NTNU
Norwegian University of
Science and Technology

Mapping and Navigation for collaborating mobile Robots.

Eirik Thon

Master of Science in Cybernetics and Robotics
Submission date: June 2016
Supervisor: Tor Engebret Onshus, ITK

Norwegian University of Science and Technology
Department of Engineering Cybernetics

Summary and conclusion

In this thesis software modules for mapping and exploration using mobile robots have been developed, and a simulator of the robots has been created.

Simulation: The work on the simulator started with a study of the behavior of the robots. The behavior was implemented step by step into what finally became a simulator. When the full system was completed the accuracy of the simulator was tested by running the program with both real robots and the simulator and comparing the results.

The tests gave very similar results both in robot behavior and the finished map. Additionally both the mapping module and navigation module were developed using the simulator, and did not have to be modified when first testing with the real robots. Based on this it was concluded that both the design and implementation of the simulator were successful and that it can be used as a tool for future development of the mapping and control program.

Mapping: Mapping can become quite difficult if the pose is unknown. A part of this thesis was to study existing methods that are able to deal with errors in the estimated pose. A couple of methods were found but there was not enough time to implement any of them. Instead a much simpler design that is not able to correct the pose was implemented.

Tests have shown that the mapping module works adequately if the environment is small and only one robot is used. Otherwise there are two problems:

- The position error of the robot grows over time so in a large environment it becomes so significant that the resulting map is becomes unusable.
- When several robots are used the map becomes unusable very quickly even in small environments, because their position-errors tend to grow in different directions.

The use of multiple robot is a key element of the project and it is necessary that the mapping module is able to generate consistent maps. Either the robots must become better at estimating their own pose (for example through more accurate sensors) or the mapping-module must be switched for a decent SLAM-algorithm.

Navigation: There was a number of challenges that had to be solved during the creation of the navigation-module. An utility function was created to select new target points for idle robots. A search-algorithm called A* was used fo find paths through the map. A system for controlling robots along a path using way points was developed. And finally a separate systems for detecting and handling possible collisions between a robot and an obstacle in the map or between two robots were created.

Testing with the real robots showed that the navigation-module was able to control a robot through the environment and efficiently explore it based on the map generated by the mapping-module. The only issue when testing was that the system did not have much time to react if the robot was on collision course with an obstacle. A change in the design of the collision-detection system might be needed to improve this. Unfortunately one of the robots had a damaged encoder, so it was not possible to test how the navigation-module handled multiple moving robots in that test.

Testing on bigger maps in the simulator showed that the time it took to explore an area was dramatically reduced when using more robots, however the utilization of the robots was not optimal. For even better performance it might be necessary to use a more advanced target-selection method than a utility-function. The multi robot tests also showed that the robots occasionally collided with each other, and there was some reason to believe that the detection system for collisions between robots was not working exactly as it should.

Oppsummering og konklusjon

Simulering: Arbeidet med simulatoren startet med å studere oppførselen til robotene. Oppførselen ble brukt som mal for å designe og implementere simulatore. Når resten av systemet var ferdig ble simulatoren testet ved å kjøre serverprogrammet med de virkelige robotene og simulatoren of sammenligne resultatene.

Testene ga svært like resultater både i oppførsel og det ferdige kartet. I tillegg hadde både kartleggingsmodulen og navigasjonsmodulen blitt utviklet ved å teste med simulatoren, og det var ikke nødvendig å gjøre endringer når de skulle brukes sammen med de virkelige robotene. Basert på dette ble det konkludert med at både designet og implementasjonen av simulatoren var vellykket.

Kartlegging Kartlegging kan bli vanskelig hvis robotes positur (eng. pose) er ukjent. en del av denne master oppgaven bestod i å undersøke metoder som er i stand til å håndtere feil i den estimerte posituren. Er par metoder ble funnet, men det var ikke nok tid til å implementere de. I stedet ble en mye enklere løsning designed som ikke retter opp feil i posituren.

Testing har vist at kertleggingsmodulen produserer et brukbart resultet for små områder når kun en robot brukes. Hvis flere roboter brukes eller et større område skal kartlegges oppstar det problemer.

– Posisturfeilen vokser over tid så i store områder vil feilen bi så stor at kartet ikke lenger kan brukes.
– Når flere roboter brukes kan kartet bli dårlig selv i små områder fordi feilen hos de forskjellige robotene vokser i ulik retning.

Bruk av flere roboter er en viktig del av oppgaven så resultatene var ikke helt tifredsstillende. Enten må robotene bli bedre til å estimere sin egen positur (for eksemple ved at de får bedre sensorer) ellers så må kartleggings-modulen byttes ut med en SLAM-algoritme.

Navigasjon Det var flere utfordringer som måtte overkommes under utviklingen av navigasjonsmodulen. For å velge nye mål for robotene ble det laget og brukt en utility-funksjon. Søkealgoritmen A* ble brukt for å finne en vei i kartet mellom robotens posisjon of målet. The ble utviklet et system for å styre robotene langs stien ved bruk av veipunkt. Til slutt ble

det laget systemer for å detektere og håndtere mulige kollisjoner mellom en robot og en hindring i kartet eller mellom to roboter.

Testing med de virkelige robotene viste at navigasjonsmodulen var i stand til å styre en robot og utforske området ved bruk av kartet som ble laget av kartleggingsmodulen. Det ble funnet at systemet hadde liten tid til å reagere hvis en robot var på kollisjonskurs med en hindring. Det vil kanskje være nødvendig å endre litt på designet av kollisjonsdetesjonssytemet for å forbedre dette. Desverre hadde en av robotene en feil på en av enkoderene så det var ikke mulig å teste navigasjonsmodulen med mer enn en bevegelig robot.

Videre testing med større kart i simulatoren viste at tiden det tok å utforske et område sank dramatisk hvis det ble brukt flere roboter, men det virket ikke som robotene ble utnyttet til det maksimale likevel. For å øke ytelsen til systemet vil det kanskje være nødvendig å bruke en mer avansert metode enn en utility-funksjon for å velge nye mål for robotene. Testingen viste også robotene av og til kolliderte med hverandre og det var grunn til å tro at systemet for å detectere kollisjoner mellom to roboter ikke fungerte helt slik det var designet.

Acknowledgements

I want to thank my supervisor Tor Onshus for guidance and support during the project. I want to thank Edmund Førland Brekke for taking some time to teach me about SLAM. Lastly but not least I would like to thank Erlend Ese, Thor Eivind Svergja Andersen and Mats Gjerset Rødeth for all the help i received and for a great semester!

"A man's greatest joy is crushing his enemies"
-Genghis Khan

Contents

List of Figures

List of Tables

Glossary

multi-robot SLAM A SLAM algorithm that uses data from multiple robots to make the map..

on-line SLAM A SLAM algorithm that creates the map in real time.

pose The orientation and position of a robot. For robots operating on a flat surface, the pose consists of its x-,y-coordinates and its heading.

List of Acronyms

API	Application Programming Interface.
BFS	Breadth-First Search.
LIDAR	Light radar.
SLAM	Simultaneous Locatlization And Mapping.

Chapter 1
Introduction

1.1 Project vision

Exploration using multiple robots has been an important part of the robotics field for many years. Multiple robots can potentially cover an area much faster than just one if the work together. To make the robots cooperate and generate an accurate map requires both timing and precision. Several existing system can do this using laser-scanning technology and GPS, but is it possible to create such a system using only cheap and simple technology? This thesis is part of a project with the goal of creating such a system.

1.2 Project overview

The LEGO-robot project has been going on since 2004. At the start of this semester it consisted of a couple of robots, a robot simulator and a program for mapping the environment and controlling the robots. The robots uses IR-sensors to measure the distance to nearby obstacles and reports the measurements to a computer running the mapping program (hereafter referred to as the server). The program uses the IR-measurements to create a map of the environment and navigate the robots to unexplored parts of the environment.

Erlend Ese, Thor Eivind Svergja Andersen and Mats Gjerset Rødseth have been working on the project at the same time as I have. The responsibilities were as follows:

- **The server program:** Thon (me), Andersen and Rødseth
- **The software on the robots:** Ese
- **The software on the bluetooth chips:** Andersen, Rødseth and Ese
- **The simulator:** Thon

Adiditonally Andersen and Rødseth have built a new robot for the project. The work of Andersen and Rødseth can be be found in [AR16], and the work of Ese can be found in [Ese16].

In the beginning of the semester Andersen, Rødseth and I decided that instead of trying to improve the old server-program it was better to create a new one. This was because the old system was performing poorly [Ese15] and the code was poorly maintained and difficult to understand.

It was decided to create the new software system using Java because it is widely used and well suited for building complex software-systems. This meant that the simulator would no longer work because it was written in Matlab. The simulator was not performing well either [Ese15] and it seemed best to remake it as well.

1.3 Task

Figure 1.1 shows main components of the total system, along with the inner components that the soft-ware program needs to operate. The task of this thesis is to design and implement both the mapping- and the navigation-module, and also create a new simulator.

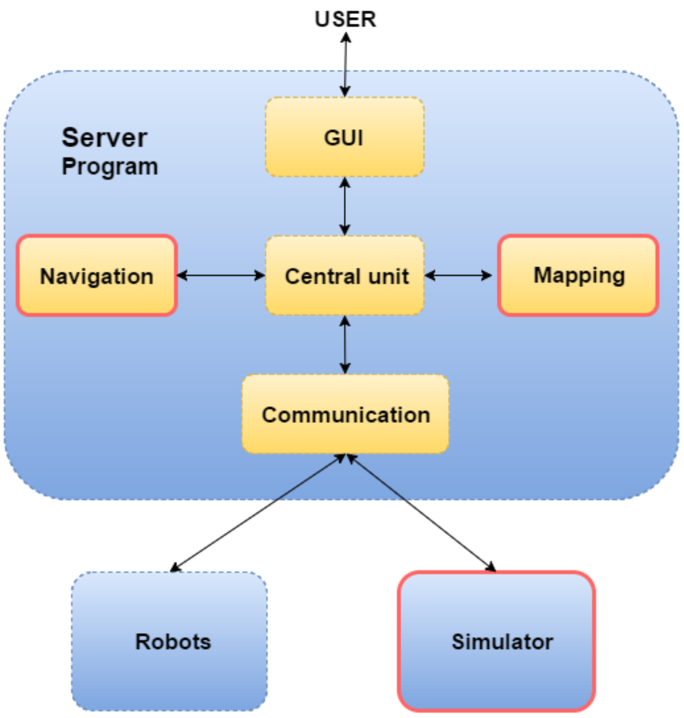

Figure 1.1: An overview of the core components of the system. The task in this thesis is to create the modules that are circled with red

1.3.1 Task 1 - Creating the simulator

The simulator has to be made first as it is needed in order to develop and test both the navigation- and the mapping-module. It should be designed so that it fulfills the following criteria:

- It must be able to simulate N-number of robots.
- It should replicate the behavior of the real robots in such a way that one can feel certain that if the server-system performs well in the simulator it performs almost equally well with the real robots.

