NTNU – Trondheim
Norwegian University of
Science and Technology

Hardware platform for a Multi-Robot System

Adam Leon Kleppe

April 24, 2013

ASSIGMENT

Research, design and implement a multi-robot system for small robotic vehicles. It is encouraged to design the system to have low cost, weight and size, and discouraged the use of heavy computing and complex sensors. The following items should be considered:

- Study methods and current implementations of multi-robot systems.
- Research requirements and constraints for the multi-robot system, and propose a solution for the system.
- Propose a hardware platform for the multi-robot system, and investigate the different hardware components required to implement the platform.
- Conduct experiments and simulations supporting the possibility of the implementation.
- Within the limitations of available time, create a prototype implementation.
- Conclude your findings in a report, and evaluate the plausibility and suitability of implementation.

SUMMARY

This report presents a multi-robot system and the plausibility of it being successfully implemented. The purpose of the presented system is to look at how simple a multi-robot system could be and still have cooperative properties. The system will therefore have many constraints, and discourage the use of heavy computing and complex sensors.

The full multi-robot system is not covered in this report, only the hardware platform, and obstacle detection and positioning system. These aspects are thoroughly discussed, and proposed solutions for these aspects are presented.

A set of hardware, protocols and algorithms are presented, all whom which may be implemented on the hardware platform, and solve possible problems presented in this report. The report also presents experiments, tests and findings that supports the plausibility of successfully implementing the multi-robot system.

The plausibility of the presented system is questioned and discussed, and it is concluded that the system may be implemented.

CONTENTS

Contents **vii**

List of Figures **ix**

1 Introduction **1**
1.1 Simple overview of the system 2
1.2 Organization of this report 3

2 Problem description **5**
2.1 Main problems 5
2.2 Constraints 6
2.3 General Methods 7
2.4 Proposed Solutions 8
2.4.1 Obstacle detection 9
2.4.2 Localization 9

3 Hardware Components **11**
3.1 Introduction 11
3.2 Microcontroller 12
3.2.1 Arduino Mega 13
3.2.2 UC3-A3 Xplained 14
3.2.3 Other alternatives 15
3.3 IR-laser range sensor 16
3.3.1 Sharp GP2Y0A21YK IR Range Sensor 16
3.4 Omniwheel 20
3.5 IMU 20
3.5.1 MPU6050 21
3.5.2 HMC5883L 21
3.6 Communication 23
3.6.1 XBee 23

3.7 Ultrasonic sensor 24
3.7.1 400SR100 and 400ST100 24

4 Obstacle Detection **27**
4.1 Introduction 27
4.2 Previous work 28
4.2.1 SLAM 28
4.2.2 Particle filters 30
4.3 Inverted particle filter 32
4.4 Implementations 35
4.4.1 Scenarios 36

5 Relative positioning **41**
5.1 Introduction 41
5.2 Measuring relative positions 43
5.3 Ultrasonic sensing 47
5.4 Protocol 49
5.4.1 Multidimensional scaling 51
5.5 Synchronized clocks 53

6 Discussion **57**
6.1 Problem description 57
6.2 Hardware 57
6.3 Obstacle Detection 58
6.4 Relative positioning 59
6.5 Overview 59

7 Conclusion **61**

A Appendix **63**
A.1 MPU6050 Breakout board 63

Bibliography **65**

List of Figures

1.1 The prototype of the robot built for this project 3

3.1 The Arduino Mega 2560 viewed from the front. Image provided by arduino.cc 13
3.2 The UC3-A3 Xplained viewed from the front. Image provided by atmel.com 14
3.3 The Arduino Due was recently released. Image provided by arduino.cc 15
3.4 The Sharp GP2Y0A21YK IR Range Sensor is the range sensor used on the robot to detect obstacles. Image provided by sparkfun.com . 16
3.5 The set-up of the experiment. The blue lines is the field of view of the laser. At point 1 the Arudino Mega can be seen. At point 2 the laser mounted onto the servo. It is hard to see, but the motor is under the cardboard box, and the laser is behind it facing the various objects. 18
3.6 The contour map created by the IR-laser. The contours represents if the set-up on Figure 3.5 is viewed from the right, mirrored around the horizontal axis. The laser is situated in (0, 0) and facing in the positive X-axis 19
3.7 The omniwheels used by the robot, provided by kornylak.com . . . 20
3.8 The omniwheel viewed from the top. The wheel can roll in the purple direction and slide in the green direction 21
3.9 MPU6050 is a microcontroller with gyroscope and accelerometer. Image provided by sparkfun.com 22
3.10 HMC5883L is a triple-axis magnetometer. Image provided by sparkfun.com 22
3.11 The XBee Series 2 is a module using ZigBee as a protocol for the wireless communication. Image provided by sparkfun.com 24
3.12 The 400SR100 and 400ST100 are respectively a ultrasonic receiver and transmitter in the 40 MHz range. Image provided by prowave.com 25

4.1 A graphical representation of the resampling wheel. Here each particle has a slice based on the weight ω. The ω_i is the weight of the particle i 31
4.2 The principle of the Inverted Particle Filter. On the top of the image: The robot is placed beside obstacles A, B and C. The image on the bottom represents the map of the Inverted Particle Filter at three different times. The first map is the initial map of the particle filter where all the particles are scattered, and they all are red (weight of 0.5). The second map is when the robot has rotated and gained information about the surroundings. Both obstacles A and B are detected and the particles are black (weight of 1). The third map is when the robot has all the information about the environment, and all the obstacles are detected. The pink particles are particles with weights close to zero. 33
4.3 To the left: A robot moving towards an obstacle. To the right: the map created by the Inverted Particle Filter. It can be seen that the particles represents the obstacle in front of the robot, and as the robot move towards the obstacle, the particles move towards the X (representing the robot) 34
4.4 A robot turning 4 radians, starting from $\frac{\pi}{4}$ heading at position (0, 0). 37
4.5 A robot moving in a circle, starting from (-25, -25) with a radius of 50 units, moving counter clock-wise. The map is set up with two obstacles. One with centre (50, 75) other with centre (100, 25. Particle count: 10 000) 38

