter 6.4) serves as the foundation for the obstacle avoidance functionality. The robot reacts by stopping when it approaches an obstacle (Chapter 6.4.2 shows that it is not working in all cases). This reactive behavior is a crucial component of obstacle avoidance. The goal for the upcoming part is to implement a solution where the robot finds a way around the obstacle once it has reacted when approaching it. This entails using the environmental information it has now retrieved (i.e., the detection of the obstacle) to plan a path around it—a path planning technique.

The upcoming parts in this chapter uses the algorithm presented in Figure 1.5 (Algorithm 1) in Chapter 1.2, as a starting point. As described in Chapter 1.2 this algorithm was inspired by [Baras et al. (2019)]. [Peng et al. (2015)] employ a similar approach, where the robot's reference direction is towards the target, but when detecting obstacles, it plans a new direction based on the density of obstacles in different directions. The further development of Algorithm 1 into Algorithm 2 and Algorithm 3 is inspired by [Peng et al. (2015)] as well as [Baras et al. (2019)]. All three algorithms mainly use reactive methods but incorporate path planning to determine the direction to move and, sometimes, the distance to move after the robot has reacted to the environment (detected an obstacle).

7.2 Algorithm 1

Algorithm 1 is the first and most basic algorithm. This algorithm was first designed in [Eidsnes (2023)] which is summarized in Chapter 1.2 and further development in this section.

7.2.1 Development and implementation

The first and simplest algorithm is shown in Figure 1.5 in Chapter 1.2 where additionally, the expected behaviour is described. The robot is not expected to perform perfectly, but due to the uncertainty surrounding the capacity of the nrf-board, it was decided to implement this algorithm. This algorithm is simple which gives the robot a higher chance of handling it. The main steps of the implementation and the functionality of Algorithm 1 is listed below:

1. The robot was set to turn 90 degrees to the left when it approaches an obstacle and notify the sensor tower task that it now needs to know if the lane towards the target is free, in addition to watch out for obstacle coming up in the driving direction of the robot. The position controller uses equation 7.1 and 7.2 to give a new intermediate target.

$$xTargt = xhat + 3 \cdot cos(thetahat + \pi/2) \tag{7.1}$$
$$yTargt = yhat + 3 \cdot sin(thetahat + \pi/2) \tag{7.2}$$

2. For the sensor tower task to determine which direction to look for a free lane, it needs to know the target's position. An initial attempt was made to provide the sensor tower task directly with the target's position (x, y), but this resulted in unexpected outcomes. Therefor, the position controller now provides the sensor tower task with the angle of the target relative to the current robot angle. The sensor tower task then uses this information to calculate the target's position.

3. After the sensor tower task is notified, and has determined the position of the final target, it continuously checks for free lane towards the target and watches out for upcoming obstacles. When one of these two occurs, the position controller is notified.

4. If the notification from the sensor tower task says that there is a free lane towards the target, the robot turns towards it and moves forward until it either reaches it or approaches a new obstacle. If the robot reaches the intermediate target it will also turn towards the target and either go forward towards it or turn 90 degrees to the left if there is an obstacle in the target direction.

5. If the robot still has not reached the final target after 20 tries it stops and waits for a new target. The number tries are defined as the number of times the robot stops for an obstacle.

During the execution of the tests presented in Chapter 4.4 it was discovered that Algorithm 1 did not work as expected. After the robot had turned left when approaching an obstacle, it did not check for free lane in the target direction. Rather, the direction it checked seemed to be random each time it turned left. Since this was not discovered before the final testing, and because this was the algorithm that was expected to perform worst, it was decided to not start debugging to fix the error. Instead it was decided to make a quick change to be able to interpret the results. The robot was therefor set to check for free lane 90 degrees to its right after turning left, but still turning towards the target each time it

has a free lane on its right. Because of this change, the robots behaviour was expected to be even poorer than what is shown in Figure 1.6 in chapter 1.2. The actual algorithm implemented during the final tests and the behavior that is revealed by the plots in the upcoming results part, is the algorithm shown in Figure 7.1.

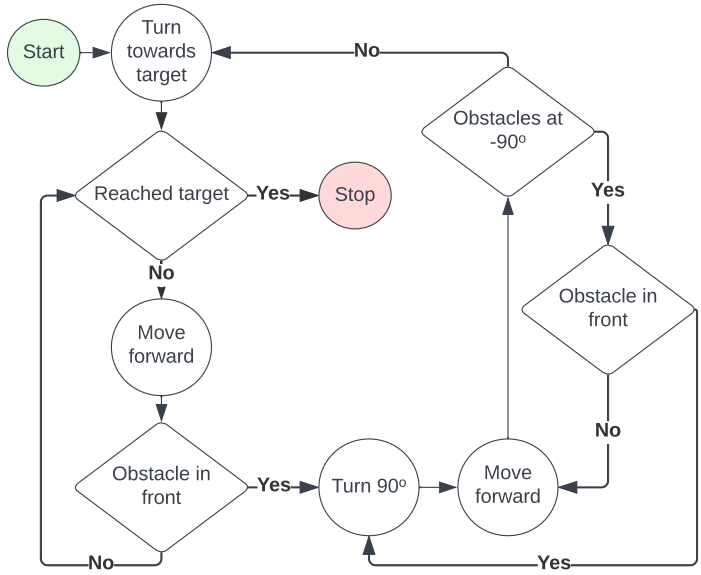


Figure 7.1: Algorithm 1. Because of an error in the implementation of the Algorithm 1 discovered during the final testing, a quick change was done. This is the actual implemented algorithm at time of testing.

7.2.2 Results

The upcoming figures show the resulting robot behaviour for Algorithm 1.

Figure 7.2 shows the resulting robot paths for the 5 attempts in Test 1. For all of the five attempts the robot stops close to the corresponding targets giving Algorithm 1 a score of 5 in Test 1. Note that for some of the attempts there are accumulations on the path. One example is on path 4. At the position where path 4 (green) crosses path 2 (pink), there are an accumulation on path 4. This pattern shows that the robot has turned 90 degrees to the left and then turned towards the target multiple times (it is oscillating) before it continued towards the target (and the obstacle). The same is visible twice on path 3 before the robot turned 90 degrees left after approaching the obstacle and also towards the end of path 5.

In Test 2, Algorithm 1 performs significantly worse than in Test 1. The paths are shown in Figure 7.3. Only for attempt 4, the robot stops within the 15 cm radius around the target. However, for attempt 2 and 3 the robot thinks it has reached the target. In these attempts,

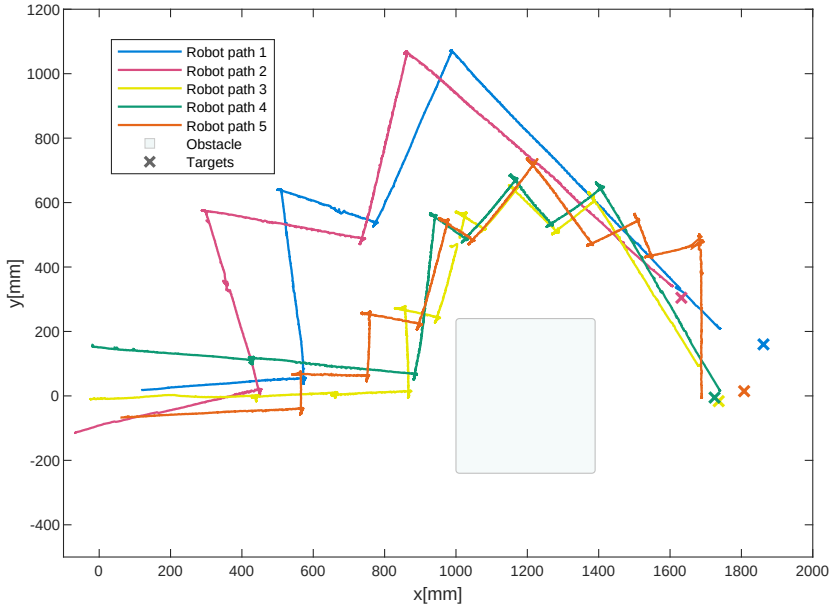


Figure 7.2: Test 1 for Algorithm 1: 5 attempts where the robot is to drive around a single box.

the server (described in Chapter 1.1.3) reported that the robot had reached the given target. When the robot is turning back and forth many times the position estimation drifts off and therefor the target position no longer correspond to the position estimation. The robot stops far away from the target in both attempt 2 and 3, while the server shows that it has reached the target. In attempt 1 and 5 the robot ends up stuck in the corner of the obstacle and stops after the maximum attempts of reaching the target, is reached. Hence, Algorithm 1 ends up with a score of 1 point in Test 2.