- Data from the simulator should not have to be handled differently that data from the real robots.
- The user should be able to see what is going on inside the simulator.

To fulfill these criteria it is necessary to study the behavior of the robots closely, and to openly discuss details with the other members of the project during the design process.

1.3.2 Task 2 - Creating the Mapping-Module

The mapping-module has to be created secondly because the map is needed for testing and developing the navigation-module. The goal of the design is to make the mapping module do the following:

- Use the data from every robot to create the map.
- Create the map in real-time so that it can be used by the navigation-module.
- Create a map that is consistent with the real environment.

Mapping is a big and quite complex field. A research study will first be conducted in order to learn about different solutions to the mapping problem. The findings of the research is covered in chapter 2.

1.3.3 Task 3 - Creating the Navigation-Module

The last task in this thesis is the design and implementation of the navigation module. The goal of the design is to make the navigation-module do the following:

- Use every robot when exploring.
- Distribute the robots evenly throughout the map to make the exploration as efficient as possible.
- Make the robot not collide with one another or with the obstacles in the environment.

Based on this goals a number of different tasks had to be done.

- Design and implement a utility function that can be used for target selection.
- Selecting and implementing a path-finding algorithm.

- Create a system for controlling a robot along a path.
- Create a system for detecting possible collisions.
- Design and implement routines for avoiding collisions.

1.4 Report structure

Here is a brief description of the contents of each chapter:

- **Chapter 1 - Introduction**
- **Chapter 2 - Background.** Covers both a short description of the system, along with some of the theory used in the rest of the thesis.
- **Chapter 3 - Simulator.** Covers the design and behavior of the Simulator, as well as a brief overview on implementation.
- **Chapter 4 - Mapping.** Same as for chapter 3 but with regards to the mapping module
- **Chapter 5 - Navigation.** This chapter presents main challenges of creating the navigation module, how each of them were solved separately, and lastly an overview on implementation.
- **Chapter 6 - Results and Discussion.** This chapter covers a number of tests that were used to measure the performance of the simulator, the mapping-module and the navigation-module, and a discussion of the results.
- **Chapter 7 - Conclusion.** This chapter covers a separate conclusion on the performance of the simulator, the mapping-module and the navigation-module.
- **Chapter 8 - Further Work.** Chis chapter lists a number of things that could or should be done with the simulator, the mapping-module and the navigation-module, and in some cases suggestions on how to do it.

Chapter 2 Background

2.1 System overview

The whole system in this project consists of the robots, a computer (hereafter referred to as the server) and some environment that the robots can explore. The robots are more or less brainless agents that are controlled by the server. The robots and server uses Bluetooth to communicate with each other. An image of the AVR-robot can be seen in figure 2.1

The robots have 4 IR-sensors mounted on a rotating tower with which they can measure the distance to nearby obstacles. They also have wheel-encodes, an IMU and a compass which they use to compute an estimate of their own pose. The robots periodically reposts the measurements and the estimated pose to the server, and the server uses this information to construct a map and navigate the robots to unexplored areas of the map.

The robots can execute simple commands one at a time. The commands must be on the format [angle, distance]. The the angle tells the robot how much to rotate, and the distance tells the robot how much to move forward. When a robot has completed a command it sends an "Arrived"-message to the server.

2.1.1 Project background

In 2004 th LEGO-robot project was started on NTNU. The motivation behind the project is to create a system that efficiently maps an area using cheap sensors and simple robots

The first robot was built in 2004 [Skj04] and a second one in 2008 [Bak08]. Both were using LEGO for construction

[Mag08] created a simulator of a robot in a maze.

[Syv06] created a software-system that was able to do mapping and exploration with one robot. Both the mapping algorithm and navigation-algorithm were modified and improved in the following years. [Tus09] and [Hom13] gives a nice overview of the progress from 2004 to 2012.

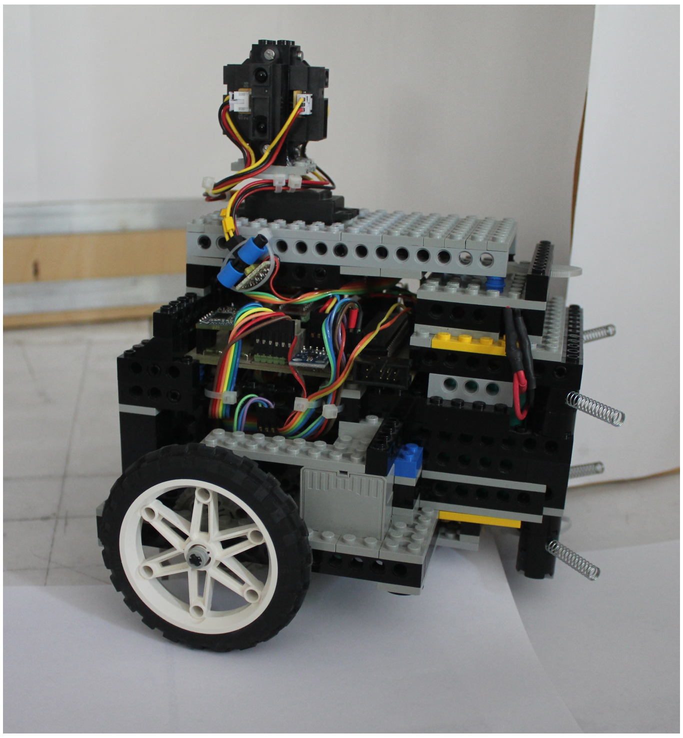

Figure 2.1: The AVR-robot in the project

[Hal12] modified the system to include both robots. The mapping algorithm was modified to merge the maps created from each robot after the completion, while the navigation algorithm was changed from using left-wall following to an frontier-based approach.

In the fall of 2015 Erlend Ese was working with the system and discovered a number of issues regarding both the robots and the server-program [Ese15].

2.2 SLAM

The mapping problem in this project falls within a category of problems called SLAM-problems. Therefore a lot of time and effort were put into studying existing solutions to the SLAM-problem. This section documents much of the information found. It should be noted that none of this theory or methods were used in this thesis. The purpose of this section is to document the research, as a part of the work done in this thesis, and also provide tips and recommendations for further work.

2.2.1 What is the SLAM-problem

SLAM is an acronym for Simultaneous Localization And Mapping. It is the problem of generating a map when the robots pose is unknown. It only applies to robots who measure its environment relative to themselves, using for example a range-sensor or some visual sensor. The difficulty of the problem lies in that the pose is needed to place the measurements correctly to generate a map, but a map is needed to correctly compute the robots pose based on its measurements. In order to solve the SLAM-problem both the map and the pose must be estimated and updated simultaneously. [ST05] and [Sta16] provides two great sources for introductory SLAM-theory. Both sources were used extensively in this research.

2.2.2 Map representation

In all solutions to the SLAM-problem that were found during research one out of two types of map-representations were used. The first is feature based maps. The slam algorithm relies on ectraxting visible landmarks form the measured data, identifying those and placing them in the map. The other type is occupancy grid-maps, where the map is divided into a grid and each cell represents the binary state of free or occupied. Both have certain advantages and disadvantages:

Landmark based mapping:

- Is well suited when the environment contains easily recognizable features.
- Low data usage as the map consists only of a set of coordinates for each feature.
- One of the biggest challenges is to distinguish new landmarks from previously observed landmarks.

Occupancy-grid based mapping:

- Well suited if the robot uses range sensors.

- Easy to perform path-planning and navigation.
- A big problem is data-usage. Each cell in the map require a binary variable to say if its occupied or not. A map that is 100x100 meters with a resolution of 1cm would require 100 Mega-bit to represent, even if it was completely empty.

2.2.3 Short overview of different methods

There are 3 paradigms in solutions to the SLAM problem. Solutions using an extended Kalman-filter (EKF), solutions using a particle filter (PF) and graph-based methods. Graph-based methods originally only solves the full-SLAM problem, but methods exists that uses a sparse extended information filter (SEIF) to solve the on-line slam problem. Of all these methods the SEIF-method is the most modern and according to [**?**] well suited for multi-robot handling. However the underlying math is a bit more complicated compared to EKF and PF.

2.2.4 Existing implementations

cg mrslam cg mrslam [mrs16] is a graph-based solution to the on-line multi-robot SLAM-problem. [Laz13] is a paper documenting the theory behind the algorithm.

GMapping GMapping is a single-robot, off-line algorithm using a Rao-Blackwellized particle-filter. The algorithm was developed by, among others, the author of [ST05]. It is used by the slam_gmapping-method which is part of ROS [ros16]

HectorSLAM A less advanced single-robot SLAM method that uses scan-matching to correct errors in the robot pose, but is unable to do loop-closing [hec16].

2.3 A*

A* (pronounced a-star) is a search-algorithm that utilizes the principle of best-first. It is similar to Dijkstra's algorithm in several ways. The algorithm uses an open set and a closed set and new nodes are inserted into the open set if they are not seen before. The algorithm computes a cost for each node and at each iteration it selects the node with lowest cost from the open-set. The cost is the sum of a traversed cost and a heuristic cost.

The traversed cost of a node is the accumulated edge-cost of traveling along the shortest path from the start-node to the node. The heuristic cost is an estimate of the unknown cost of traveling from the node to the target. The heuristic cost must be designed by the programmer, and if well designed the algorithm will search through the graph in the direction of the target node.

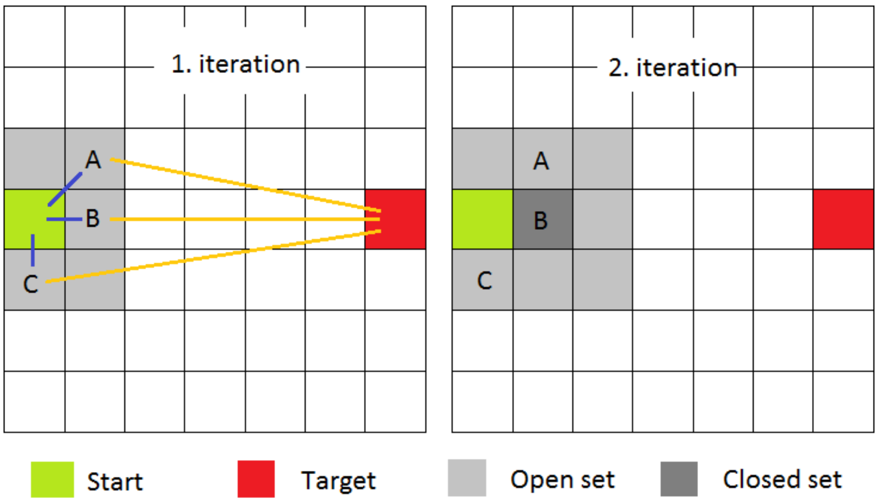

Figure 2.2: One iteration of the A* algorithm

Figure 2.2 illustrates one iteration of the A* algorithm on a problem of finding the best path through a grid. The blue line illustrates the traversed cost and the yellow line illustrates the heuristic cost. The total cost is the combined distance of both lines. For this example the heuristic cost has been chosen as the distance along a direct line to the target cell. The figure shows that B gets selected because it has the lowest total cost.