5.1 Robot viewed from above. Here an angle θ between two light sources on robot B are measured by robot A. It can be seen that because robot B has an angle, robot A can not differ if robot B is positioned at B or C 46
5.2 Robot viewed from above. T represents a ultrasonic transmitter, and R represents a ultrasonic receiver 48
5.3 Transmitter viewed from the side. As seen a sound wave emitted is spread out into a circular wave. 48
5.4 A sound wave is pulsed from one robot and the three receivers on another robot measures the pulse 49
5.5 A graphical representation of the protocol. Each robot transmits a signal at the beginning of their time slot. The other robots then receives the signal. Since each robot has a given time slot the other robots know which robot sent the signal. 50
5.6 Three robots with their distances measured between each other. Equation 5.1 describes the distance δ. The distance measured from robot 1 to 2 is not the same as measured from 2 to 1 52

5.7 Problems with non-synchronized clocks. Robot A's clock is one second earlier then Robot B's. When Robot A transmits and Robot B receives, the real time between them is Δ_t, but the calculated difference is $\Delta_t - 1$. When Robot B transmits the calculated time difference is $\Delta_t + 1$ 55

A.1 The MPU6050 breakout board 63
A.2 The schematic of the MPU6050 breakout board 64

CHAPTER 1

INTRODUCTION

In recent years the there has been a huge growth in the need of autonomous robots for exploration. Rescue missions, mapping of unknown terrain, military operations and exploration of hazardous areas are only some of the tasks that would benefit from autonomous systems. There is no easy solution to this problem. A robot requires many sensors, good mobility and smart algorithms and controllers for it to be able to complete a mission on this scale. To further complicate things, the robot should also be flexible enough to complete many of these tasks, maybe even at once.

For a single robot to be able to accomplish this, it would require heavy computing, and many sensors. Therefore a new branch in solutions have emerged: multi-robot cooperative systems. Ten robots would be able to scan an area much faster than one robot would; ten robots would be able to lift heavier objects than one robot would; and ten robots would be able to survive longer than one robot would. A multi-robot cooperative system is more effective than one robot, but it also introduces more problems. A multi-robot system would need to be able to communicate and coordinate with each other.

There are many solutions to multi-robot systems [Burgard and Moors, 2000], [Davison et al., 2007], [Lucidarme, 2002], [Pagello et al., 1999], [Thrun, 2006], [Tuci et al., 2006]. While many of them lie within the definition of swarm systems, some are defined as an autonomous cooperative system. The robots in this definition are able to cooperate on tasks (e.g. formation, navigation) and have clearly defined purpose, not just move in a swarms.

With a multi-robot system the complexity is high. This is because most of these robots use complex sensors and require large amounts of processing power. It is therefore desirable to look at how simple and low cost a system can be and still have cooperative properties. This is a key element in fault-tolerant systems. A

fault-tolerant system is a system which can still operate after encountering a fault in one or more of its components. Knowing how few components is required to have cooperative properties is hence useful.

An educational project was the established at NTNU this fall to create a multi-robot system. The goal of this project is to see how low cost and simple the multi-robot system could be. The multi-robot system is focused on cooperation, and will perform tasks like formation holding and obstacle avoidance. Combining these, much larger tasks like search-and-rescue could be accomplished. This system is also going to be used as a demonstration, which limits both the size and set-up. The current goal is 6 robots. This means that while it is desirable to have scalability, some aspects of the system might compromise this in favour of simplicity. These aspects must be redesigned in future work, if scalability is still desirable.

This project is divided into different subsections:

- **Control System** Creating a cooperative behavioural control system that makes the robots able to do different tasks.
- **Hardware Platform** Creating a hardware platform able to implement different cooperative control systems
- **Detection Systems** Different systems and sensors able to detect other robots and obstacles, which are feeded as input to the control system.

Only a brief overview of the control system will be described in Section 1.1 while the full report may be read in [Klausen, 2012]. The plausibility of creating a hardware platform will be discussed in this report, and a list of components that might be used will be given. In this report there will also be given solutions on how to detect obstacles and other robots. Some of these solutions are described in literature while some are developed by the author of this report. It is worth noticing that the term project regards the project as a whole, and not only what is featured in this report.