Figure 7.4 shows the resulting paths for Algorithm 1 in Test 3. The robot reaches the 15 cm radius around the target only in attempt 4. However, for attempt 2, 3 and 5 the robot thinks it has reached the target, such as described for Test 2. For attempt 2 and 5, the ending position is not too far off the actual target position. Half a point is therefor given in these two attempts. In attempt 1 the robot drives back and forth close by the second obstacle until it reaches the maximum number of attempts of reaching the target. Hence, Algorithm 1 ends up with a score of 2 points in Test 3.

The results for the final test for Algorithm 1 is shown in Figure 7.5. Similar behaviour as in Test 3 is revealed, but here, none of the attempts end in success. The robot drives back an forth around the obstacles and the robot stops after maximum attempts, far away from the target for all of the attempts, except from attempt 3. Path 3 shows that the robot manages to pass the obstacles, but drives too far after passing the last one. The server revealed that the robot thinks it stopped at the target in this case and since it is not too far off, half a point is given. Algorithm 1 gets 0.5 points in Test 4.

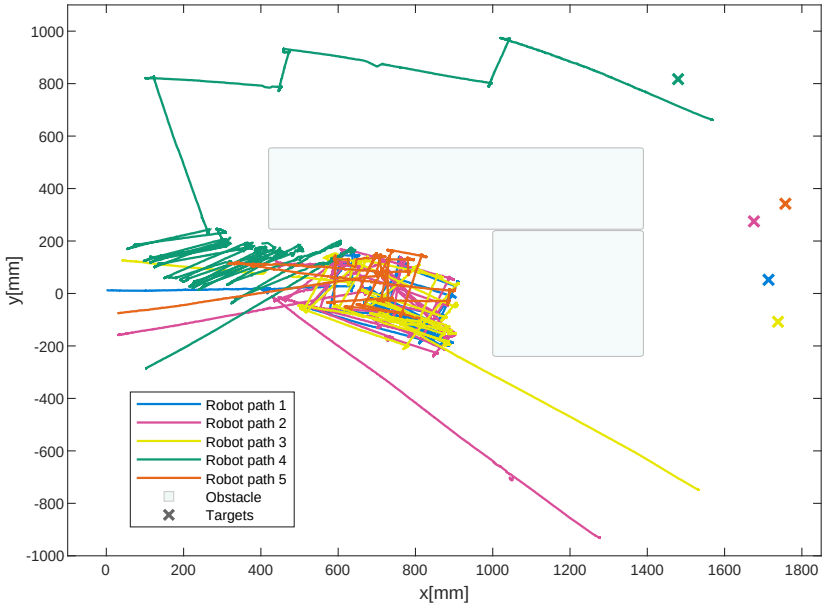


Figure 7.3: Test 2 for Algorithm 1: 5 attempts where the robot is to drive around a a ”L” shaped obstacle.

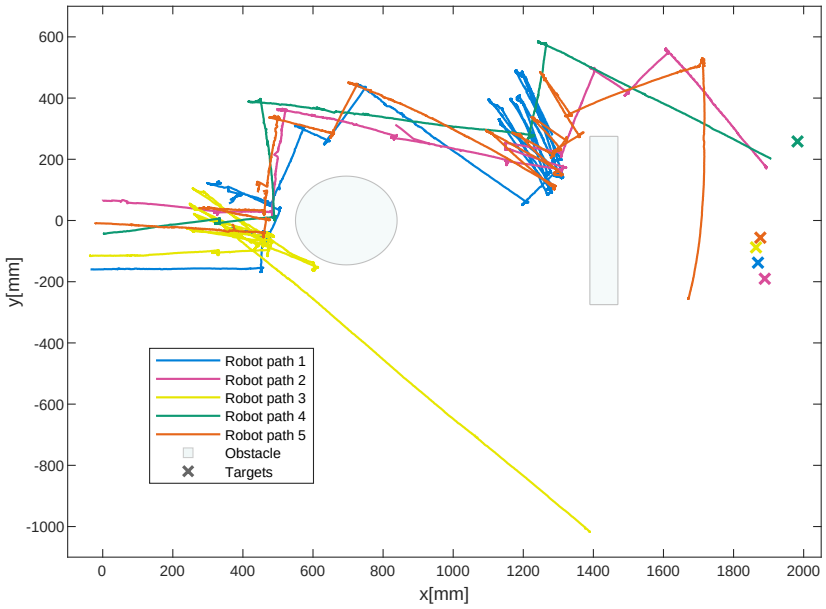


Figure 7.4: Test 3 for Algorithm 1: 5 attempts where the robot is to drive past tow obstacles, cylinder and a wall.

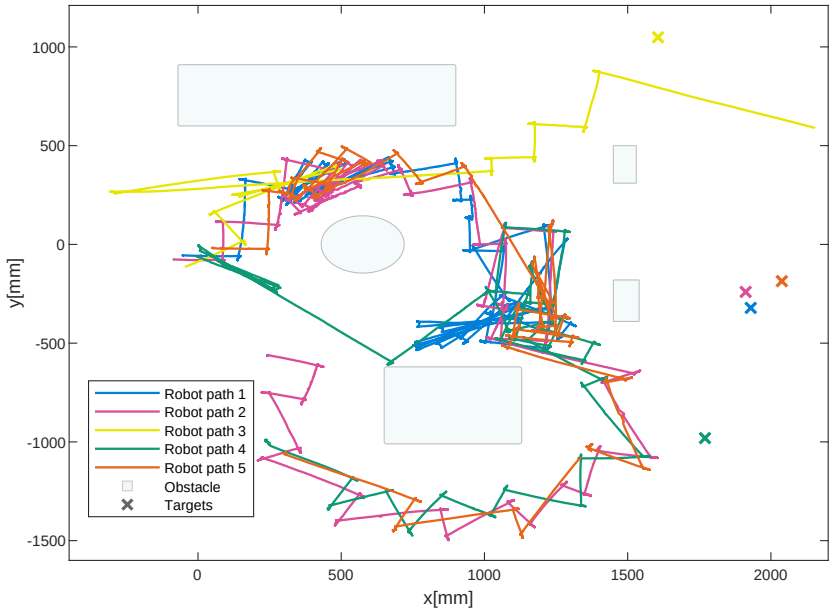


Figure 7.5: Test 4 for Algorithm 1: 5 attempts where the robot is to drive through/around a labyrinth of different obstacles.

7.3 Algorithm 2

7.3.1 Development and implementation

Because of the expected shortcoming of Algorithm 1 discussed in Chapter 1.2 and shown in Figure 1.6, Algorithm 2 was designed in the specialization project [Eidsnes (2023)]. Additionally, the results presented for Algorithm 1 reveals insufficient behaviour which advocates moving on with a new algorithm. Figure 1.7 in Chapter 1.2 shows Algorithm 2. While implementing, some adjustments were made as it seemed more convenient. Additionally, inspiration from [Peng et al. (2015)] contributed to the decision of doing changes on Algorithm 2. The adjusted Algorithm 2 is shown in Figure 7.6. Instead of first checking if it is possible to go left and if not it checks if it is possible to go right, it now turns left until it is possible to start moving forward. Additionally, the robot only turns 20 degrees at a time. That is, when the robot approaches an obstacle, it now turns 20 degrees left regardless if it it possible to move in this direction or not. This means that if it is not possible to move in this direction, it turns another 20 degrees to its left, and does so until it is possible to start moving forward. This is the implemented version and the results presented later in the results section show the robot behaviour with this algorithm implemented.

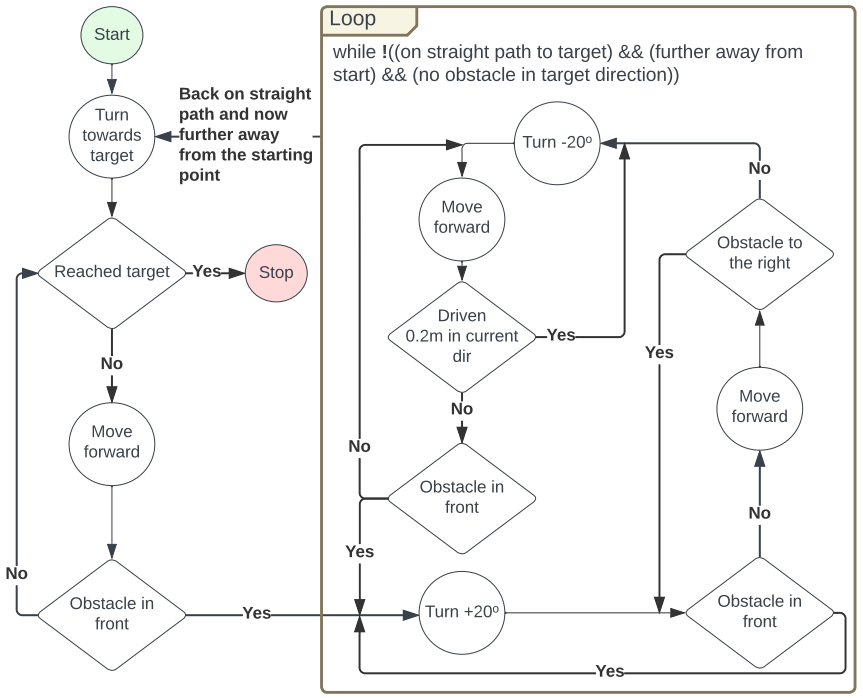


Figure 7.6: Adjusted version of Algorithm 2. Algorithm 2 designed in the specialization project [Eidsnes (2023)] and that is shown in Figure 1.7 in Chapter 1.2, was adjusted during implementation. This is the implemented Algorithm 2.