For the A* algorithm to find an optimal solution in a graph the heuristic cost must be consistent. That means that the estimated cost of reaching the target node from a node n, should be shorter or equal to the heuristic cost of any neighbor n' plus the traversed distance between n and n'. Or formally: $h(n) \leq c(n, n') + h(n')$.

Source: [SR14].

2.4 Utility Theory

Utility theory is a much used concept for simple decision-making systems. In such systems utility is used to describe what the agent wants. In most cases the agent should select the action that maximizes its expected utility.

2.4.1 Utility Function

The utility function is a function that puts a value on each of the agent's possible actions. The difficult bit is to design the utility function so that the best action gets the highest value. Often in more complex environments a successful utility function consists of several weighted attributes that maps to different parts of the state of the environment (Multi-attribute Utility Function).

Source: [SR14].

Chapter 3
Simulator

3.1 Approach

The idea behind the simulator was to create a virtual version of the robot environment. This was achieved by creating virtual robots that behave in the same way as the real robots and are able to communicate with the application (see section 2.1 for robot behavior). The virtual robots operate in an environment that has obstacles. The robots can measure the distance to both nearby obstacles and other robots in 4 directions. The server can give commands to the simulated robots through the API of the simulator. When a robot receives a command it executes it the same way as the real robots by first rotation and then moving.

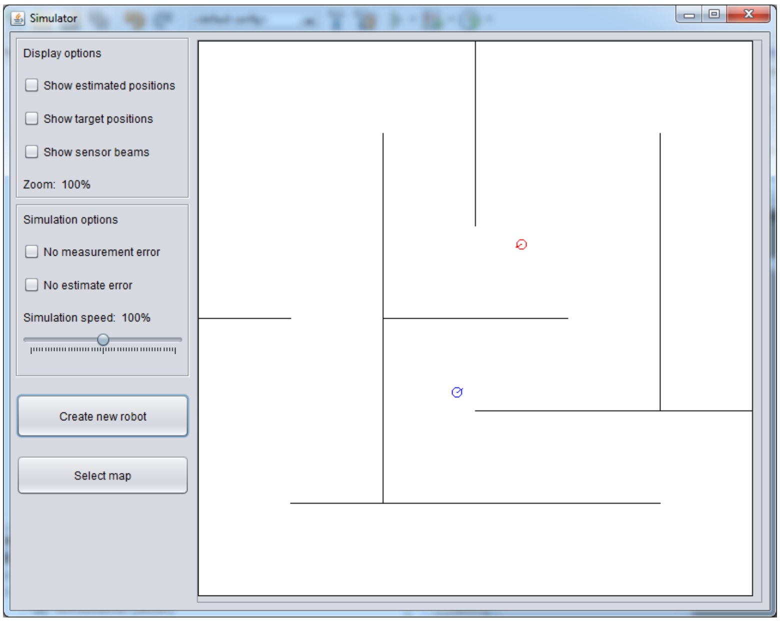

Figure 3.1: The interface of the simulator

The virtual environment is simply a coordinate system. The environment contains obstacles that are represented as a line between two points. The robots can move freely around in this environment, but can not go through obstacles or each other.

An interface was created for the simulator to allow user to interact with the simulator to some degree. The interface can be seen in figure 3.1. The main component of the interface is the display. Here the environment in the simulator is drawn in real-time like an animation. This enables the user to see the actual positions of the robots and walls and see what is going on in the simulator.

Some simulation options were added to make the simulator a more valuable developing tool for the project. These options can be adjusted through the interface. A slider lets the user adjust the simulation speed from 1-1000% of normal speed, and two check-boxes lets the user turn on and of noise.

A manual has been made for the simulator on how to use it. It can be found in the appendix.

3.2 How measurements are made

The simulator does not have a clear representation of the real IR-sensors. Instead it represents the direction of the measurement tower as an angle and uses the angle to compute where the line of sight for each sensor should be. A measurement is created by finding the nearest intersection point between the line of sight and any of the other objects in the environment, and returning the distance from the robot to this point. This is illustrated in figure 3.2

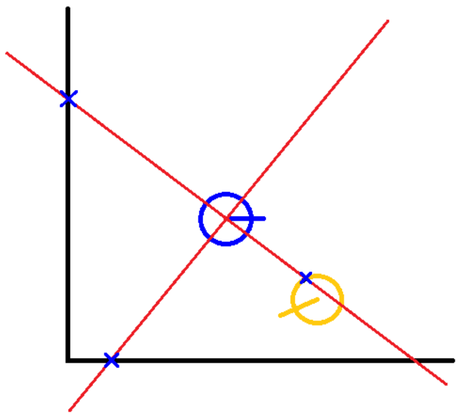

Figure 3.2: This figure illustrates where a simulated robot makes measurements. The red lines represent the line of sight og each sensor, and the blue X's represents the position of each measurement.

The intersection points are found mathematically. The intersection between the sensor's line of sight and a wall can be written as:

$$p_1 + tv_1 = p_2 + sv_2.$$

p_1 and p_2 represent the starting point of the line of sight and the wall respectively. v_1 and v_2 represent unit-vectors in the direction of the line of sight and wall respectively. t and s are scalars. t represents the distance from the robot to the intersection point and is used as a value for the measurement. The equation set was solved and the explicit solution for t is used in the simulator for generating measurements of walls. This method was found on an on-line forum [lin16].

The method for finding intersections with other robots was inspired by the first method. The other robots are represented as a circle :

$$(x - x_c)^2 + (y - y_c)^2 = r^2,$$

where $[x_c, y_c]$ is the position of the other robot. The intersection with the sensor's line-of-sight was found by constraining the x- and y-values of the circle to position along a straight line. This led to the equation:

$$(p_x + tv_x - x_c)^2 + (p_y + tv_y - y_c)^2 = r^2,$$

where $[p_x, p_y]$ represents the starting point of the line, and $[v_x, v_y]$ represents the direction vector of the line. The explicit solution for t was found using the quadratic formula. t can have 0, 1 or 2 real solutions and if a solution exists it represents the distance from the center of the measuring robot to the point where the ray intersected with the other robot. The solution with the lowest value is therefore selected as measurement.

3.3 How noise is simulated

The biggest challenge of generating a map in this project is noise. Measurement noise can distort the map, while errors in the pose-estimate may make the map completely inconsistent. Because this is such a central problem for the project, noise is also added to the simulator.

The simulator can add noise to the values computed by finding intersection points (see the previous section). This is done by drawing a random variable from a Gaussian distribution (See fig 3.3) and directly adding it to the measurement. The distribution has a mean value of 0, value and the variance was tuned so that it approximately matches the variance of the sensors.

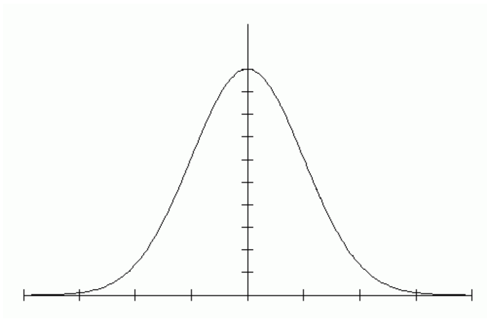

Figure 3.3: The Gaussian distribution was used for generating noise in the simulator

The simulated robots has both a real pose and an estimated pose. Both poses are updated almost exactly the same during movement, however a small Gaussian value is added to the estimated pose at each simulation step. This makes the estimated pose randomly diverge from the real pose during movement. When the robot measures obstacles it uses the real pose for reference, however when it sends the measurements to the server it sends the estimated pose. This causes a growing mismatch between the measured position of an obstacle versus its actual position in the simulator.

3.4 Implementation

3.4.1 From the outside

From the outside the simulator can be seen as a black box. Although it is a part of the server-program's source files it should be viewed as a completely different module (See fig 3.4).

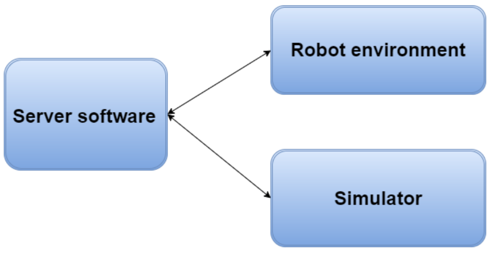

Figure 3.4: The relationship of the simulator to the other parts of the project

Figure 3.5 shows exactly what parts of the total system that is simulated. It was designed this way to avoid having to use serial communication to send messages to the simulator. In practice the line in figure 3.5 marks the API of the simulator. The API of the simulator contains the same functions as the "serial send"-module of the server. This makes it easy for the programmer to make the application switch between the real robots and the simulator.

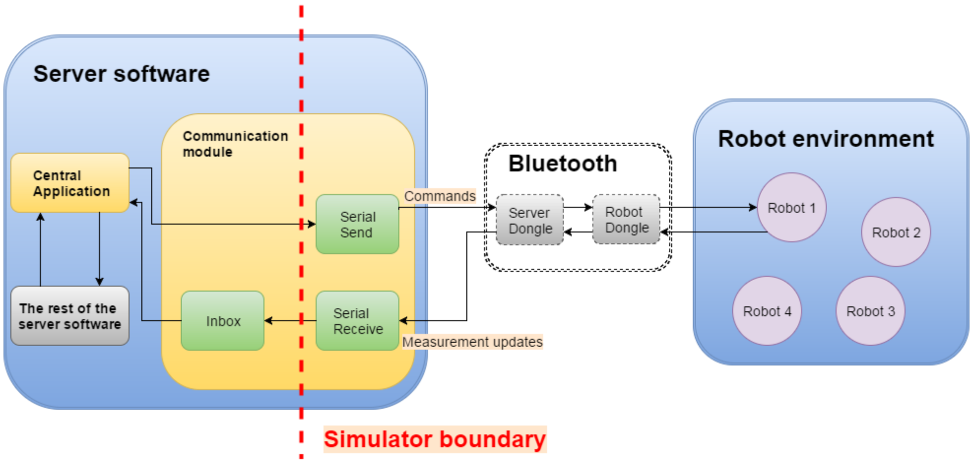

Figure 3.5: The figure illustrates the boundary of the simulator. The simulator simulates everything on the right side of the dotted line.