1.1 Simple overview of the system

For simplicity the robots move on the ground. This to make them easy to control, and to minimize the damage if something goes wrong. This in contrast to making the robots fly, who might crash if something went wrong. For making the kinematics as simple as possible, the robots were made holonomic. This is done by the use of omniwheels (see Section 3.4). The robots have a hexagonal shape, and every odd side has an omniwheel on it. Using this shape and the properties of the omniwheels, the robot is able to move in the plane without any non-holonomic

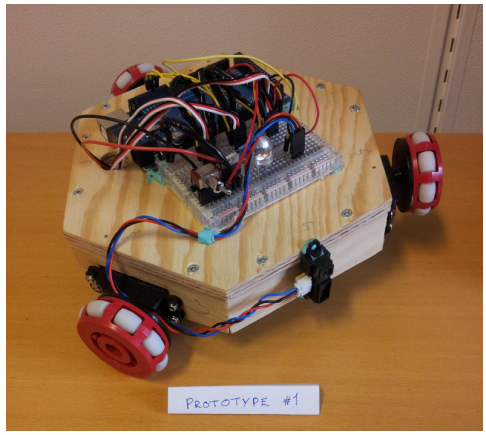

Figure 1.1: The prototype of the robot built for this project

constraints. Figure 1.1 shows the prototype of this robot. It can be seen in the figure that the omniwheels are placed on the odd sides, and in the front there is an IR-laser range sensor. Further description of this can be read in Chapter 3. It is worth noting that while the shape is hexagonal, all the illustrations in the report is feature the robot as a triangle with each wheel in the corners of the triangle.

This project focuses on simplicity and scalability. This limits the cooperative control system in many ways. Each robot uses the Null Space Behaviour(NSB) [Studi et al., 2006] method for "keeping their own integrity" (e.g. avoiding obstacles and robots, and avoiding other inhibitions compromising their actions). For the robots to cooperate a framework for solving the agreement problem using the methods in [Arcak, 2007] has been used. This choice of control system is very scalable and applicable to many Newtonian and Lagrangian systems. For a better description consult [Klausen, 2012].

1.2 Organization of this report

Since there are many topics discussed in this report, it is divided into smaller individual chapters to provide for better readability. These chapters are Problem Description (Chapter 2), Hardware Components (Chapter 3), Obstacle Detection (Chapter 4) and Relative Positioning (Chapter 5). Each chapter is somewhat self-contained, but for a more complete view of the system the reader is encouraged to read the chapters in the given sequence. They all have a small introduction presenting background materials like previous work, restrictions and problems.

There will also be discussions within each chapter as well as a discussion chapter at the end of this report.

This report first introduce the problem of multi-robot systems and this project. Some general methods and the methods for this projects will be presented and discussed before the current plausible solution is introduced.

The report continues with a look at hardware. Each section in this chapter will feature a hardware component, what requirements it has and why it should be used. The section will then follow up with possible products and the advantages and disadvantages will be discussed. Some of these products have been tested, which is also featured here. The questions of what hardware is required, and why will be answered.

The obstacle detection system will then be introduced in Chapter 4. Previous work and current solutions are discussed, followed by a presentation of a new obstacle detection system called Inverse Particle Filter.

Following this will be a chapter featuring the positioning systems. The main branches of solutions and previous work will be discussed, and the problems with these will be presented. A system using ultrasonic sensors to detect the robots relative position to each other is then presented, as well as what is required to implement this system.

The report ends with a discussion chapter discussing loose threads from the previous chapters, and a total plausibility of the system.

CHAPTER 2

PROBLEM DESCRIPTION

There are many solutions for creating multi-robot systems, and there are no limits on how these systems can be created. There is however constraints in this project, which limits the solutions possible. This chapter discusses previous works, constraints to this project, and the possible solutions to this project.

This chapter starts with a definition of what the main problems regarding this project is, as well as the constraints within it. Then comes a definition of the general methods of solving such problems. Lastly there will be a discussion regarding the current approach for solving these problems, and why they are chosen.

2.1 MAIN PROBLEMS

The main problems with this project can be split into three different categories:

- **Cooparate control system** The cooperate control system may be defined as the behaviour of the robots. As mentioned in 1.1 this will not featured in this report.
- **Obstacle detection** The robots are required to move around in an unexplored environment. They are therefore required to detect obstacles, which then can make the control system avoid the obstacles.
- **Localization** The robots are also required to move in formation and to communicate where they are to each other. This requires localization and a form of communication.

The two latter problems will be described in further detail in Chapter 4 and Chapter 5. There are however many ways to solve these problems, and later in this

chapter there is a description of what solutions were thought of and why the current plausible solutions were looked into.

2.2 Constraints

There exists many solutions on creating multi-robot cooperative systems. Some with platforms that would easily fit cooperative control system in [Klausen, 2012], which is used in this project. However the focus in this project is to see how simple a cooperative control system can be. Restricting the solution to only contain small microcontrollers, and the robots to be small and low in cost are only some of the constraints on the solution. These constraints are not only for simplicity, but as recalled from 1.1 these robots are meant as a demonstration tool. There are therefore some constraints in size and weight due to handling and transportation. Presented here are the main constraints:

- **Cost** The main concept of this project is to create many robots being able to do tasks as good or better than one robot. This means that the cost of many simple robots must be less or equal to one complex robot. Therefore the cost must be low on each robot.
- **Size and weight** The robots are meant to be small. This for making it easier to carry and handle. It would be less practical to have many robots that would be harder to transport and handle, than to use one more complex robot able to do the same task.
- **Power Consumption** By using small light weight robots, the energy source needs to be small as well. This means that large motors or power hungry sensors cannot be equipped on the robot.
- **Memory capacity** Each robot contains a small microcontroller. Since most microcontrollers have very limited memory, the system cannot use memory hungry methods.
- **Processing capacity** The microcontrollers used on the robots are not suited for heavy computations either. Most microcontrollers are 8-16 MHz , meaning that for it to be able to compute everything in time, the system cannot have many heavy computations.
- **External references** Since the robots are to be used in various environments with few possibilities to deploy and set up the robots, the robots should be able to work on the fly. This means that the robots should not depend much on external references, if at all. This will require smart solutions for localization and interaction with the environment.