The changes/additional steps to move on from Algorithm 1 and implement Algorithm 2 is therefor:

1. Just as in Version 1, when the robot approaches an obstacle it turns left, notifies the sensor tower task and starts moving forward. However, the sensor tower task starts checking for a free lane to the right instead of in the target direction.

2. In an attempt to keep the robot driving close to the obstacle, it only turns 20 degrees instead of 90 degrees when it approaches an obstacle. The robot turns 20 degrees both when it turns left and right. This means that after entering the obstacle avoidance loop, the robot is given a new intermediate target in the direction of 20 degrees to the right when the lane is free from obstacles in this direction. Hence, the intermediate target when the robot approaches an obstacle is the equations 7.3 and 7.4, and afterwards when the lane is free 20 degrees to the right the intermediate target will be 20cm in this direction such as shown in equation 7.5 and 7.6.

$$xTargt = xhat + 3 \cdot cos(thetahat + \pi/6) \tag{7.3}$$
$$yTargt = yhat + 3 \cdot sin(thetahat + \pi/6) \tag{7.4}$$

$$xTargt = xhat + 0.2 \cdot cos(thetahat - \pi/6) \tag{7.5}$$
$$yTargt = yhat + 0.2 \cdot sin(thetahat - \pi/6) \tag{7.6}$$

3. After the robot has turned 20 degrees to the right because of a free lane in this direction, it will drive 20cm such as equation 7.5 and 7.6 implies. When the robot reaches this intermediate target, the same target will be given again. The robot will continue going 20 degrees to the right and 20cm forward until it either approaches a new obstacle or the robot is back on the straight path between the starting point and the target, but now further away from the starting point than when it first entered the obstacle avoidance loop.

4. If a line was drawn between the starting point and the target, this would be what here is referred to as ”the straight path between the start and the target”. When the robot is back on this straight path, it turns towards the final target and drives towards it.

The expected path for Algorithm 2 shown in Figure 1.8 in chapter 1.2 is still valid with the changes done in the algorithm. The figure can be useful to look at when trying to interpret Algorithm 2.

7.3.2 Results

This section presents the resulting behaviour of Algorithm 2.

Figure 7.7 shows the resulting paths for Algorithm 2 in Test 1. The robot succeeds in all the five attempts. In attempt 3 the robot crashes into the obstacle and the robot stops further away from the target than for the other attempts, which suggests that the position estimation has drifted off. However, the robot still stops within the 15 cm radius from the target. In the rest of the attempts the robot stops close to the target. Note that when the robot drives close to the obstacle before turning left, it tends to make a larger detour around the obstacle. This applies particularly to path 4. On the other hand, path 2 shows the attempt where the robot turns left furthest away from the obstacle and the path shows that this is the attempt where the robot drives closest to the obstacle. Algorithm 2 receives a score of 5 points in Test 1.

In the second test the robot manages to get around the obstacle in 4 out of 5 attempts. The resulting paths are shown in Figure 7.8. In attempt 1, the robot turns left early when approaching the obstacle and turns left again when it approaches the upper part of the obstacle. Unexpectedly it turns approximately 260 degrees, while it should have stopped turning after approximately 100 degrees. Path 1 shows that the robot then end up on the straight path towards the target, which allows the robot to turn towards the target (see Chapter 7.3.1 for explanation of this requirement). The robot approaches the obstacle again and turns left, but here robot gets stuck oscillating until it has used the maximum number of attempts. The rest of the attempts end in success. However, the robot takes some unexpected large detours around the upper left corner of the upper part of the obstacle. Algorithm 2 receives a score of 4 points in Test 2.

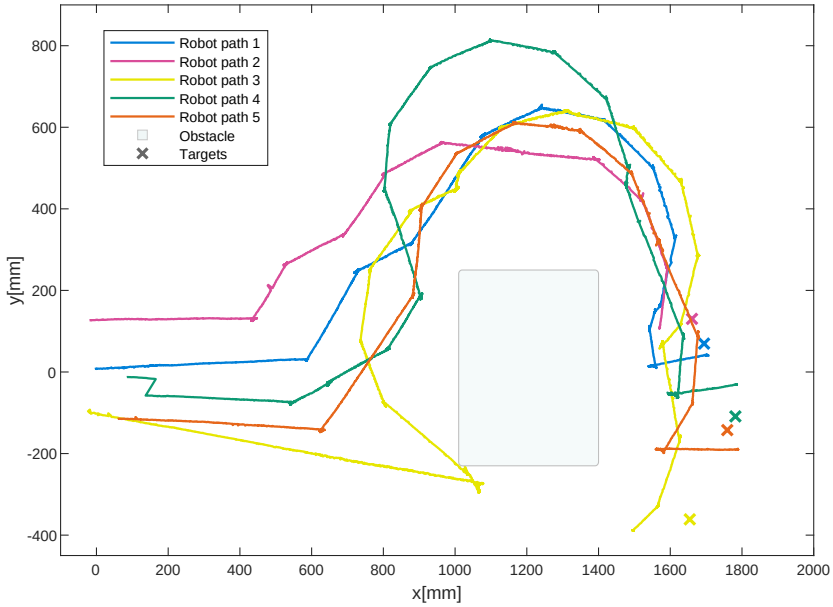


Figure 7.7: Test 1 for Algorithm 2: 5 attempts where the robot is to drive around a single box.

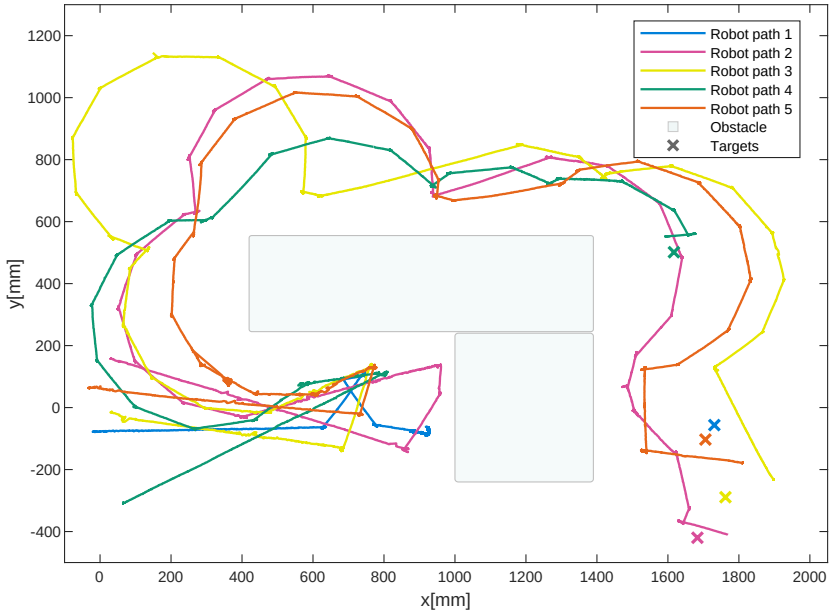


Figure 7.8: Test 2 for Algorithm 2: 5 attempts where the robot is to drive around a a ”L” shaped obstacle.

Figure 7.9 shows the resulting paths for Algorithm 2 in Test 3. In this test, all the attempts are successful. The paths in the figure show that the robot drives all the way around the first obstacle until it reaches the straight path between the starting point and the target, but on the other side of the first obstacle. It then turns towards the target and moves forward until it approaches the second obstacle. Finally it drives around the second obstacle and reaches the 15cm radius around the target. In attempt 2 the robot stops at the border of the 15cm radius, but a full point is still given. Algorithm 2 ends up with a score of 5 points in Test 3.