3.4.2 On the inside

Figure 3.6 shows the internal structure of the simulator. The virtual robots are objects contained in a world object. Each robot has a thread that can access the robot and control its behavior. The display of the simulator has access to the simulated world and through it also the robot. The display uses this access to acquire the necessary information to draw the simulated environment.

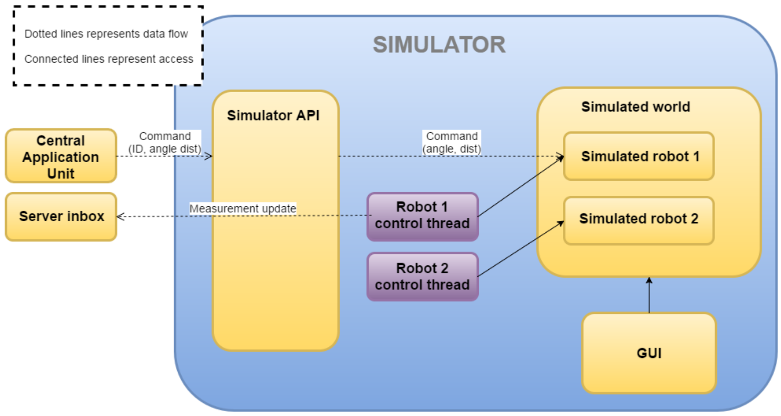

Figure 3.6: The internal structure of the simulator

The virtual robots are simply a set of variables representing the real pose, the estimated pose and the tower angle. The thread controlling them operates according to algorithm 3.1.

Algorithm 3.1 The robot-behavior algorithm

```
counter = 0
while thread is not interrupted {
   sleep(10 ms);
   if robot is paused {
      continue;
   }
   robot.move();
   counter++;
   if counter is greater than 19 { //  200 ms has passed
      counter = 0;
      robot.turnSensorTower();
      robot.makeIrMeasurement();
      message = robot.makeUpdateMessage();
      inbox.addUpdate(message);
   }
}
```

A short note about the different functions:

– **robot.move()** moves the robot a little bit according to its current command. For example if the robot is not done rotating the function rotates the robot a little bit. A simple if-based controller regulates this behavior. The estimated pose is updated similarly but with noise added.

– **robot.turnSensorTower()** increases/decreases the sensor-tower-angle by 5 degrees. The angle stays within 0-90 degrees.

– **robot.makeIrMeasurement()** calculates the measurements according to the section 3.2 and stores the values.

– **robot.makeUpdateMessage()** returns a string containing values of the estimated pose, sensor tower angle, and the last IR-measurements in the format used for robot updates.

Chapter 4
Mapping

4.1 Design

The initial idea for how to make the mapping module was to find and implement a suitable SLAM-method (See section 2.2 on SLAM). However it became apparent through research that SLAM methods are to complicated for one without experience to implement within a few months. Another option was to use an existing implementation. Some implementations were found however the all required some modification to fit into this project. After having spent much time on research and study it was simply not enough time left to modify one of the existing implementation, and so a much simpler solution had to be designed.

The idea behind the mapping module is simply to plot the IR-measurements directly into a map using the estimated pose of the robots. No adjustments are made to the pose-estimate and no information about surrounding obstacles is extracted from the measurements. For map representation a grid-map is used because it suits the problem in this project nicely for several reasons (cf. chapter 2.2 for theory on the grid-map):

- The real environment is fairly small (the pre-built labyrinth is only 1.5x1.5 meters), so the grid-map will not have the problem of using a lot of data.
- No advanced path-planning method or map-processing is needed for navigation.
- The other option is to use a feature-based approach, however this approach is somewhat ill suited because the environment contains mostly straight walls and it is hard to locate the position of a wall based on a small number of IR-measurements.

4.2 Behavior overview

The robots initially sets their estimated position and heading to just zeros. In order to use the estimate to locate a robot in the map, the mapping algorithm needs to know the robots initial pose relative to the map. This must be typed into the program by the user. When the program is running the mapping-controller uses the initial pose to translate and rotate the estimated pose of the robots.

Using the robots estimated pose int the map, and the rotation angle of the measurement tower, the mapping controller can compute the location of each IR measurement. The map is created by finding the cell in the map that corresponds to the location of the measurement and setting its occupied status to true. The mapping controller also finds each cell along a straight line between the robots position and the position of the measurement and marks them free. This is to indicate that the sensor had a free line of sight to the point where the measurement was made. A 40 cm free line of sight is created if the sensor did not measure anything or if the measurement is above 40 cm. The cells along the line are found using Bresenham's algorithm [bre16]. The threshold of 40 cm was chosen because at higher distances the variance of the sensors becomes to big for the measurements to be valuable. An example of how this process looks is shown in figure 4.1.

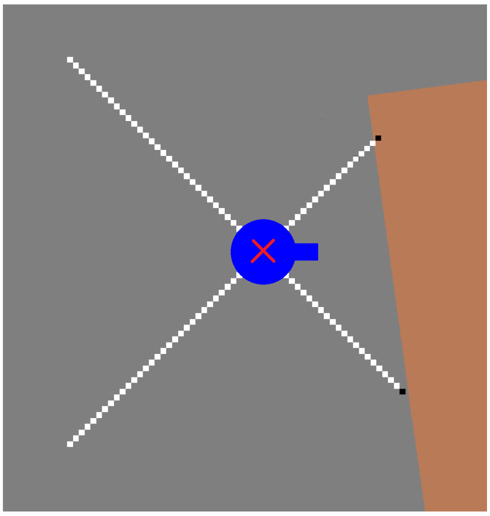

Figure 4.1: In the figure one can see a robot and an obstacle. The red X illustrates where the sensors are pointing. The obstacle has been observed by two of the sensors, while the other sensors measured nothing. The figure illustrates how the grid map would be updated according to these measurements.

Some features of the mapping:

- Filling inn unexplored area. The rotation of the sensor-tower leaves small unexplored gap between in the sensor lines from two consecutive measurements. The robot environment in this project does not contain small obstacles, so it is fairly safe to assume that the gap is actually free of obstacles. Therefore the mapping-module periodically searches for small unexplored areas and fills them inn by updating each cell. This makes the mapping process look smoother, and also simplifies exploration because such small areas could potentially be selected as targets for the robots (See section 5.3 for how target points are selected).

- The mapping controller adds more space to the map of measurements are located outside it. This makes it so that it is not necessary for the user of the program to know the size of the environment when mapping.
- Omitting robot measurements. If a measurement is within 10 cm of the position of any robot, the mapping controller does not use it. This was due to a problem that happened when a robot observes another robot. The problem is illustrated in fig 4.2. The navigation module is going to make the blue robot back away due to the measurement made by the green robot.

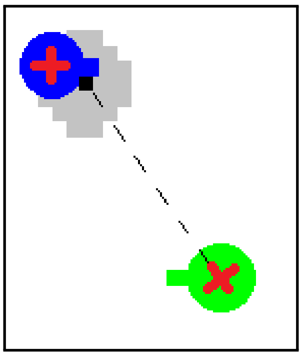

Figure 4.2: The figure illustrates that the robots observing each other can lead to problems.

4.3 Software design

The mapping module has very little functionality and so it's structure is simple compared to the other modules in this thesis. Figure 4.3 shows an overview of the components of the mapping module. The "measurement handler" objects are used for finding the position of the robots and the IR-measurements in the map. The mapping thread the uses these positions to plot the measurements in the map. The "map clean-up"-thread fills in small unexplored gaps in the map.

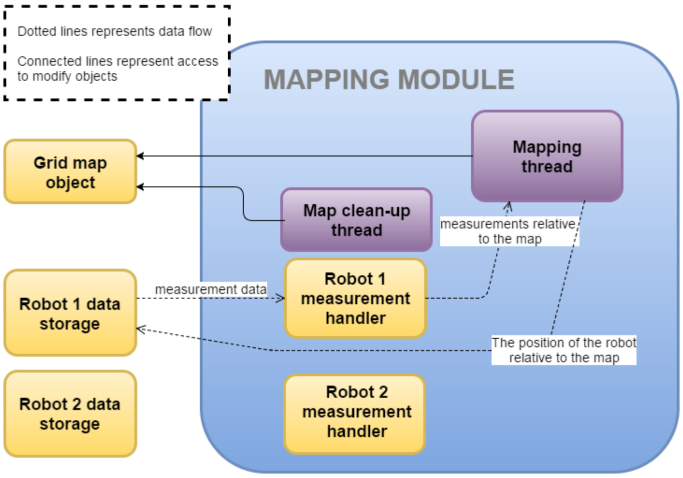
Dotted lines represents data flow
Connected lines represent access to modify objects
MAPPING MODULE
Grid map object
Mapping thread
Map clean-up thread
measurements relative to the map
Robot 1 data storage
measurement data
Robot 1 measurement handler
The position of the robot relative to the map
Robot 2 data storage
Robot 2 measurement handler

Figure 4.3: An overview of the components of the mapping module and how they interact.

Chapter 5 Navigation

5.1 Overview

In order to make the robots explore the environment in an effective and safe manner four different tasks needs to be done.

- Selecting the best target for an idle robot.
- Finding a path between the robots position and the target.
- Controlling the robot along the path
- Avoiding collisions with walls and other robots

The navigation module was designed and implemented so that it handles all these tasks simultaneously for each robot. The following sections of this chapter takes a closer look on how it does that.

5.2 Restricted and weakly restricted cells

Restriction and weak restriction is an idea that was used to make sure that the robot does not collide with obstacles in the map. This idea affects almost every part of the navigation module and therefor it makes sense to begin this chapter by explaining what it is.

Restriction and weak restriction is used to mark an area of the map that is close to an occupied cell. The work of keeping track of this area is actually done by the mapping module. The mapping controller makes sure that all cells within a 15 cm radius of an occupied cell are marked restricted, and all cells within 25 cm are marked weakly restricted (See fig 5.1).

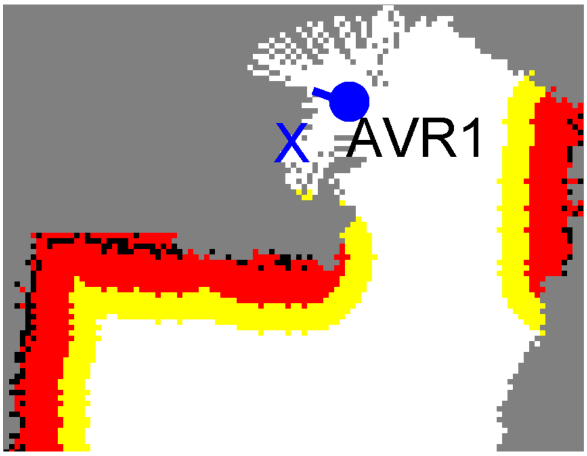

Figure 5.1: The figure is from a test run. It illustrates the restricted area with red color and weakly restricted area with yellow color.