- **Sensor measurements** The cost of the robots should be low, meaning that cheaper sensors should be considered. The robots must also be small in size, meaning that the sensors should be both small and cheap.

There will also be some race conditions and scheduling problems that needs attention. The robots require the position, velocity and orientation from each other to be able to cooperate. The velocity and orientation can be measured and sent wirelessly. This requires a fast and precise communication for the information not to be outdated, which would give race conditions. For acquiring position, communication is not enough. As mentioned above the dependence on external references should be low, meaning relative positioning rises as an alternative. Each robot must therefore be able to detect all the robots close to it. This has to be communicated and calculated to create an estimation of their position. This requires good scheduling.

2.3 General Methods

For a robot to work within the real world there are three problems that arises [Mullane et al., 2011]: Mapping, localization and path planning. Since our robots would not have any external references of the world, these problems would be needed to solved simultaneously. This is called an *integrated approach*.

Mapping is the problem best defined as "What does the world look like?" and is to integrate all the data from the sensors and merging them to an image of the world. This is opposite to the localization problem where the robot ask "Where am I?", where the position of the robot relative to the map is needed. Path planning is to understand where the robot is going, which requires a map, location and a desired goal [Mullane et al., 2011].

Since such a system is integrated all the aspects are dependent on each other. Therefore to create such a systems all problems must be handled simultaneously. Taking this to the multi-robot level, the number of problems multiplies. The goal with such systems is to create a completely generalized system. A robot able to locate itself within the world, while understanding what the world is and able to complete its tasks would be a general purpose robot. Having a multiple of these robots able to cooperate, would be a solution able to handle numerous tasks in different environments.

There are different approaches of implementing behaviour control systems. If the system to control is imagined as a black box with an input and an output, there are roughly two main methods of controlling it:

- **Probabilistic methods** Probabilistic methods try to look for patterns be-

tween the input and output without trying to understand the black box. This is done by applying an input and estimating output as a probability. If the output complies with the estimation, the method gets more confident. This is a good approach if the black box is hard to model (e.g. too big, too complex, too many variables). A disadvantage with this approach is that the black box may never comply, and therefore render the method useless.

- **Model-based methods** Model-based methods tries to make a mathematical model of the black box and to find the right way of controlling the box. The advantage of this is that the black box is defined, and therefore easy to get the wanted output. To get the complete model of the black box may be extremely hard, and complete control might also be impossible. Another disadvantage is that the model might differ from the actual black box, and may therefore give unwanted errors.

When creating a behavioural system for robots it is most beneficial to combine some aspects of these branches. Since both the location and environment cannot be modelled in advance, probabilistic methods can use probability and Gaussian-distributions to calculate the likelihood of both the location of the robots and the environment. Probabilistic methods can better process input which does not give direct information, but rather has to be interpreted, e.g. images and maps. Images for instance, can be processed to give probabilities of objects being in front of the robots. Model-based systems on the other hand, has the advantage of that it can be simulated and tested before it is implemented, and a deviation will soon be detected. Control references such as motion and formation can be defined in advanced; a model of this will give the wanted behaviour. Combining these two methods will prove beneficial. Making the behavioural decisions and cooperation with model-based methods will make the behaviour of the robots consistent; making it able to simulate scenarios in advanced. This combined with the detection and exploration with probabilistic methods, which can sense the outside world. Sensors have measurement faults and may misbehave or input error to the system. These errors may be counteracted with a probabilistic systems.

This combination is a target for the multi-robot system in this project, and the solutions presented here has this in mind.

2.4 Proposed Solutions

The proposed solutions of the two main problems mentioned in 2.1, are covered in this report. Presented here is an introduction to the proposed solutions for solving these problems.

2.4.1 Obstacle detection

The first thought on obstacle detection was to use SLAM (Section 4.2.1), but the field of SLAM is so big that the focus on this project would switch solely to SLAM rather than the hardware platform. Another problem with using SLAM is that it requires much processing power and memory, which according to Section 2.2 is not desirable.

The current solution is to use one or more IR-lasers and an Inverted Particle Filter. The Inverted Particle Filter creates a local map which spans in a circle around the robot. As the robot moves forward it "forgets" the obstacles detected behind him. This gives a finite number of detected obstacles, and has a low cost in processing power and memory.

2.4.2 Localization

It was early decided that the positioning system would be relative. This means that there is no absolute position of the robots, rather a relative position between each robot. This is due to simplify for the set-up of the robots and to make the robots able to move more freely.

When the thought of using SLAM was on the table it would be possible to find the relative position of each robot by comparing the maps each of them had created. This again would be complicated and when SLAM was declined so was this idea. When the Inverted Particle Filter started to linger as a solution, the thought of using ultrasound to detect the relative position rose. The initial thought was that a robot sent a ultrasonic wave, which the other robots detected. The range could then be calculated and a merging of the Inverted Particle Filter maps could locate each robot. After consulting an acoustics engineer, it was understood that to detect the angle of a sound wave was not hard, but to detect the distance required synchronized clocks.

The current solution is therefore to use ultrasound and synchronized clocks to detect the angle and distance of each robot, and a protocol (Section 5.4) to distinguish each robot from each other.

CHAPTER 3

HARDWARE COMPONENTS

Hardware is an important part of any physical system. If the hardware does not meet the requirement of the system, the system will not work. It is therefore vital for any system to know what is required and why. It is also important to understand what the hardware can and cannot do.