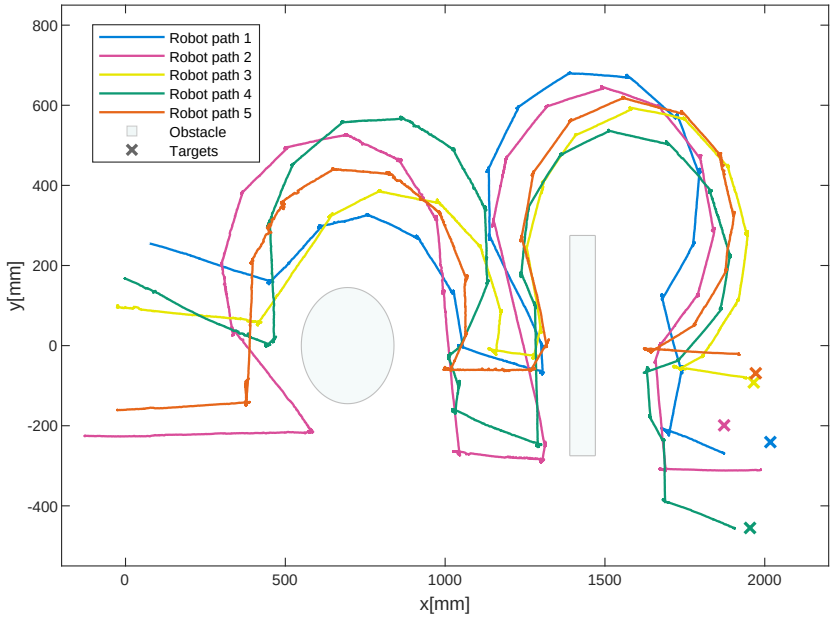


Figure 7.9: Test 3 for Algorithm 2: 5 attempts where the robot is to drive past tow obstacles, cylinder and a wall.

In the final test, the robot reaches the target in 3 out of 5 attempts. Figure 7.10 shows the robot paths for Algorithm 2 in Test 4. Path 1 and 2 shows that the robot drives between the circular and the upper obstacle and then all the way around the circular one until it is back on the straight path towards the target. In attempt 2, the robot had a free lane towards the target between the two smaller obstacles, while in attempt 1 the robot takes a slight detour to avoid the lower small obstacle before it finally turns towards the target. On the other hand, the rest of the paths shows that the robot does not manage to drive between the circular and the upper obstacle and therefor makes its way all the way around the upper obstacle. In attempt 3 the robot makes an unexpected large detour around the upper obstacle, but manages to reach the target. However, in attempt 4 and 5, the robot ends up driving around the upper obstacle and not reaching the straight path towards the target. For these attempts to end in success the robot would have had to either start off driving between the

circular and the lower obstacle, or it would have to approach the circular obstacle after driving around the upper one, such that it would drive to the bottom side of the circular obstacle and then meet the straight path towards the target. Algorithm 2 scores 3 points in Test 4.

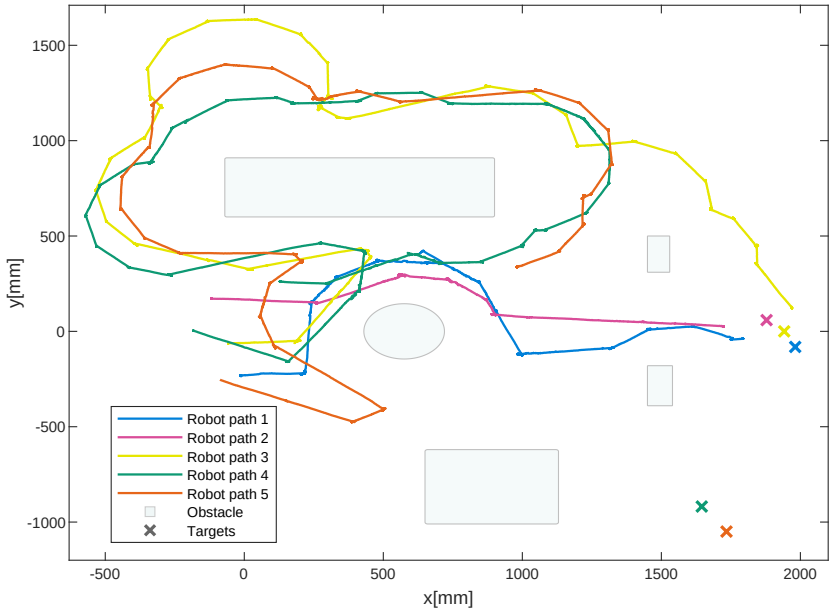


Figure 7.10: Test 4 for Algorithm 2: 5 attempts where the robot is to drive through/around a labyrinth of different obstacles.

7.4 Algorithm 3

7.4.1 Development and implementation

Algorithm 2 performed significantly better than Algorithm 1, but is still relatively inefficient. To make the obstacle avoidance more efficient, it was decided to further develop Algorithm 2 by letting the robot decide which direction it is best to turn when it approaches an obstacle, similarly to what was done in [Peng et al. (2015)]. This resulted in Algorithm 3 which is shown in Figure 7.11 and the implementation steps are listed below:

1. The sensor tower task notifies the pose controller when the robot is approaching an obstacle, but now, instead of just notifying about the obstacle, it additionally tells if the robot should turn left or right. As the sensor tower task discoverers the obstacle, it also checks the angular range from -60 to 0 and 0 to 60 degrees in the array keeping obstacle information and includes information about which direction (left or right) there were less obstacle readings, which is the direction the robot should turn.

2. The direction determined when the robot approaches the obstacle, does not change until it has left the obstacle avoidance loop. That means, if the robot approaches an obstacle and the density of obstacle readings are less at the angles -60 to 0 degrees, than from 0 to 60 degrees, the determined direction is right. Every time the robot approaches a new obstacle, within the loop, it will always turn right, which also means that after turning right, the sensor tower task will continuously check for free lane 90 degrees to the left. The robot will continue in this pattern until it is back on the straight path between the starting point and the target, the same way as in Algorithm 2.

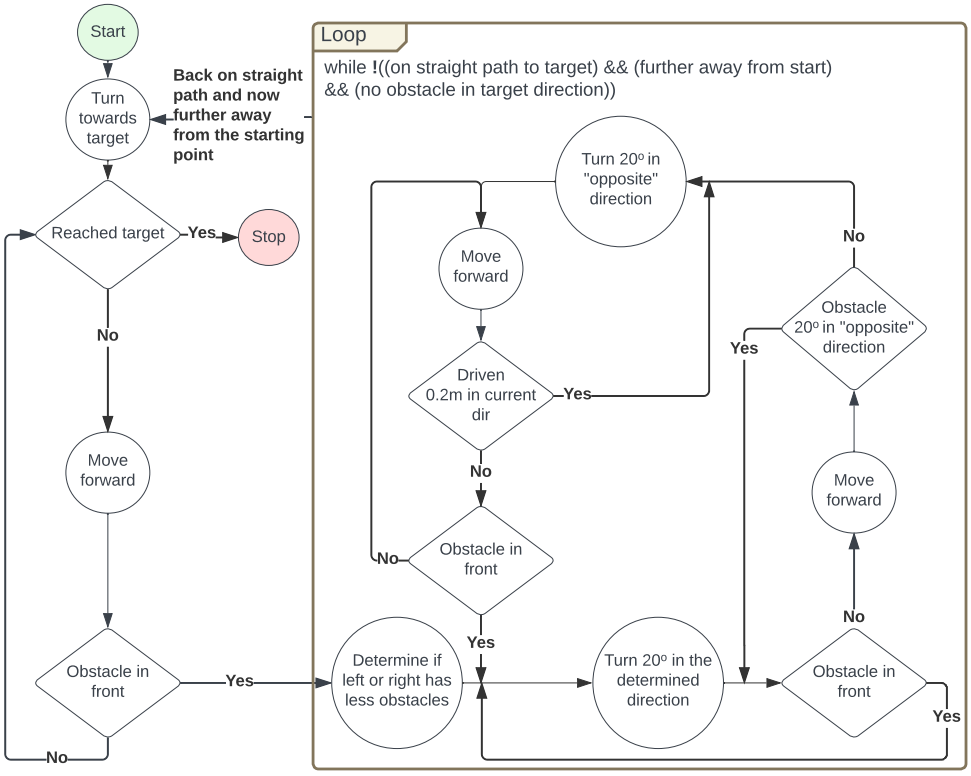


Figure 7.11: Algorithm 3. The resulting algorithm after further developing from Algorithm 2.

7.4.2 Results

This section presents the resulting behaviour of the robot with Algorithm 3 implemented.