The primary idea is that robots should not be allowed to operate in restricted parts of the map, as they may be in danger of colliding with the obstacle causing the restriction. If the robots end up in restricted area, special procedures should be followed to guide the robot out to a free area. To minimize the occurrence of such events the robots should preferably no be close to the restricted area, as they are not 100 percent accurate in their movement. An extra layer of weak restriction was added around the restricted area. The idea of weak restriction is that the robot should not enter a weakly restricted cell, unless no other option is available.

5.3 Target-selection

5.3.1 Overview

The target-selection method is used every time a robot is idle to select a target-point in the map that the robot should drive to. The standard approach in multi-robot exploration is to find frontier regions in the map and evaluate them according to some utility-function [Yam98]. In this thesis a somewhat simplified version of this strategy was used.

The algorithm that was implemented finds all explored, non-restricted cells in the map (weak restriction is ok) that has an unexplored neighbor-cell. To simplify the target selection, a big chunk of the cells are filtered out by only selecting frontier-cells that are 5 cm away from each other. Figure 5.2 shows an example of potential targets.

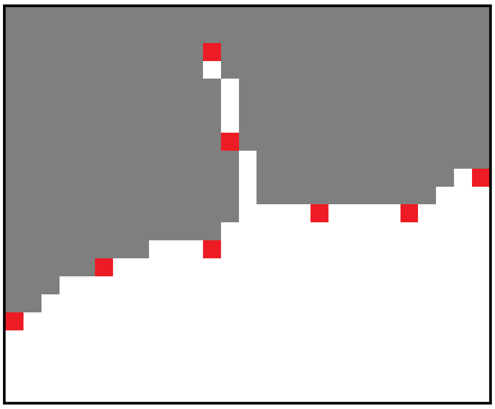

Figure 5.2: The figure shows a zoomed-in part of the map to illustrate how potential targets are selected. Grey is unexplored cells, white is free cells, and red is the cells selected for potential targets.

Next the utility for each potential target is computed. The method selects the location with the highest utility and attempts to find a path between the robot and the location (See section 2.3 for path planning). If a path is found the target-selection is done. Otherwise the method attempts to find a path to the point with the second highest utility and so on. If no targets are reachable the robot simply has to wait, and the navigation-module will try to find a new target later.

5.3.2 The utility-function

Section 2.4 gives a very short description on utility theory. Utility was used because it is easy to understand and implement, although it requires a little bit of cleverness.

The goal when designing the utility-function was to make it so that it assigns high values to locations that the robots ideally should go to to make the exploration more efficient. To do this several attributes had to be selected for the utility. The plus and minus denotes how the attribute affects the utility.

- 1. + The unexplored area around the target point. Implemented by finding the number of unexplored cells around the target and multiplying that with the area of one cell. Variable: **area**.
- 2. - The direct distance between the robot and the location. Variable: **distance**.
- 3. - How near the target is to other target points. The goal of this attribute was to distribute the robots throughout the map. The following formula was created to implement this goal: $\sum_{1}^{nRobots} \frac{1}{distance}$, where distance is the distance between the target point and the target point of the other robot. Variable: **distribution**.
- 4. + Whether or not the robot has a free line of sight to the location. The goal of this was to make the robot avoid selecting targets that are nearby, but on the other side of an obstacle. Variable: **lineOfSight** = 1 if there are no unexplored or restricted cells between the robot and the target, 0 if not.
- 5. - Whether or not the target is inside a weakly restricted area. The goal of this was to preferably select free targets as opposed to weakly restricted targets. Variable: **weaklyRestricted** = 1 if weakly restricted, 0 if not:
- 6. - The angle the robot has to turn to point in the direction of the target point. The goal of this was to reduce how much the robots have to turn, because turning negatively affects the pose-estimate. Variable: **angle**

Each of the attributes was weighted by trial and error. The final result was:

```
Utillity = 0.2*area-3*distance-40000*ditribution+500*lineOfSight-
2000*weaklyRestricted-angle
```

5.4 Path-planning

5.4.1 Using A*

Path planning is used to find a path through the map from the robots current position to its target-point. For this task the A* algorithm was used. This was because it is very effective and simple to implement for searching through ha grid. Also the author was familiar with the algorithm from previous experience. For some short theory on the A* algorithm see section 2.3.

The algorithm was implemented so that it can traverse both directly and diagonally in the map. The traversed distance between to cells was set to 1 if they are directly connected, and 1.415 if they are diagonally connected. The heuristic cost was set to the direct distance in the map between the cell and the target cell. It should be noted that the diagonal traversed cost is the square root of two rounded up at the 3rd decimal point. Rounding up was important or else the heuristic cost would not be consistent.

Restriction and weak restriction is incorporated into the A* algorithm to make the algorithm avoid generating paths near obstacles. Restricted cells are never added to the open set. This guarantees that the path will contain no restricted cells. Weakly restricted cells are added to the open-set, but their traversed cost is 10 times as high as completely free cells. This ensures that the path will only go through a weakly restricted area if no other option to reach the target exits, or if the alternative paths are over 10 times as long.

5.4.2 Testing the A* implementation

Some testing was done to see if the implementation of the A* algorithm was successful. Figure 5.3 a) illustrates the closed set after a path has has been found between two points. The figure shows that when the algorithm is unable to explore in the direction of the target point it searches through the map as a BFS-algorithm would. However as soon as it rounds the lower corner it searches in a direct line to the target point. This indicates that the algorithm has been successfully implemented.

Figure 5.3 b) shows the resulting path from the search. From the figure it looks like the algorithm found the optimal path without searching through a large part of the map. It can be seen that the path goes a little distance away from the restricted area so the idea of increasing the traversed cost for weakly restricted cells seems to have worked. It can be seen that the algorithm searches around the target point a bit. This is because the target point was weakly restricted, as it is close to a restricted area, so the algorithm hesitates to open the target-node. All in al it seems that the use of the A*-algorithm has been successful.

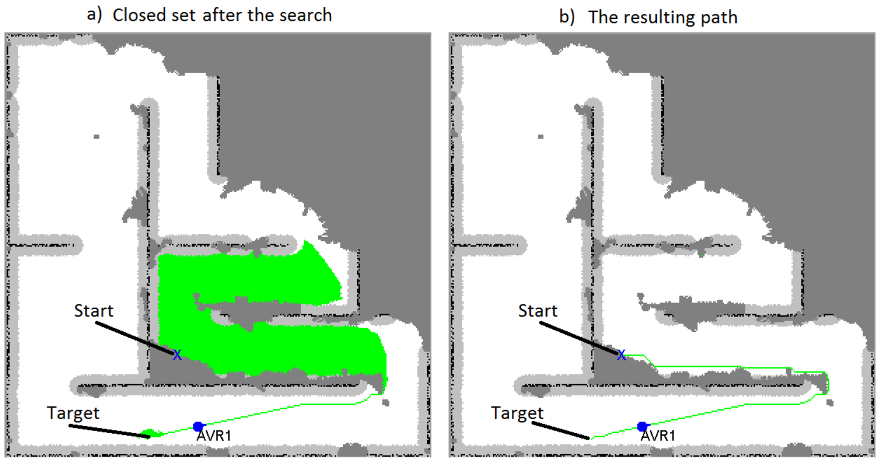

Figure 5.3: A test-run with with the A* star algorithm. a) the map-cells in the closed set are marked with green color. b) the cells in the resulting path are marked with green color.

5.4.3 Complexity

In the test it took approximately 14 seconds to find the path. When the area between the starting point and the target point is completely free the path is found instantaneously, however if the algorithm has to search through the entire map in can take a long time.

The implementation has a time complexity of $O(n^2)$ where n us the number of explored reachable cells. With A* the worst-case time complexity is almost never an issue unless the target is unreachable. The environment in this test was 6x6 meters (the simulator was used). The cell size was 2x2 cm meaning that the whole map consists of 90 000 cells. If the whole environment is explored and the target-point is unreachable, the time for the algorithm to finish is about 6-minutes.

5.4.4 Searching from target to start

In the finished program the path-planning searches for a path from the target to the robot position as can be seen in figure 5.3. It was experienced through testing that the robots target points are often not reachable for the A* algorithm. Figure 5.4 illustrates how this happens.

If the algorithm starts from the position of the robot it can not reach the red cell by only searching through the free white cells. The A* algorithm does not stop searching until it has checked every cell in the map, so with a large map a lot of time

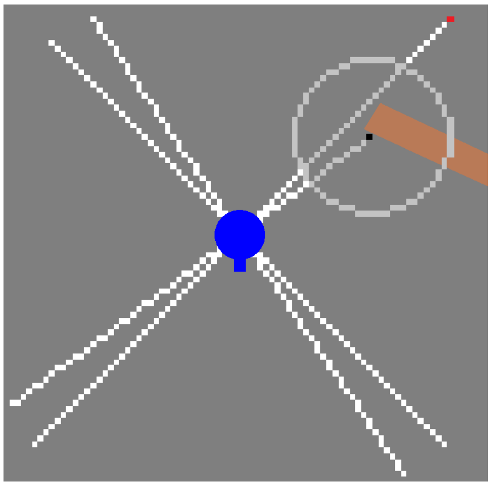

Figure 5.4: This figure illustrates how a target point (the red cell in the map) can easily be obstructed by restricted cell. The brown area represents an obstacle in the environment which the robot has just observed (the black cell in the map). This causes all cells within the light gray circle to become restricted.

would be wasted on trying to reach unreachable targets. This problem was solved by starting the algorithm in the target point, and making it search for a path to the robots position.

5.5 Robot control

The goal of the robot-control system is to control a robot along a given path in an effective manner. The robots only moves in straight lines so the path had to be divided into straight segments. This is done by selecting and storing way points along the path.

The behavior of the robot-control is simple. For each of the robots the robot-controller has stored a queue of way points along the path from the robot to it's current target. When a robot has completed a command it sends a special message to the server. The navigation then checks the robots position and orientation and computes and sends the correct command needed to reach the next way point.

5.5.1 Selecting way points

The selection of way points had to be done smartly because if the distance between way points is to long there is a danger of cutting curves in the path, causing the robot to collide with walls. However if the spacing alway is short the robot ends up stopping a lot, making it unnecessarily slow.

A special algorithm was designed to avoid these problems. The algorithm works by computing the fit of a straight line along the path. The line is made longer and longer until the fit goes below a threshold. The endpoint of the line is then selected as a new way point and the process is repeated from that point until the algorithm reaches the last point along the line. The idea is that if the path is fairly straight the line can become quite long before it no linger fits sufficiently to the path, however if the path is curved the fit of the line will quickly become bad.