In this chapter the hardware on the robots used in this project will be discussed. First there will be an introduction discussing the hardware in the robots in general. For each section thereafter it will be discussed microcontrollers, IR-lasers, omniwheels, communication and ultrasonic sensors. Each section will describe what each hardware component is, and what is required by the system. It then proceeds with suggesting different products for the system. Some of these products have been tested, and some experiments have been conducted. This is presented in each section.

3.1 INTRODUCTION

Using many cheap robots to replace one expensive robot has many advantages. By using many robots, both risk and error is spread out into each robot. Therefore if one robot fails, it does not affect the work of the other robots to much. If the robot is also cheap, it is also easily replaceable. Therefore expense was one of the main focuses in this project. One of the other constraints on the robots was that they were primarily going to be used in demonstrations, where environment can be very diverse, and where external equipment cannot always be set up. This meant that the robots should not have any external references, and since they also primarily are going to be indoors, they cannot use GPS either.

With these restrictions and the ones mentioned in Section 2.2, the choice of hardware is crucial. The size of the robot restricts the size of components, and the budget restricts the quality of the components. These combined restricts the performance of the components. The use of simplicity and integration is key for making the whole system work.

In this project it has been a big focus on commercially available modules. This is due to the time constraint in finishing the prototype. Creating a complete board with all functionality will be done at a later point.

3.2 MICROCONTROLLER

The most vital part of the hardware is the microcontroller. It needs to have good performance and low power consumption. There has been many discussions on which microcontroller to settle for, and which properties that are important to look for. Some of the properties needed are:

- **Speed** For the cooperative control system to work efficiently and for all the sensor data to be processed in time, the processor has to be fast enough. Since the sensor complexity and system requirements are not known to the full extent yet, this property is hard to fulfil.
- **Resources** The cooperative control system might require many resources, like memory. The system is designed to some extent so that it does not collect and store new data without replacing old data, the resource requirement of the system is therefore thought to be constant.
- **Programmability** The microcontroller should be easy to program. The cooperative system is most likely going to be implemented in SimuLink, using SimuLink Coder to generate the code. The microcontroller should be easy to connect to the computer, download this code and run it.
- **Simplicity** The microcontrollers should come on a complete board kit. This because it is not enough time to create a circuit board from scratch. Board kits these days comes in many forms and qualities and are usually very cheap.
- **Prototyping** It should be easy to prototype with the microcontroller. If the microcontroller is on a board, with the I/O pins laid out, it will be easy to prototype, by adding and removing sensors.

The two microcontrollers that has been tested are the ATMEGA2560 on the Arduion Mega2560 board, and the 32UC3A3256 on the UC3-A3 Xplained board.

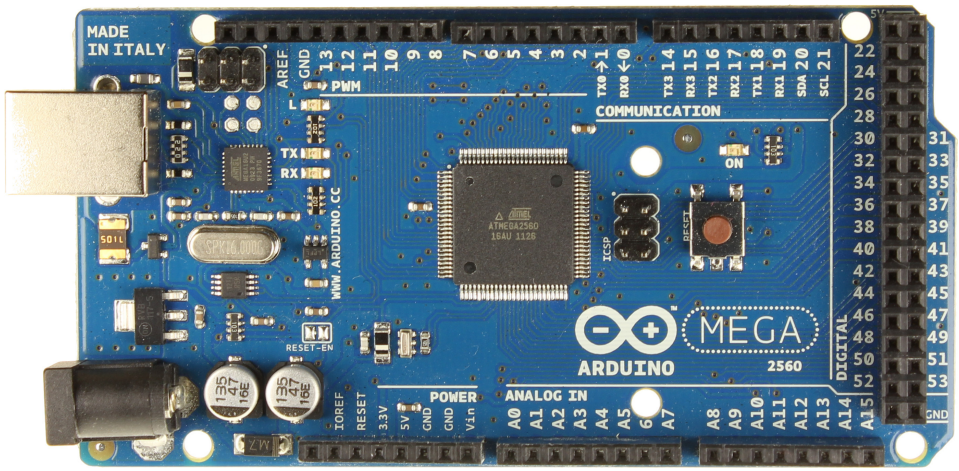

Figure 3.1: The Arduino Mega 2560 viewed from the front. Image provided by arduino.cc

They are both on sold on complete board kits. Both boards are cheap and easy to obtain.

3.2.1 Arduino Mega

The Arduino Mega 2560 is a board based on the ATMEGA2560. It is a product from the Arduino family where the focus is on prototyping. It has 57 I/O with 14 of them being PWM outputs, 16 analogue inputs and 4 UART ports. This is more than sufficient when it comes to connecting to the rest of the hardware. With 16 MHz the speed might be a problem, but the full system has not yet been tested on the chip. Simulink has an Arduino expansion pack making it able to create code, compile it and upload it to the Arduino with only one click. This is very favourable in this project. It is also easy to create pure C++ programs on the Arduino and to create a library to be able to implement the C++ functions wanted. This is very beneficial, because then you could create simulators in either C++ or Simulink and test implementations. The implementations can then be almost directly be put onto the microcontroller. For one of these simulators see Section 4.4. Arduino is designed for hobby projects and prototyping, and not well suited for a complex systems or systems with real-time requirements. This can be a problem when it comes to scheduling and positioning. The Arduino Mega has an inbuilt bootloader, meaning it does not need any external programmer for it to be programmed, but also means that it doesn't have a debug function.