Figure 7.12 shows the resulting robot paths for Algorithm 3 in Test 1. For attempt 1, 2 and 3 the robot drives around the obstacle and reaches the target without any major problems. Path 1 and 3 show that the robot chooses to drive left around the obstacle while path

2 shows that the robot chooses to go right. In attempt 5 the robot goes right, but crashes into the lower left corner of the obstacle a couple of times before it manages to go around. This results in poor position estimate, causing the robot to stop outside the 15cm radius around the robot, but since it is not too far off, half a point is given for attempt 5. Attempt 4, on the other hand, is completely unsuccessful. The robot takes an unexpected turn to the right in the very beginning and when it approaches the obstacle it gets stuck oscillating in the same position. Algorithm 3 gets a score of 3.5 in Test 1.

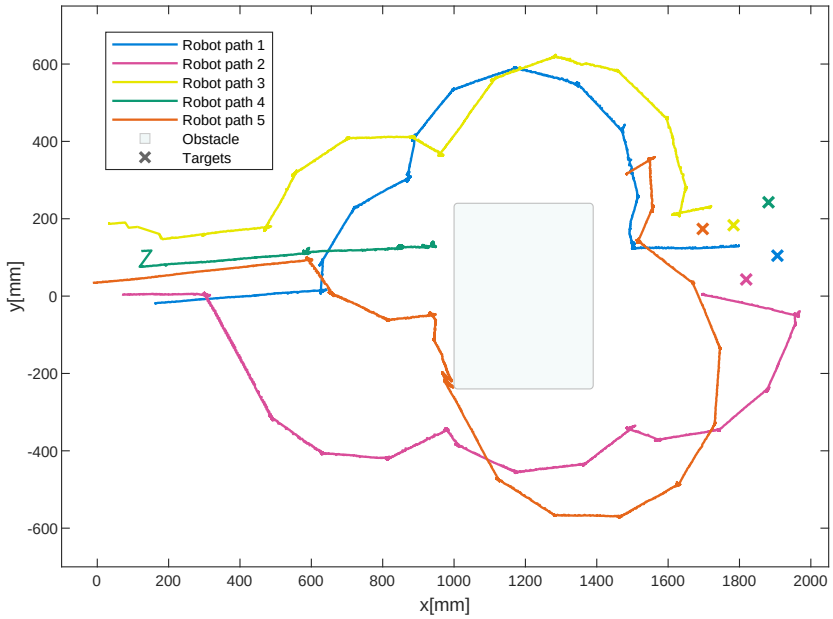


Figure 7.12: Test 1 for Algorithm 3: 5 attempts where the robot is to drive around a single box.

In the second test, the robot chooses, in attempt 4, to go left and therefor takes a detour around the upper part of the obstacle, while in attempt 2, 3 and 5, the robot chooses to go right which is a smarter choice. This is shown in Figure 7.13. In attempt 1, the robot loops without getting around the obstacle. It turns too much left and ends up on the straight path towards the target a couple of times before it gets stuck oscillating in the position where the path ends. In attempt 5 the robot crashes into the left corner of the lower part of the obstacle and such as in Test 1, the position estimate gets poorer resulting in the robot stopping outside the 15cm target radius. Also in this case, this gives half a point. Algorithm 3 receives 3.5 points in Test 2.

Figure 7.14 shows the resulting robot paths for Algorithm 3 in Test 3. In attempt 1 and 4 the robot reaches the target. Path 1 shows that the robot chooses to go right around both of the obstacles. However, path 4 shows that the robot goes left around the circular obstacle all the way around until it is back on the straight path. Then it turns towards the target and chooses to go left again when it approaches the second obstacle. In both these

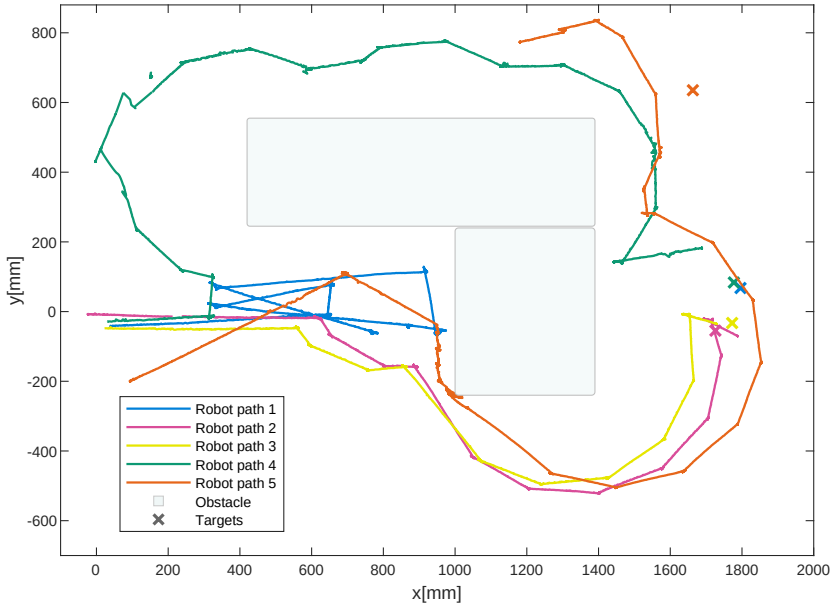


Figure 7.13: Test 2 for Algorithm 3: 5 attempts where the robot is to drive around a a ”L” shaped obstacle.

attempt the robot chooses the smartest direction. In attempt 3 the robot goes to the right around the first obstacle and turns towards the target when it is back on the straight path. When it approaches the second obstacle it turns left. From studying path 3 it looks like it would have been smarter for the robot to turn left in both of the cases. Additionally, the robot does not seem to notice the second obstacle when it has already passed it. The robot crashes into the obstacle on the back (on the right side of it) and tries to drive trough it for a while until it turns towards the target. Also in this case this results in poor position estimation, but half a point is given. In attempt 3 and 5 the robot crashes into the second obstacle several times and when it finally passes the obstacle it start to crash into the back of the same obstacle. Also in these attempts it seems like the robot, unexpectedly, does not register the obstacle. Algorithm 3 receives a score of 2.5 points in Test 3.

Also in the final test, Test 4, the robot has some unexpected behaviour. Figure 7.15 shows that the robot crashes in attempt 4 and 5. In attempt 4, the robot crashes into the back of the upper small obstacle, but ends up on the straight path towards the target and eventually reaches the target. For attempt 5 on the other hand, the robot crashes into the lower obstacle and does not seem to have registered it since it tries to drive trough the obstacle. For the rest of the attempts, the robot reaches the target successfully. Path 1 shows that the robot takes a left around the circular before it drive between the smaller ones. Path 2 shows that the robot takes a right, before it takes a left to avoid the lower smaller obstacle. Finally path 3 shows that the robot takes a right both around the circular and finally around the lower smaller obstacle. Algorithm 3 scores 4 points in Test 4.

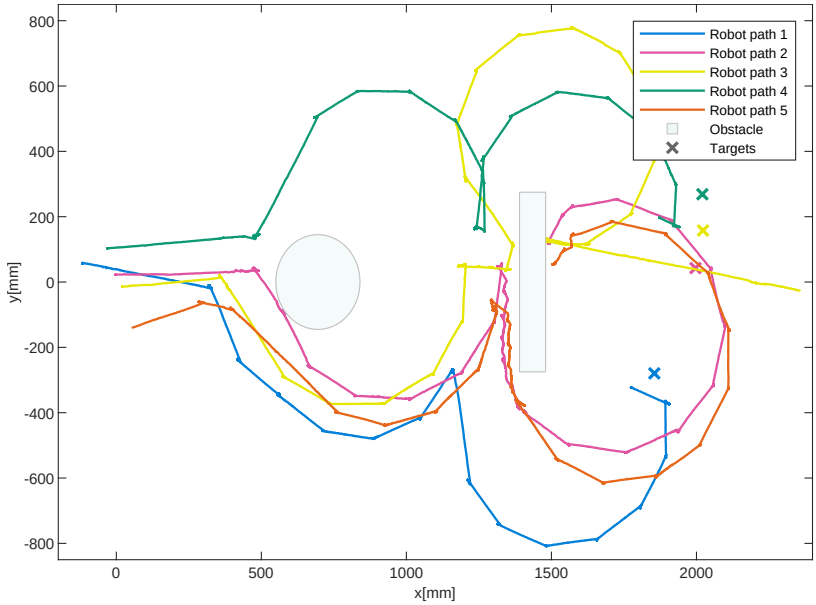


Figure 7.14: Test 3 for Algorithm 3: 5 attempts where the robot is to drive past tow obstacles, cylinder and a wall.