Figure 5.5 illustrates the resulting way points after using this algorithm on a path.

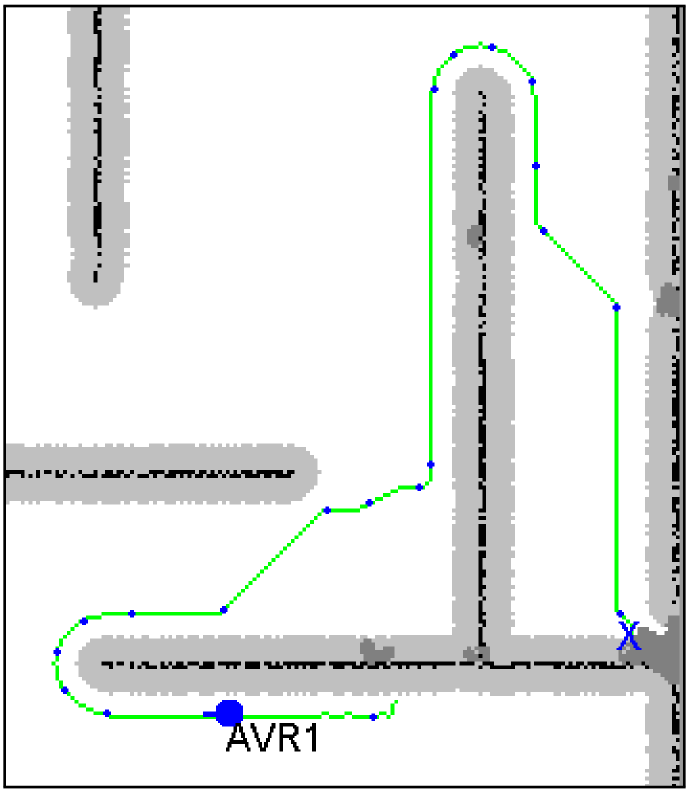

Figure 5.5: Results from a test run to illustrate how the algorithm performs. The cell containing the way points have been visibly enhanced with blue color.

5.6 Collision avoidance

If the map is inconsistent a path that is collision-free in the map may not be collision free in the robot-environment. Also the robots may collide with each other. Therefore a collision and detection and handling system was created.

5.6.1 Avoiding wall-collisions

To keep the robots from colliding into obstacles that can be measured bu the IR-sensors the collision-system attempts to keep the robots away from the restricted area in the map. If a robot is found to be in a restricted cell, a new thread handles the navigation of that robot until it has escaped the restricted area. First the thread check if the map-cell that the robot would occupy if it simply reverse 10 cm is also restricted. If not it orders the robot to reverse 10 cm, as this is the safest thing to do in most situations when the robot is near a wall. In some cases however that rear cell will also be restricted. Then BFS is used to find the nearest unrestricted cell and a command is sent that makes the robot go to that cell. The BFS-algorithm was easy to implement because it is simply A* without sorting, and the A* algorithm was already made. When the selected command is completed the thread checks if the robot is still in a restricted area. If it is a new command is sent and so on.

At the end of the collision handling all the robots way points are deleted so that it does not continue on the obstructed path. This causes the navigation module to select a new target point for the robot.

5.6.2 Avoiding robot-collisions

The collision module continuously check if if robots are about to collide. This is done by first checking if the distance between any two robots is less than 50 cm, then if one of the robots is moving, and then if the moving robots current trajectory goes nearer than 25 cm of the other robots current position. This is done by checking if a line along the robots between the robots position and its current way point intersects with a circle with radius 25 cm at the other robot's position. if an intersection is found, a new thread is created to handle the robot until it has a free movement path.

Typically typically collision-avoidance is done by pausing the colliding robot until the other robot has cleared its path. But sometimes the robots may be on collision course with each other and the other robot may already be waiting for the first robot. Therefor the collision handler check if the other robot is already in a collision and with whom. If a mutual collision is found the first robot is ordered to move to the side. This will cause the other robot to continue moving because its path is no longer obstructed. When the other robot has passed the first robot can resume operation. In some narrow areas it might not be possible for the first robot to step aside. In that case all way points are cleared so that a new target point is selected.

The protocol for robot collision avoidance is very simple, and in a lot of situations the robots may stay locked. Therefor a timeout is implemented so that if the collision handling last more than 20 seconds it is aborted and new targets are selected for both robots.

5.7 Implementation

The navigation module was the most complex module that was created. Figure 5.6 shows an overview of the basic components of the mapping module and how they interact. Yellow represents basic objects and purple represents threads.

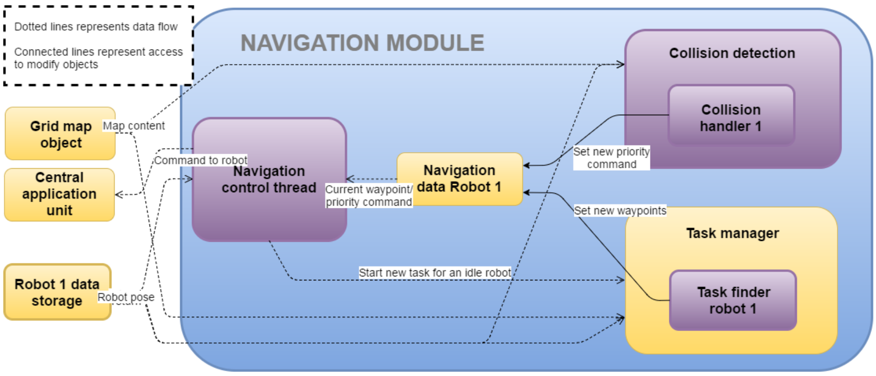

Figure 5.6: an illustration of the components of the navigation module and their relations.

5.7.1 Creating a new tasks

Creating a new task means finding a new target and generating way points between the robot and the target. The navigation controller thread initiates this process if a robot has no more way points. The process of finding the target and way points is done in a new thread because searching for a path may take a long time. When the task-finder thread is done it stores the way points in a special data object for the specified robot and terminates. The navigation controller has access to the data object and can extract the new way points. The process of this thread is presented by algorithm 5.1.

Algorithm 5.1 The the target selection algorithm

```
frontier_points = map.getFrontierPoints()
potential_targets = selectSpreadPoints(frontier_points)
assigned = false
while not assigned {
    select the target with the highest utility from potential_targets
    find a path from robot to the target
    if no path was found {
        remove the target from the list of potential targets
    }
    else{
        find way points along path
        store the waypoints int the navigation robot
        assigned = true
    }
}
```

Chapter 6
Results and discussion

6.1 Testing mapping and navigation with the real robots

The goal of this test is to see how well the mapping module and the navigation module performs with the real robots. For this purpose an 215 cm by 215 cm test area was made, and the Arduino- and AVR-robot were used. Unfortunately one of the wheel encoders on the Arduino was damaged so it was unable to generate a good estimate of its position. It was therefore decided that the Arduino-robot would not execute the orders it received from the server. Despite of this issue, it sent its measurement and was connected to the system in a normal manner. Figure 6.1 shows the initial setup for the test. The system was ran 3 times and both the robots and the map on the screen were recorded.

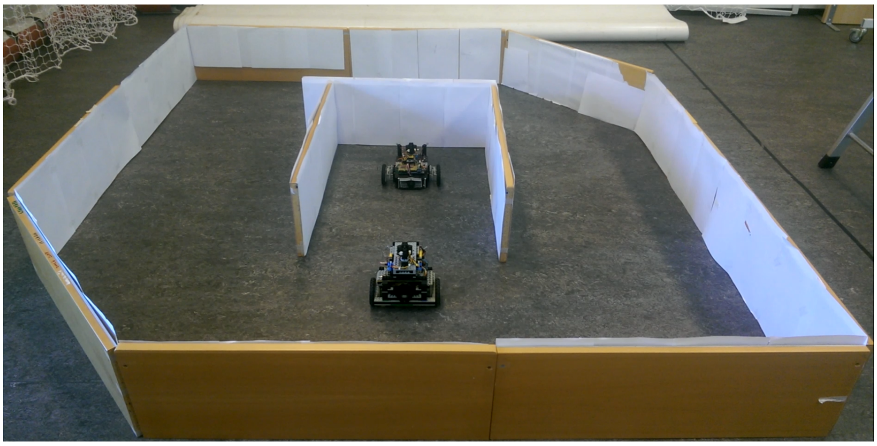

Figure 6.1: The set-up for the full system test

6.1.1 Discussion of the mapping-module

Figure 6.2 shows the finished maps from the three tests, plus an extra map that was created in during the demonstration of the project.

Here are some of the observations discussed in bullet points:

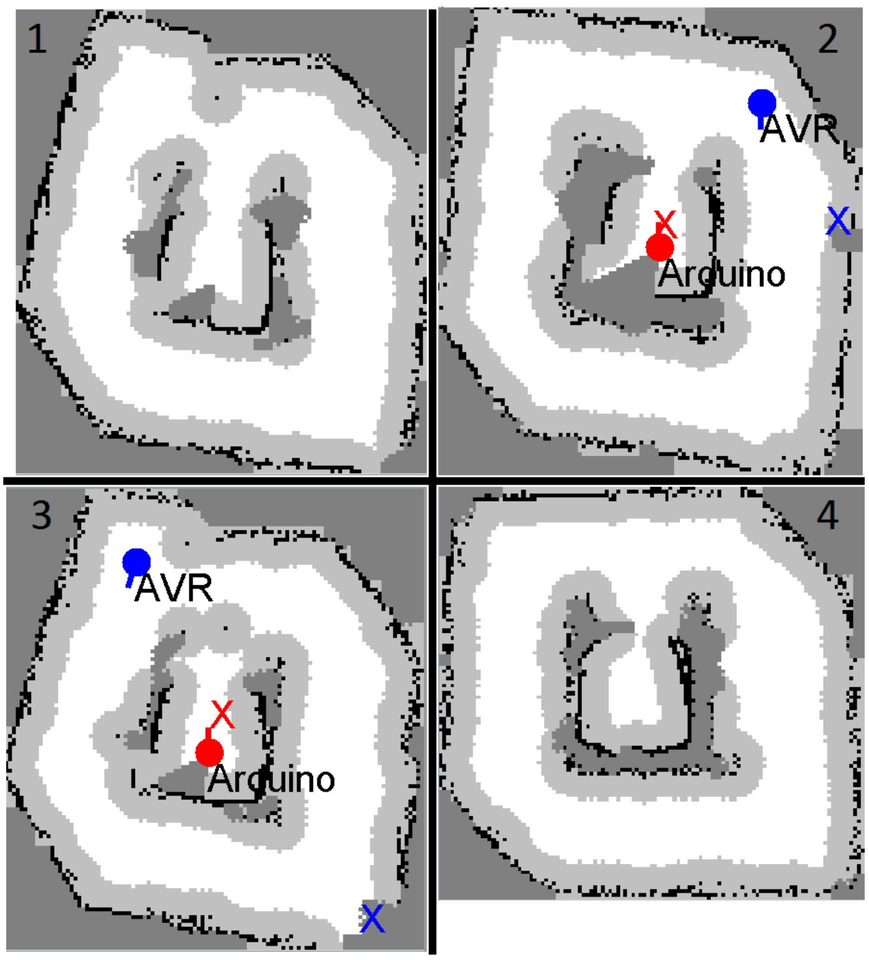

Figure 6.2: 1-3: The resulting map from each test. 4: The resulting map from one of the runs from the demonstration.