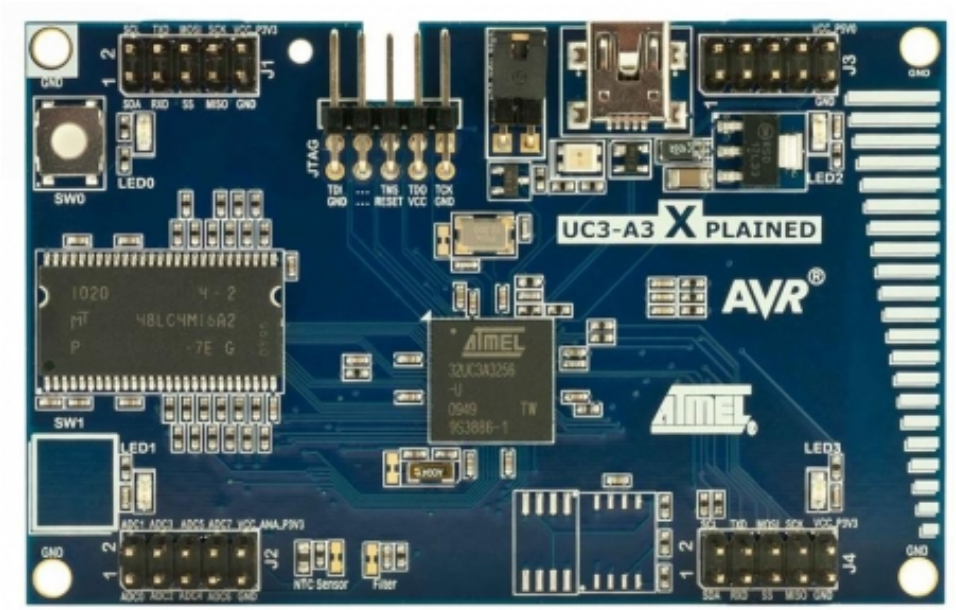

Figure 3.2: The UC3-A3 Xplained viewed from the front. Image provided by atmel.com

3.2.2 UC3-A3 XPLAINED

The UC3-A3 Xplained is a board based on the 32UC3A3256. It is part of the Xplained series, which is a series created by Atmel used for learning and prototyping. The 32-bit microcontroller run at 12 MHz , but can run at up to 66 MHz . The biggest problem with the board is that it does not have any PWM drivers, and that it does not have more than 4 PWM outputs. This may cause a problem when we are attaching other hardware. It only has 32 I/O ports, while the microcontroller has 144. This means that the pins are multiplexed, which might result in functions being unable to be used together. It has an external programmer, and therefore can be debugged. Since it is a 32-bit microcontroller, it is both faster and bigger with more memory than the ATMEGA. The microcontroller is programmed in C, and the most necessary external libraries available are created by Atmel. Since there are less libraries available it would also require more time to implement all the necessary functions. It is further worth noting that there is no direct extension packs for Simulink to the UC3.

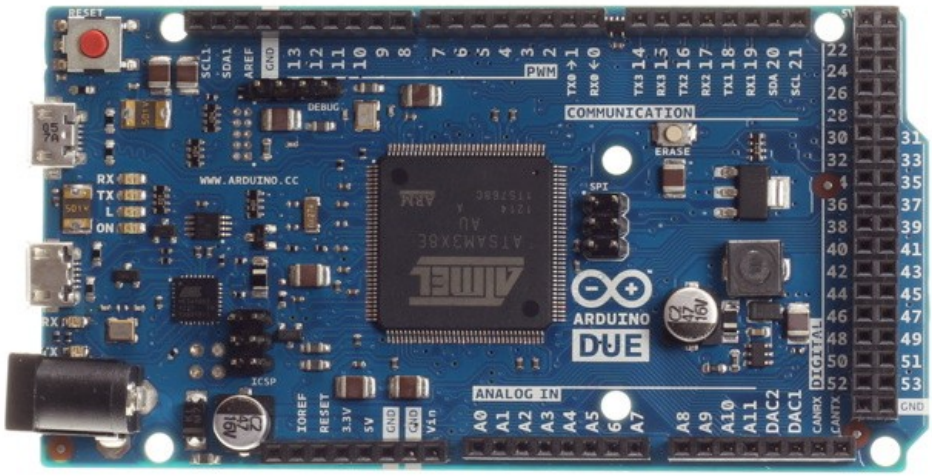

Figure 3.3: The Arduino Due was recently released. Image provided by arduino.cc

3.2.3 Other alternatives

The Arduino Due was released on 22nd of October 2012, and is the first Arduino board with a 32-bit processor. Its layout is almost the same as the Arduino Mega (with 54 I/O pins, where 12 are PWM, and 12 analogue inputs and 2 analogue outputs). It has a staggering 512 KB flash memory and 96 KB SRAM. The clock speed is 84 MHz . It even has a DMA on board. It comes with all of the advantages that the Arduino Mega has, and it is much faster, has more memory and is less power consuming. This is the most promising microcontroller, and a further look into this will be done.

Other alternatives might be the Raspberry PI or the BeagleBone. These are very fast and powerful, and can have Linux distros mounted onto them. They have good enough power to do image processing, which could make it easy to detect objects. The system the robots are going to use, does not require much to run, but require many analogue and PWM signals. It is easier to control the signals using a smaller microcontroller. For later use it might be beneficial to use these microcontrollers.

The one worth looking into for the implementation is the Arduino Due, both since it has many advantages regarding this project, but also because the Arduino plat-

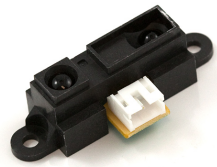

Figure 3.4: The Sharp GP2Y0A21YK IR Range Sensor is the range sensor used on the robot to detect obstacles. Image provided by sparkfun.com

form is familiar to the participants of this project.