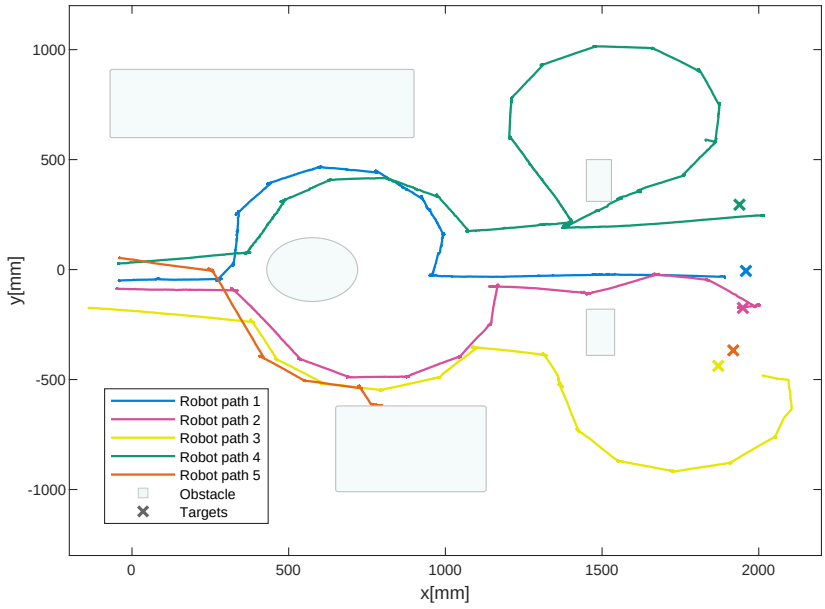


Figure 7.15: Test 4 for Algorithm 3: 5 attempts where the robot is to drive through/around a labyrinth of different obstacles.

7.5 Summary and comparison

For all four tests, all three algorithms has been rated by a score from 0 to 5 points based on the number of successful attempts. Figure 7.16 shows an overview of the scores given to the various algorithms in each of the four tests. For Test 1, Algorithm 1 and 2 gets full score while Algorithm 3 scores 3.5 out of 5. In Test 2, on the other hand, Algorithm 1 only scores 1 single point while Algorithm 2 scores 4 points and Algorithm 3 receives 3.5 points. Also Algorithm 1 has the poorest score in both Test 3 and 4, where it scores 2 and 0.5 points respectively. Algorithm 2 gets full score in Test 3 and 3 points in Test 4, while Algorithm 3 only scores 2.5 in Test 3, but gets the best score with 4 points in Test 4. This means that Algorithm 1 scores poorly on all tests except Test 1. Algorithm 2 gets the best score on Test 1 (together with Algorithm 1), Test 2 and Test 3. Algorithm 3 scores the poorest in Test 1, but has the best score in Test 4.

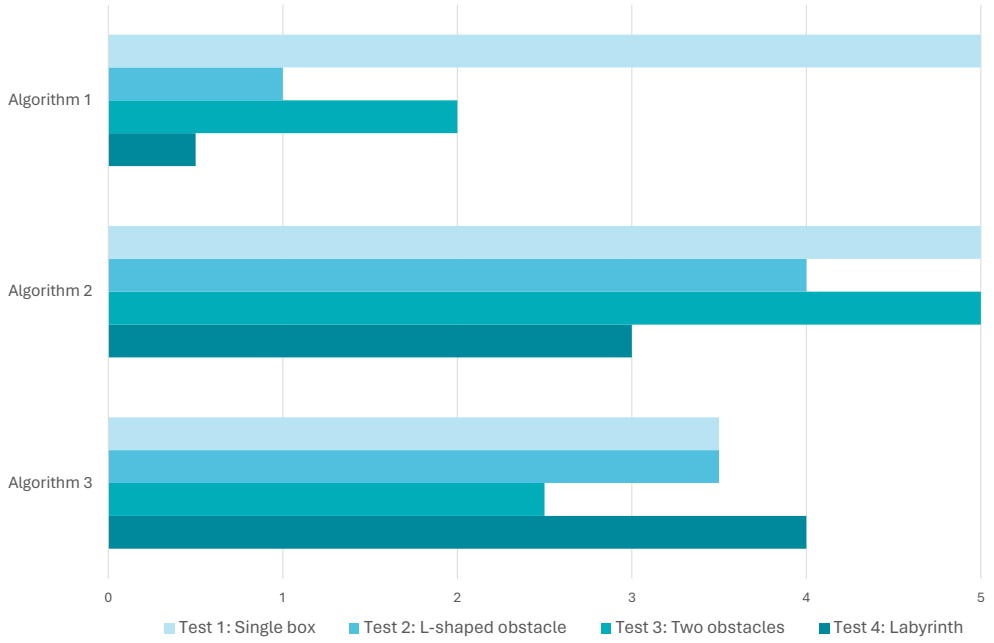


Figure 7.16: The figure shows the score for each version in all four tests.

Figure 7.17 shows the total score for each of the three algorithms. Algorithm 1 ends up with a total score of 8.5 out of 20 possible points, Algorithm 2 ends up with a score of 17 points and Algorithm 3 which suffers from a lot of unexpected behaviour scores 13.5 points. This means that Algorithm 1 gets the lowest score while Algorithm 2 receives the highest score by far.

The most important aspects of the algorithms is if the robot is able to reach the target.

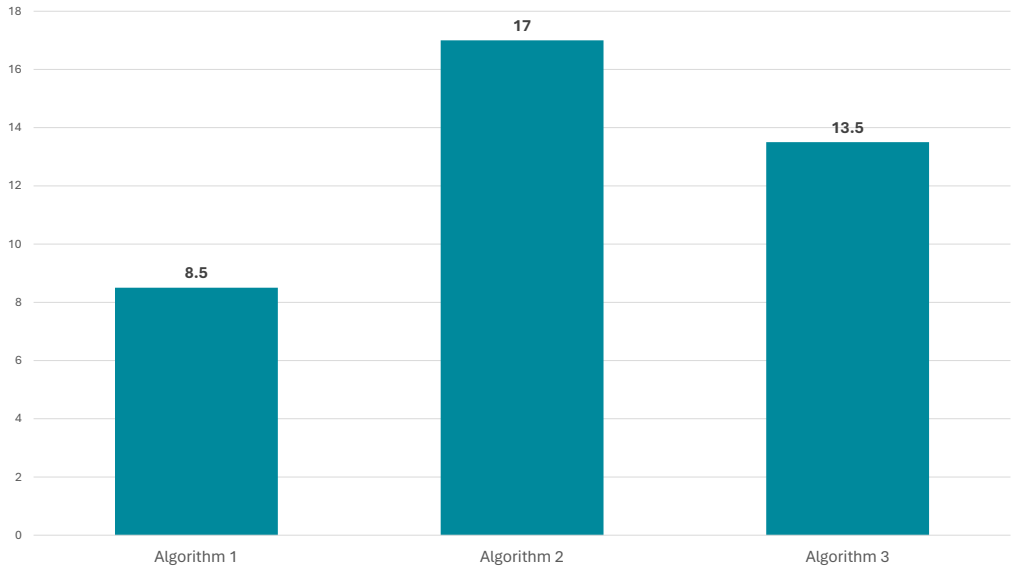


Figure 7.17: The figure shows the score for each algorithm in all four tests.

But, additionally, it is interesting to look at the efficiency of the robot in the successful attempts. By looking at the different plots and analyzing the successful attempts, Algorithm 1 seems to be pretty efficient in Test 1, but as soon as the obstacle is different from a single box, there is no doubt that Algorithm 1 is the least efficient one. In most of the successful attempt in Test 2, 3 and 4, the robot has numerous turns without moving significantly towards the target before it manages to drive around the obstacle. For both Algorithm 2 and 3, the robot drives long detours in some of the successful attempts to reach the target, but Algorithm 3 seems to choose more efficient routes, especially in Test 2 and 4. This means that, while Algorithm 2 is the best at leading the robot to the target, Algorithm 3 seems to do it more efficiently in its successful attempts.

7.6 Discussion

7.6.1 Error sources

The error sources discussed in the collision avoidance discussion (Chapter 6.6.1) is just as present in the obstacle avoidance as in the collision avoidance development and results. The robot position estimation drift off is even more visible for the obstacle avoidance tests, than they were for the collision avoidance tests. Additionally, the obstacle avoidance algorithms, only have 5 attempts in each test. This number should be significantly higher for more valid results.

7.6.2 Algorithm 1

As expected, Algorithm 1 suffers from the fact that it does not follow the obstacle all the way around before it turns towards the target. This causes the robot to move away from the obstacle when it approaches it, but instead of following the obstacle completely around, the robot often turns back towards the obstacle. After moving away from the obstacle, the obstacle is no longer detected on the robot's right side, allowing the robot to turn towards the target, which only leads it back to the obstacle. This issue occurs in several of the attempts in Tests 2, 3, and 4.