- Map 2 and 4 has almost the exact same structure as the real environment, while 1 and 3 shows some degree of inconsistency. The inconsistencies are caused by an error in the robot's pose estimate. In both test test 1 and 3 the robot traveled clock-wise around the map. When it reaches the upper left area it believes that is has rotated more that it actually has which causes the map to bend inwards. Ese 2016 covers inconsistencies in the estimate in much greater detail.

- It seems from most of the maps and especially in number 4 that the inner square is much bigger on the outside than on the inside. This is not an error from the mapping, but is due to a bias that was found on the sensors, but not corrected. The bias makes the measurements shorter which makes walls seem as if they are closer to the robot than they really are.

– In the first three tests there is an unexplored area near the Arduino-robot. This was found to be caused by a malfunction on the sensor covering that area.

These results were expected as the mapping module was not designed to make corrections to the estimated pose. The resulting map was not to bad, but for a larger environment one would expect the errors to increase further, and the map to become less accurate.

6.1.2 Discussion of the navigation module

Figure 6.3 illustrates approximately where the path of the robot went in the first test run. Video real_robots_test_1 shows the movement of the robot in that test. real_robots_test_2 and -3 documents the other two tests.

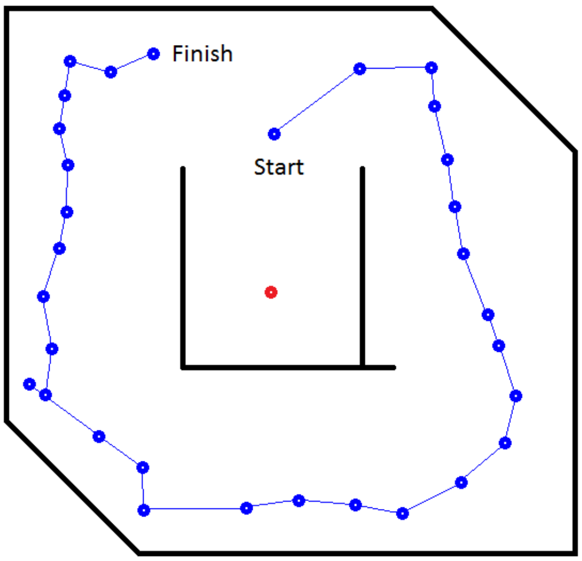

Figure 6.3: An approximate plot of the path the robot took in the first test.

Some observations:

– The trajectory of the robot follows the available path in the environment in a quite efficient manner.

– The distance between each new target is quite small. This is because the robot is only ordered to go to areas that are already explored and it can only see 40 cm into the distance.

– The robot collided with a wall in the lower left corner of the figure. This can be seen in the video.

The last observation requires some discussion. Wall collisions are avoided by checking if the robot is inside restricted area. The restricted area is 15 cm around each wall, while the corner of the robot is 11 cm from its center. With the robot moving at around 15 cm/s the robot must receive a command to stop within 0.27 seconds to not collide with the walls. Bluetooth-messages may be lost and then retransmitted so the reaction time of the system may be a bit to slow. More narrow objects can stay hidden in the sensors blind spots, which may leave zero time to react.

One solution to the problem would be to increase the radius of the restricted area, but this solution has its problems as shown by figure 6.4. The figure shows 3 corridors of different width. In a) there is no problem, the robot can follow the path without any interruptions. In b) there is no completely free area which means that the path may go arbitrarily close to the restricted area. For the given path in the figure the robot will often end up slightly within the restricted area and the collision detection system will trigger, interrupting the robots operation. The rightmost corridor c) is so narrow that no path will ever be generated through it.

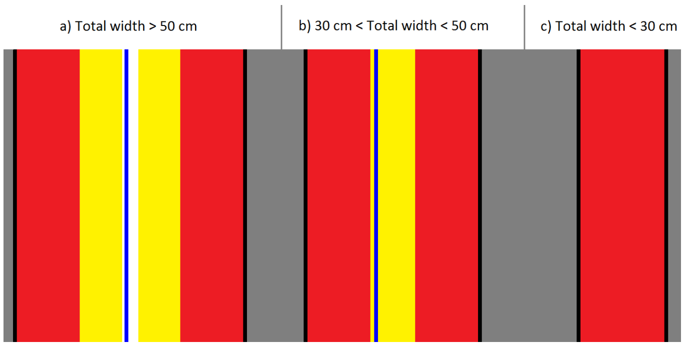

Figure 6.4: The figure illustrates the main problem of using restricted area to avoid wall collisions. Red is restricted area, yellow is weakly restricted area, and blue is path.

Already corridors of 30 cm width are completely blocked, and ideally the width should be over 50 cm. Increasing the restricted area would further limit the movement of the robots, so it may be better if the method for detecting collisions with obstacles is modified.

6.1.3 Earlier testing with two robots.

Some testing was done at an earlier stage in the process before the encoder on the Arduino-robot was damaged. These tests showed that the system was able to utilize both robots when exploring the map. Unfortunately there were some issues with the estimated pose of the Arduino-robot'so the results were not documented.

6.2 Comparing the simulator with the real robots

This test was performed to see if there is any difference between running the system with the real robots or on the simulator. A map of the environment from the first test was made for the simulator. For this test only one robot was used, just to give it more area to explore. The initial setup for both the real world and the simulator can be seen in figure 6.5.

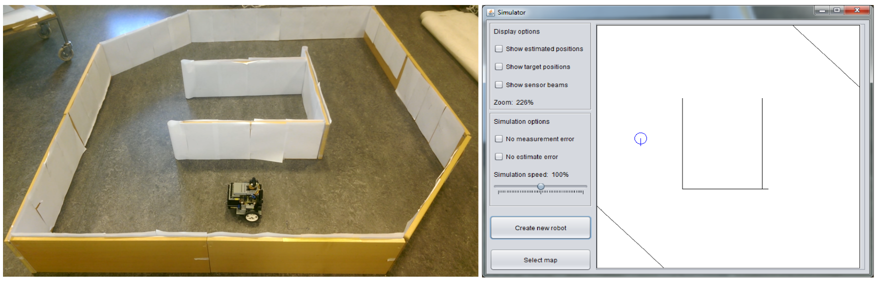

Figure 6.5: The initial set-up for the comparison test between the real world and the simulator.

Figure 6.6 shows the finished result with both the real robot and the simulator. Video testing_the_accuracy_of_the_simulator shows the full mapping process for the whole test.

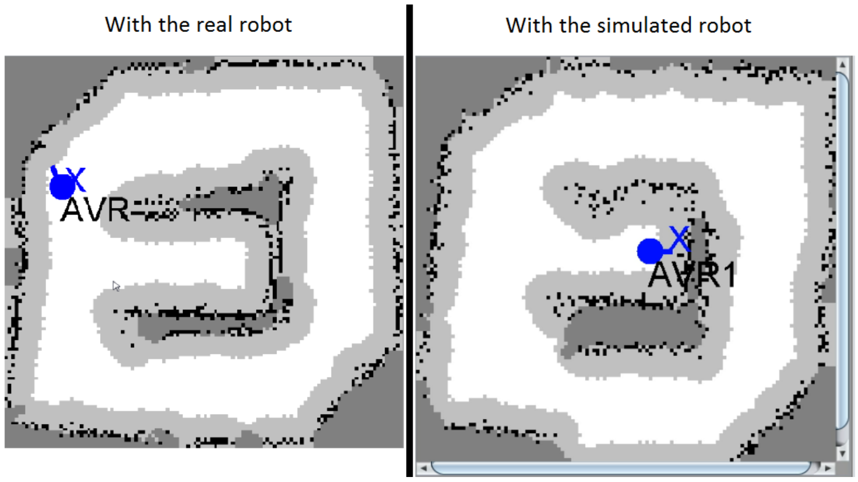

Figure 6.6: The two results from mapping with a real robot versus a simulated robot

- The map looks quite similar. The measurements are denser in the map from the real robot. This is because the simulated robots move a bit faster.
- The error in the map seems to bee of similar magnitude. This indicates that the simulator is able to quite accurately simulate both measurement noise and estimate error.
- The simulated robot took a much longer path, than the real robot. This is only because the simulated robot missed a corner in the map, and had to go back later. That is expected to happen whether or not simulated robots are used.
- From the video it looks like both robots are behaving almost identically.

6.3 Testing mapping with multiple robots in the simulator

In the test with the real robots, one of the robots were stationary. The results in section 6.2 have shown that the simulator can generate quite similar results as the real robot, so it was used to test how the mapping module performs with multiple robots. For this test the same simulator-map as can be seen in fig 6.5 was used and two robots were added.

The test was not recorded as one run with the simulator was already recorded. The resulting map can be seen in figure 6.7.

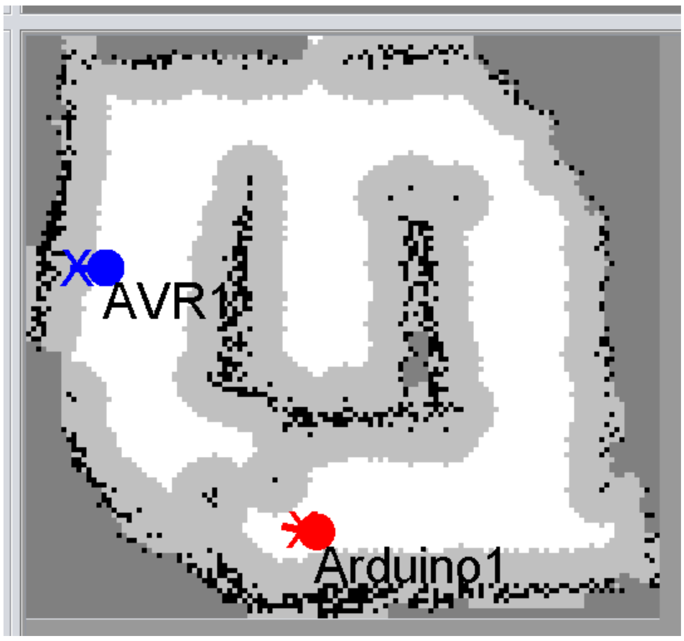

Figure 6.7: The result from mapping with two robots

The mapping module has two issues with multiple robots:

1. The pose-estimate-error of the two robots grows in different directions. With one robot th map may be inconsistent, but with multiple robot the map quickly becomes completely unusable.