3.3 IR-LASER RANGE SENSOR

For detecting obstacles, one or more IR-lasers are to be used. Reasons for this are described in Chapter 4. There are some requirements to the IR laser.

- **Accuracy** The laser must be accurate enough to be able to work predictably.
- **Range** Range is not a huge requirement. It is required that it ranges far enough to respond to the information given from the laser. A range of 50 cm or more is sufficient.
- **Strength** The strength of the laser beam needs be low. The robots are required to not harm humans, so a beam that can potentially burn eyes or skin cannot be used. Additionally, the beam is infrared, meaning that if it is pointed towards the eye the beam is not seen. A harmful beam aimed at an eye would therefore not be detected before it is to late.

3.3.1 SHARP GP2Y0A21YK IR RANGE SENSOR

An IR-laser range sensor that has been looked into is the Sharp GP2Y0A21YK. It is cheap, and comes with a processor doing signal processing and filtering, so that the only interface is a analogue signal. It is fairly accurate, the only limitation is that the analogue signal is non-linear. This can easily be solved by passing the signal through a lookup table with linear interlace. The laser's strength is also at a non-harmful value. It can detect ranges up to 80 cm . Ranges farther then this

will only give random high values. There exists other versions of this laser with extended range.

One laser was bought because it was cheap. It was needed to test if the laser was suitable, so an experiment on how easy it was to create a contour map with the laser was conducted. The laser was mounted on top of a "9g small servo"[1] from SparkFun.com. The servo and laser was controlled by a Arduino Mega2560 that was programmed in Simulink. A range of miscellaneous objects were places in front of the laser, and the servo sweeped from -45°to 45°. The analogue signal was collected, and processed along with the angle accumulated to the servo. These two parameters created a circular coordinate point which then was placed in a graph. The sweeping motion was done 20 times before the program exited and created the plot.

As seen in Figure 3.6 the contour is not a good representation of the obstacles in front of the laser. If looking closely on Figure 3.5, it can be seen that at the left end there is an open area, which is also present in the contour map. This is represented by the huge spikes on the lower part of the plot. What can also be seen in the contour is that the error is systematic and can be categorized into two faults: A hysteresis and noise in form of spikes.

The hysteresis probably comes from the implementation in Simulink and the servo. When Simulink takes a range measurement it also use the angle that it has just given to the servo to calculate the coordinates. The servo has not been able to go to the position it is supposed to be at, when the measurement is taken. When the servo sweeps to the left, the servo's actual angle is a little to the right then anticipated, while when the servo sweeps to the right, the servo's actual angle a little to the left. The combined difference in angle comes in form of a hysteresis. When the servo's speed was lowered, the hysteresis decreased. To solve this, a delay can be put between the time the servo gets a new input angle and the range measurement is taken. This will give the servo time to get to the desired angle before the measurement is taken.

The spike noise that is shown is probably the servo's fault. As the servo moves it jitters a lot because it constantly receives a new value. This can again be solved by adding a delay between each servo input.

The errors that occur (hysteresis and spikes) are consistent, and the noise outside of these is small. The laser itself is therefore not the source of error, and create little noise. It is consistent and has a quick response time. The laser might therefore still be suitable for this project. Due to time constraints this experiment was not conducted again, with a delay added between the servo input and the measurement capture.

[1] It weighs 9g, hence the name

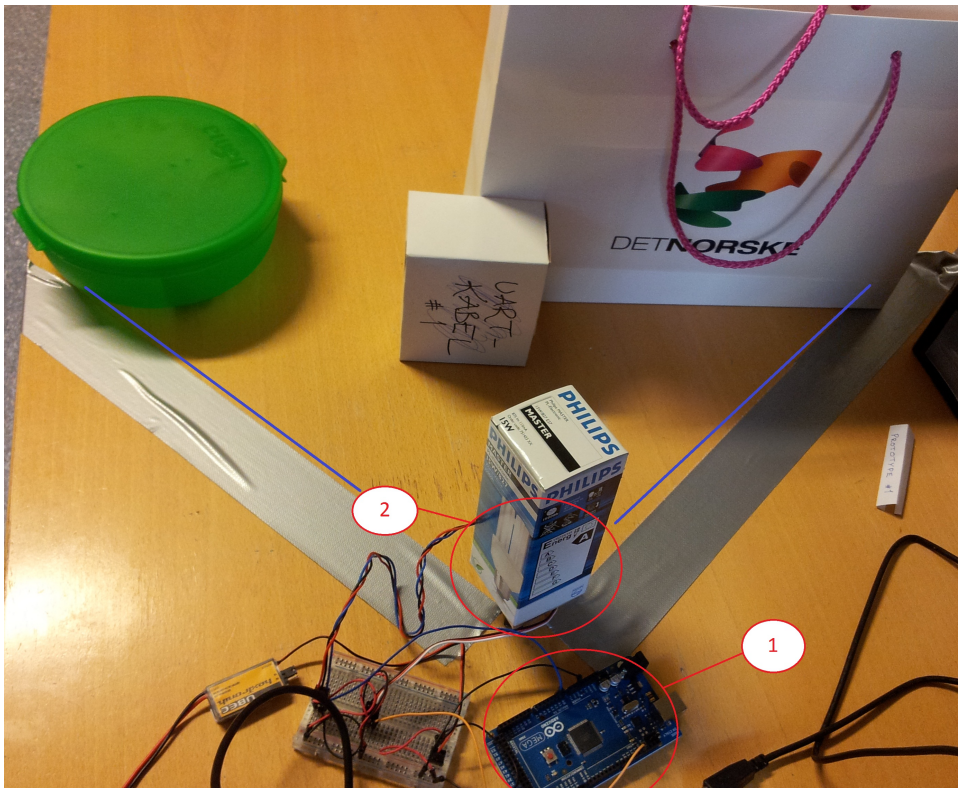

Figure 3.5: The set-up of the experiment. The blue lines is the field of view of the laser. At point 1 the Arudino Mega can be seen. At point 2 the laser mounted onto the servo. It is hard to see, but the motor is under the cardboard box, and the laser is behind it facing the various objects.