Because of the last minute change done during the final testing (see Chapter 7.2.1), Algorithm 1 suffers from the robot oscillating at the same position. This is hard to see from the plotted robot paths in the figures, but by studying the figures, it is possible to see some accumulations some places on some of the paths. These accumulations occur from this problem. The robot checks for a free lane at 90 degrees to its right, but turns towards the target instead of turning 90 degrees. This causes the robot to sometimes turn towards the target even though the lane in the target direction is not free from obstacles. The fact that the lane is free 90 degrees to the right, does not imply that the lane is free towards the target. It then turns 90 degrees to the left because of the obstacle still present in the target direction. In some of the cases where this problem occurs, the robot turns back and forth a couple of times before it moves forward in one direction, but in other cases it gets stuck oscillating until it has used the maximum number attempts it has to reach the target. Having the robot checking for a free lane in the same direction as it turns is essential to avoid this problem.

Another problem that became clear during Test 2, 3, and 4 for Algorithm 1 was the robot's poor position estimation. In the attempts where half a point was given, the position estimation indicated that the robot had reached the target, which did not correspond with the target position in the physical test area. This was revealed by looking at the server. The robot sends its estimated position to the server and the server shows the coordinates in addition to position the robot on the map. It seems that in the cases where the robot has a lot of turns, and the further it turns in each turn, the poorer the position estimation gets. It is interesting that in most of these cases, the robot seems to be turning too much right relative to the actual target position. The direction the robot is turning might determine which direction the position estimate drifts, but this has not been investigated and is therefor not the certain cause. However, this problem makes it hard to evaluate the performance of the obstacle avoidance algorithm. The performance could might have been better if the position estimation had been better.

On the other hand, Algorithm 1 performed well in Test 1. Where the robot meets a single and rectangular shaped obstacle, the robot can relatively easily follow the wall it first meets and turn right when it passes this wall. However, it is visible in the plot (Figure 7.2) that the robot turns right too early because it checks for a free lane 90 degrees to its right instead of in the target direction. By having it check in the target direction, the robot would probably take a few less turns on its way to the target.

7.6.3 Algorithm 2

Algorithm 2 is significantly better than Algorithm 1. Only 3 attempts ended in failure, which gave Algorithm 2 17 points, while Algorithm 1 only scored 8.5 point.

However, the robot seems to be less efficient in terms of time (in the successful attempts) because it only turns 20 degrees at a time. The motor speed controller slows the motors down to ensure that the robot stops at the correct angle, which adds time for each 20-degree turn. This becomes increasingly time-consuming the further the robot needs to turn. On the other hand, by having the robot turn more each time when it approaches an obstacle, it would probably take lager detours around the obstacle. By having the robot turn more each time it turns right, the robot would probably be more efficient, but still stick close to the obstacle. However, this solution would approach a problem in some cases. By having the robot turn for example 90 degrees to the right, the case in Figure 7.18 could occur. The robot approaches an obstacle which is slightly to the left of the robot. The robot rotates 20 degrees to the left, and since the obstacle was slightly to the left for the robot when it approached it, it is not visible 90 degrees to the right even after the robot has rotated. The robot is therefor told to turn 90 degrees to the right even though it has not passed the obstacle. This is why this solution would not work without further effort. A solution where the robot checks how far it needs to turn to obtain a free lane before initiating the turn, would be more efficient. Then, instead of stopping every 20 degrees, the robot would the calculated number of degrees directly.

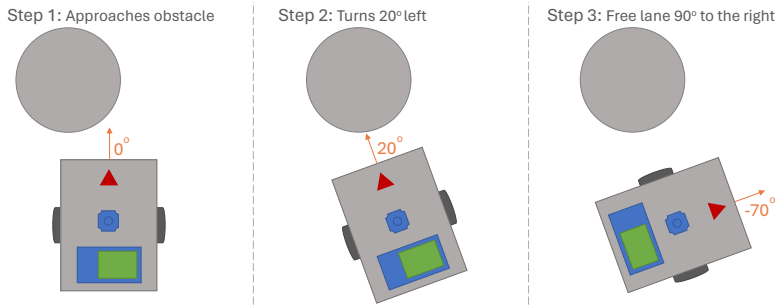


Figure 7.18: An illustration of the problem that would occur if the robot was allowed to turn further to the right when there is a free lane, compared to its allowance for turning left when it first approached the obstacle.

The robot tends to stay closer to the obstacle when it turns left further away from it, which is logical as it may need additional turns to avoid collision when turning close to the obstacle. This enhances efficiency and could justify increasing the collision threshold. However, increasing the collision threshold would cause the robot to deviate from sticking closely to the obstacle, which is also advantageous. As the results from Test 4 (Figure 7.10) shows, the robot sometimes does not end up on the straight path towards the target when it does not stick closely to the obstacle it first approaches.Allowing the robot to stick close to the obstacle will also enable it to drive through narrow passages, which is also beneficial.

Therefor, the solution discussed in the previous paragraph would be a better approach than increasing the obstacle threshold.

To further enhance efficiency, the robot could be programmed to turn right upon encountering a free lane to its right, not only after turning left to avoid an obstacle, but for all intermediate targets along its path. Only at the first intermediate target following an obstacle encounter, the robot awaits a right turn until it identifies a free lane to its right. However, for following intermediate targets (until it approaches a new obstacle or the straight path), the robot follows a repetitive pattern: it drives 20 cm forward in the direction of 20 degrees to the right. This often leads to unnecessary turns, as the robot may turn right even when an obstacle is present. Consequently, it must then turn left again before proceeding forward. However, always checking if the lane is free to the right, would result in cases where the robot would turn right immediately after approaching the obstacle (before the robot physically has had time to turn left). This may cause it to continuously turn right if there are no obstacles in that direction. A solution where the robot is not allowed to turn right before it has started to move along the obstacle would be required.

In the previous version there was a problem with the robot oscillating in the same position. By having the robot turning in the same direction it is checking for a free lane, the problem was expected to disappear, but it did not. The number of incident of this problem was reduced, but there are still some cases present in the test for Algorithm 2. Why this problem is still present is unknown and should be investigated. Additionally, the robot has some cases where it turns too far left before it starts moving forward. This suggests that there are registered obstacles further left than there actually are obstacles, which is also unexpected.

7.6.4 Algorithm 3

This version was expected to function similarly to Algorithm 2, with the added capability to navigate around obstacles to the right if there are fewer obstacles in that direction. However, Algorithm 3 showed a lot of unexpected behavior. These issues are likely due to errors in its implementation. These issue needs further investigation. One particularly strange behavior in Algorithm 3 is that collision avoidance seems to stop working in some cases. This was particularly visible in Test 3 where the robot crashed into the back of the second obstacle in two of the attempts.

Such as in the previous versions, the testing of Algorithm 3 also reveled poor position estimation. The robot believes it has reached the target, but the actual target is far from the position where the robot stopped. This issue occurred in all four tests. From the plots, it appears that the estimation worsens whenever the robot crashes and after making numerous turns. These observations are consistent with those made for Algorithm 1, which reinforces the claim that more turns and crashes (which cause the robot to turn) worsen the position estimation.

Considering that Algorithm 3 is a further development of Algorithm 2, it is strange that Algorithm 3 never leads the robot all the way around the upper obstacle in Test 4, whereas

Algorithm 2 did. There seems to be an unknown difference in the code. On the other hand, this can be a coincidence that could be revealed by conducting more tests.

The robot stops (the code appears to crash) sporadically. This happens for all versions and algorithms (both collision and obstacle avoidance), but it occurs more frequently when obstacle avoidance is implemented. In general, Algorithm 3 exhibits a lot of unexpected behavior, which needs to be fixed if this algorithm is to be used and further developed. Algorithm 3 scored worse than Algorithm 2, but if these problems are fixed, this algorithm is expected to be an improvement from Algorithm 2.

8

Further Work

8.1 Code clean up

As mentioned both in both discussions (Chapter 6.6 and 7.6) the robots suffer from unexpected software crashes. The robot code is big and unstructured and from Chapter 5.1 it was revealed that it contained a lot of code that was not even in use. The revealed unused code was removed, but the robot system would still benefit from having a more thorough clean up. In addition to limit the risk of software crashes and allowing more advanced code, such a clean up would also make it easier to discover implementation errors and unknown real time dependencies in the code.

8.2 Calibration of OptiTrack

Before conducting new tests using the motion capture system OptiTrack, it is essential to calibrate the system. This will minimize errors caused by shifting paths in the plots, ensuring that they correspond accurately with each other and with the obstacles such as observed during testing.