2. The robots can measure each other. In the figure there is a restricted area in front of the red robot. This actually comes from measurements of the red robot made by the blue robot. When there is no error in the position of the robot, the mapping-module is able omit measurements of one robot from another. This is however no longer possible if the error becomes to great. As the map shows

this is a problem because it looks to the navigation module like the corridor between the two robots is blocked, while in reality it is not. When there is no error in the pose of the robots, the mapping-module eliminates measurements of other robots, but that is no longer possible when the error grows beyond a certain limit.

Even though the simulator was used, the same problems should be expected with the real robots. The only difference is the for example the AVR robot is not able to measure the Arduino robot as the Arduino robot is not high enough.

6.4 Testing management of multiple robots without noise in the simulator

This test is performed to see how well the navigation-module handles a higher number of robots. The goal is to see if the time it takes to create a map of an environment improves when adding more robots, and also if the navigation module is able to make the robots not collide. It may seem questionable to judge the performance of the navigation module based of a noise free simulation. In the current state of the system this is true, but if it was possible to remove the error, for example with a SLAM algorithm, this test would be valid.

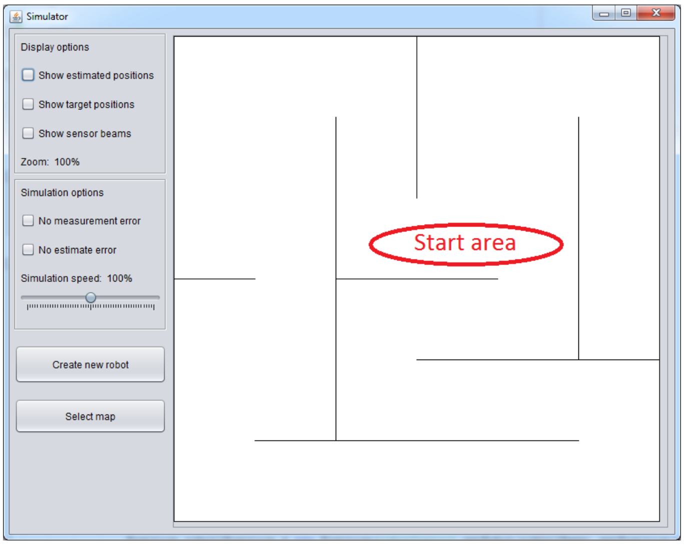

Figure 6.8: The figure shows the environment that was used for the multi-robot test.

This test was done by performing a mapping the same simulator map with an increasing number of robots from 1 to 6. The map is shown in figure 6.8. It has a height and width in the simulator of 6 meters. Mapping was done three times with each number of robots. All the robots were started within the red circle in the figure. The average mapping time is shown in table 6.1.

The table shows that the time it takes to explore the environment decreases with the number of robots used. However the decrease is not as consistent as expected. If the navigation-module was performing optimally one would expect to se an almost exponential decrease in the mapping-time. The topography of the simulated environment may be some of the cause for the result. I expect that if a completely open map was used there would be a different result, but unfortunately it was not

Table 6.1: The result from testing exploration with multiple robots

Number of robots	Average mapping time in minutes
1	21:53
2	11:21
3	9:28
4	9:42
5	7:15
6	6:06

enough time to try that.

It was observed that the robots had to do some back-tracking which took extra time. The use of an utility function for selecting target points does not give exact control over the robot behavior, and so it is difficult to use it for optimal exploration.

In some of the runs a couple of robots would collide. The collision avoidance system was not designed to be foolproof so some collisions should be expected, but it seemed that the system malfunctioned on one or two occasions and was making errors it should not have made.

6.4.1 Single test with a high number of robots.

The maximum number of robots that can be connected to the server-program is 10 [AR16]. It was tested to see how the system would perform with the maximum number of robots. A recording of the test can be seen in video: ten_robots_simulator.

In test went well for a long time, but then both the program and the simulator started lagging. It was found that the CPU was running at 100% capacity. This happened when a high number of robots simultaneously started to look for target points far away. Most likely the CPU usage was due to the path-planning algorithm being ran parallel for multiple robots.

The time of this mapping was 3:20 minutes and as an interesting side note none of the robots collided.

Chapter 7

Conclusion

7.0.1 Simulator

The result from testing shots that the simulator is able to simulate the real robots quite accurately. Additionally both the mapping module and navigation module were developed using the simulator and did not have to be modified when first testing with the real robots. In conclusion both the design and implementation of the simulator were completely successful.

7.0.2 Mapping

Results have shown that the mapping-module works as intended, but that its simple design was not sufficient. The mapping module is a able to adequately map small areas using one robot, but in larger areas or with multiple robots, growing differences and errors in the estimated poses makes the map unusable. The use of multiple robot is a key element of the project and it is necessary that the mapping module is able to generate consistent maps. Either the robots must become better at estimating their own pose (for example through more accurate sensors) or the mapping-module must be improved or replaced with a decent SLAM-algorithm.

7.0.3 Navigation

The design and implementation of the navigation-module has for the most parts been successful, but there were a few flaws. The use of restricted area worked great for path-planning, but not that great for detecting collision between a robot and a obstacle in the map. Also the results from the multi-robot testing indicated that there might have be a slight error in the system for detecting collisions between robots.

When it came to path-finding and exploration the mapping-module performed much better. The utilization of the robots was not optimal, but the module demonstrated that the time it took to explore an area was greatly reduced when more robots were connected. For even better performance the system may need another method for target selection, as the utility function does not give the developer absolute control over the robot behavior.

It should be said that the performance of the navigation-module relies the map being consistent. The module will make a best-effort with inconsistent maps, but it can not guarantee to be able to explore the whole environment.

Chapter 8 Further Work

8.1 Simulator

– After the simulator had been created, the behavior of the sensor-tower changed [Ese16]. It now behaves differently based on how the robot moves. The new behavior should be implemented into the simulator.

If the robot behavior changes in the future the simulator should also be updated.

8.2 Mapping

The error in the estimated pose has a crippling effect on the current system. I believe that either the robots must be equipped with better sensors, so they can more accurately estimate their position or a SLAM method must be used.

8.2.1 Using a SLAM method instead of the existing solution

It was a bit difficult to learn anything valuable from the research, because the SLAM-field is both big and complicated. I would recommend trying to find an existing solution that is meant for both on-line and multi-robot SLAM. This algorithm [mrs16] seems like it fills both criterias going by the information on the website, however it still requires a lot of modification. The web-site says that is uses a LIDAR-system, and that the robots communicate with each other using an Ad-Hoc network.

8.2.2 Improving the existing solution

Here are some improvements that could be made to the current mapping-module if it is decided not to use SLAM.

– The offset between the tower and the center of the wheels is not handled. The robots send this information in the hand shake and the functionality only has to be implemented into the mapping module.

– It should be possible to create some algorithm that extracts walls from the measurements, so that the obstacles does not have to be represented by scattered dots. The RANSAC-algorithm seems like a promising start [ran16].

– If the cell size is large it would be natural wait until several measurements have been made of the cell before it becomes occupied. This is not a big issue as it is possible to have a quite high map-resolution for the small environments that are currently used.

– Scan-matching can be used to fix the pose error of one robot, and is not extremely difficult to implement although changes might have to be made to the map-representation. Using scan-matching might improve the performance of the mapping module [sca16].

– It would be great if the user did not have to specify the position of each robot at the start of the mapping. Scan-matching might be a helpful tool for this problem as well.

8.3 Navigation

– Add a time-out on the path planning so it does not eat up all of the CPU-resources if no path is found.

– The target selection-method should be modified so that it compares the direct distance to the target with the length of the actual path before finishing.

– The detection and handling of collisions between two robots must be improved.

– Collision detection with obstacles in the map should be done in a more advance way. Instead of checking if the robot is inside some area, the detection system should check if the robot is actually on collision course. This is a bit more difficult due to the width of the robot.

8.4 Other improvements

The collision detection system suffered a bit from the robots having large blind spots. I would recommend that the sensors are reorientated so that they all point forward at evenly distributed angles.

References

[AR16] Thor Eivind Svergja Andersen and Mats Gjerset Rødseth. System for self-navigating autonomus robots. Master's thesis, 2016.

[Bak08] Jon Martin Harstad Bakken. Bygge og programmere ny legorobot, 2008.

[bre16] Bresenham's algorithm, 2016.

[Ese15] Erlend Ese. Fjernstyring av legorobot, 2015.

[Ese16] Erlend Ese. Real-time programming on collaborating mobile robots. Master's thesis, 2016.

[Hal12] Ulvin Halvorsen. Collaborating robots. Master's thesis, NTNU, 2012.

[hec16] Ros hector mapping, 2016.

[Hom13] Trond Kåre Homestad. Fjernstyring av legorobot, 2013.

[Laz13] *Multi-Robot SLAM using Condensed Measurements*, 2013.

[lin16] Line intersection, 2016.

[Mag08] Trond Magnussen. Fjernstyring av legorobot, 2008.

[mrs16] Cg mrslam source code, 2016.

[ran16] Ransac wikipedia, 2016.

[ros16] Ros gmapping, 2016.

[sca16] 2016.

[Skj04] Håkon Skjelten. Fjernnavigasjon av lego-robot, 2004.

[SR14] Peter Norvig Stuart Russel. *Artificial Intelligence A Modern Approach*. Pearson, 3rd edition, 2014.

[ST05] Dieter Fox Sebastian Thrun, Wolfram Burgard. *Probabilistic Robotics*. The MIT Press, 2005.

[Sta16] Cyrill Stachniss. Youtube-page for slam-lectures, 2016.

[Syv06] Bjørn Syvertsen. Autonom legorobot. Master's thesis, NTNU, 2006.

[Tus09] Janicke Selnes Tusvik. Fjernstyring av legorobot, 2009.

[Yam98] Brian Yamauchi, editor. *Frontier-Based Exploration Using Multiple Robots*, 1998.

Appendix

Appendix A

Overview of the contents on the CD:

- **javadoc.zip**: The dokumentation of the code. Navigate by opening index.html.
- **simulatorManual.pdf**: A user manual for the simulator.
- **SSNAR.zip**: The source code of the project. The software developed in this thesis was the packages: general, map, mapping, navigation, simulator.
- **videos**: The videos from all the testing.
  real_robots_test_1-3 is from the testing in section 6.1.
  testing_the_accuracy_of_the_simulator is from the test in section 6.2.
  ten_robots_simulator is from the test in section 6.4.
- **Thon2016**: A copy of this thesis