8.3 Collision and obstacle avoidance

For further work, Version 2 should be used and further developed as the chosen collision avoidance version. This is the version used in the robot code with obstacle avoidance implemented. For obstacle avoidance, Algorithm 2 performed the best, but it could be of interest to investigate the issues for Algorithm 3 described in Chapter 7.6.4. By resolving the issues, Algorithm 3 is expected to be the best obstacle avoidance algorithm. A robot code with collision avoidance Version 2 and where obstacle avoidance Algorithm 2 can be switched on and off (in the file *robot_config.h*) is provided in Tor Onshus' Git project. The robot code with Algorithm 3 implemented (without the ability to switch off the obstacle avoidance) can be provided from Tor Onshus upon request. The issues discussed

for Algorithm 1 in Chapter 7.6.2 is not seen as important to fix because Algorithm 2 and 3 is expected to be better even if these issues are resolved. Specific issues that should be further investigated and improved are listed below:

- Find out why it is not possible for the robot to store obstacle positions for each 360 degree around the robot and why it will not allow the sensor tower to turn one degree at a time. These issues are discussed in Chapter 6.6. The reason of these problems might be lack of capacity on the nrf-board, which means that this could possibly be resolved by a thorough code clean up. However, the problem might be something completely different, but finding the error would also be easier after a clean up.

- It would be interesting to have the speed controller slowing down the robot as it moves closer to an obstacle. If the IR sensors does a reading of an obstacle from far away and this reading is roughly correct, the sensors would be able to do more accurate readings as it gets closer to the obstacle. If this was implemented successfully, it would allow the robot to drive closely to the obstacle which is beneficial, especially for obstacle avoidance. This can possibly lead to the robot taking fewer turns when passing the obstacles, which makes the poor position estimation affect the behaviour to lesser extent.

- For obstacle avoidance, a new solution should be implemented where the robot checks all 180 degrees in the direction it is about to turn to determine how many degrees it has to turn to find a free lane. Today's solution where the robot turns 20 degrees at a time is time inefficient. By implementing this new solution the robot will turn directly into the position where it has a free lane ahead and start moving forward. If it is chosen to investigate the issues in Algorithm 3, this should be implemented into the fixed version of Algorithm 3. If it is chosen to move further with Algorithm 2, this solution should be implemented into Algorithm 2.

- Through the tests of both collision and obstacle avoidance it was revealed that the robot's position estimation is poor. Improving the position estimation would improve the robots behaviour in general, not only in collision and obstacle avoidance, but interpretation of obstacle avoidance algorithms would be significantly easier if this underlying problem was resolved.

- Another alternative solution to improve collision and obstacle avoidance is to use a camera instead of the IR sensors in these functionalities. This would require a complete study on how that works and how to implement it.

Bibliography

, . About freertos kernel. Website. URL: https://www.freertos.org/RTOS.html. last checked: 18.12.2023.

, . Embedded studio the leading multi-platform ide. Website. URL: https://www.segger.com/products/development-tools/embedded-studio/. last checked: 18.12.2023.

Baras, N., Nantzios, G., Ziouzios, D., Dasygenis, M., 2019. Autonomous obstacle avoidance vehicle using lidar and an embedded system, in: 2019 8th International Conference on Modern Circuits and Systems Technologies (MOCAST), pp. 1–4. doi:10.1109/MOCAST.2019.8742065.

Eidsnes, F.P., 2023. Collision and obstacle avoidance for mobile slam robots.

EMQX-Team, 2023. What is the mqtt protocol and how does it work? Website. URL: https://www.emqx.com/en/blog/the-easiest-guide-to-getting-started-with-mqtt. last checked: 18.12.2023.

Ismail, R., Omar, Z., Suaibun, S., 2016. Obstacle-avoiding robot with ir and pir motion sensors. IOP Conference Series: Materials Science and Engineering 152, 012064. URL: https://dx.doi.org/10.1088/1757-899X/152/1/012064, doi:10.1088/1757-899X/152/1/012064.

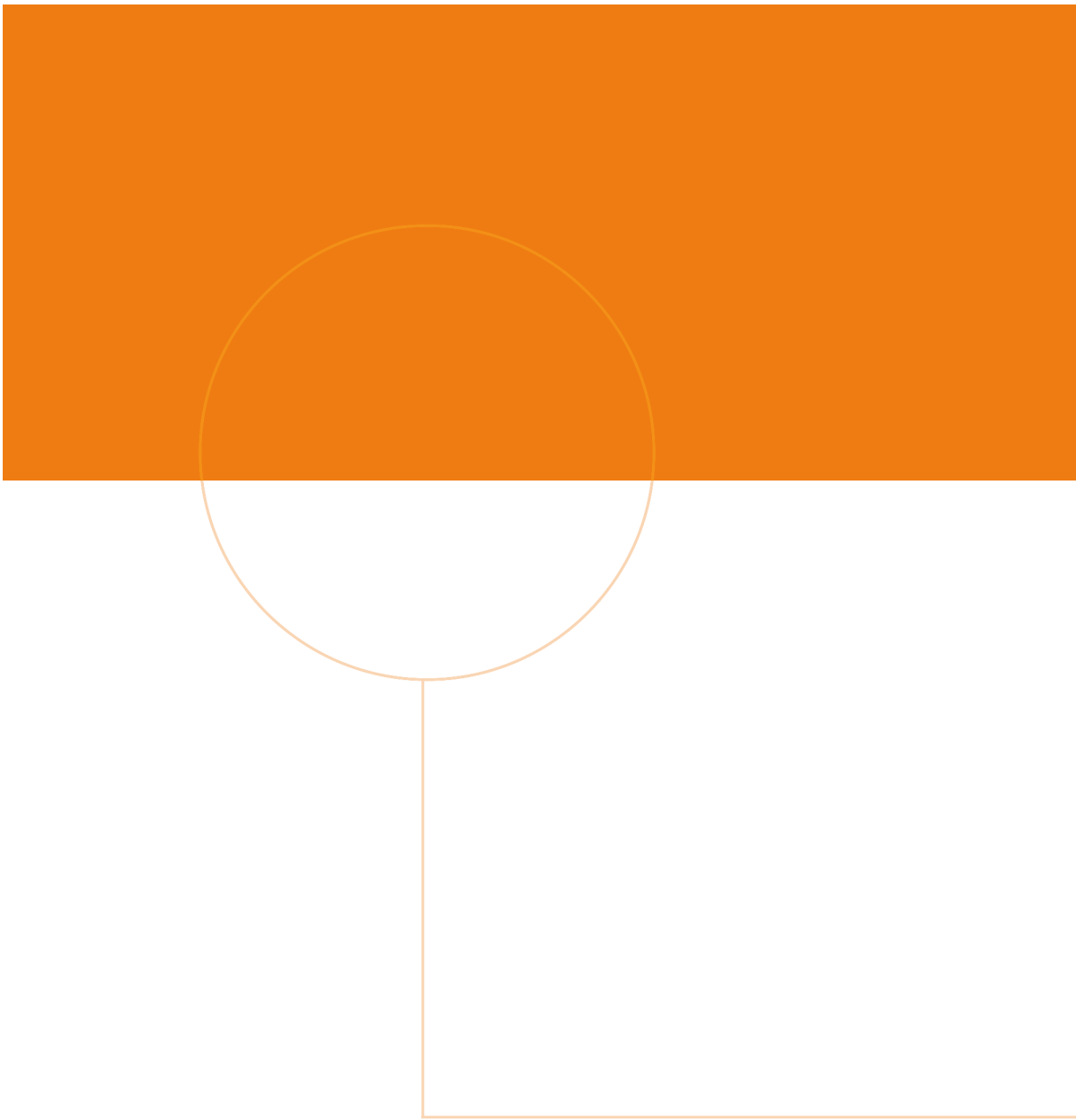
Khairuddin, A.R., Talib, M.S., Haron, H., 2015. Review on simultaneous localization and mapping (slam), in: 2015 IEEE International Conference on Control System, Computing and Engineering (ICCSCE), pp. 85–90. doi:10.1109/ICCSCE.2015.7482163.

Klose, W.A.L., 2023. A new server for the slam robot project.

Kunchev, V., Jain, L., Ivancevic, V., Finn, A., 2006. Path planning and obstacle avoidance for autonomous mobile robots: A review, in: Gabrys, B., Howlett, R.J., Jain, L.C. (Eds.), Knowledge-Based Intelligent Information and Engineering Systems, Springer Berlin Heidelberg, Berlin, Heidelberg. pp. 537–544.

Peng, Y., Qu, D., Zhong, Y., Xie, S., Luo, J., Gu, J., 2015. The obstacle detection and obstacle avoidance algorithm based on 2-d lidar, in: 2015 IEEE International Conference on Information and Automation, pp. 1648–1653. doi:10.1109/ICInfA.2015.7279550.

Yu, X., Marinov, M., 2020. A study on recent developments and issues with obstacle detection systems for automated vehicles. Sustainability 12. URL: https://www.mdpi.com/2071-1050/12/8/3281, doi:10.3390/su12083281.



NTNU

Norwegian University of
Science and Technology