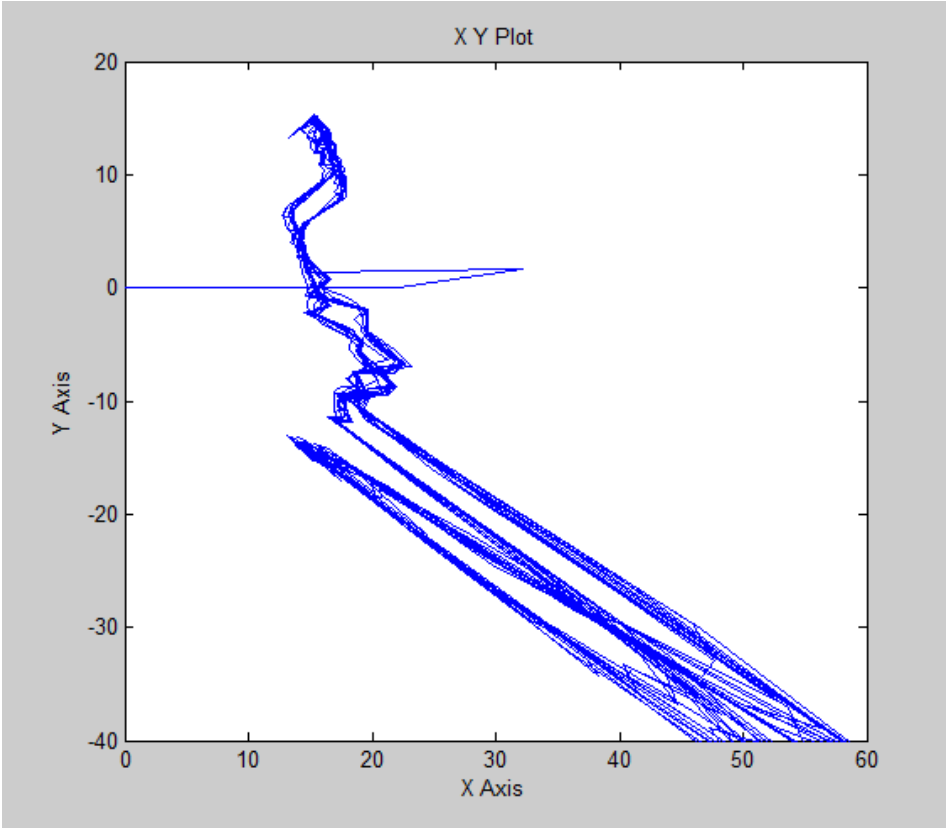

Figure 3.6: The contour map created by the IR-laser. The contours represents if the set-up on Figure 3.5 is viewed from the right, mirrored around the horizontal axis. The laser is situated in (0, 0) and facing in the positive X-axis

Figure 3.7: The omniwheels used by the robot, provided by kornylak.com

3.4 OMNIWHEEL

Omniwheels are wheels that can roll as normal wheels in one directions, and able to slide sideways. These wheels were chosen because placing these on the robot in a triangular shape will allow the robot to have true planar movement, without non-holonomic constraints.

In Figure 3.8 it can be seen how the omniwheels work. It can be seen that the omniwheel rolls in the purple direction, while it slides in the green direction. For a more detailed description read [Klausen, 2012].

3.5 IMU

For making Kalman filters and giving the robot a heading, the need of inertial measurements rise. This requires a gyroscope, a magnetometer, and an accelerometer. The gyroscope and accelerometer measures the velocity and acceleration of the robot, which can be integrated into finding the current position of the robot. The magnetometer finds the orientation of the robot.

The magnetometer has a flaw because the magnetic field shifts if the robot is in a metal container. This is fortunately a flaw that can be ignored in this project. If the robot is placed within a metal container, it will have a consistent north (which might not be the true north). The other robots (also in the container), will have the same consistent north. The robots only needs to know the relative position of

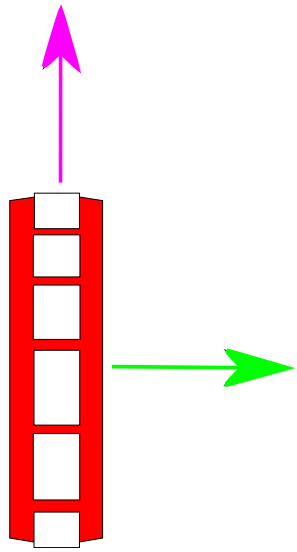

Figure 3.8: The omniwheel viewed from the top. The wheel can roll in the purple direction and slide in the green direction

each other, and therefore only needs to know the difference in angle, and not the true angle.

3.5.1 MPU6050

The MPU6050 is a 6-axis motion tracking system. It provides gyroscope and accelerometer, and has I^2C interface. It also has an external I^2C connection making it easy to connect to an external magnetometer.

A circuit for MPU6050 has been created, and the schematics and board layout is provided in Appendix A.1.

3.5.2 HMC5883L

To complement the MPU6050 a HMC5883L triple-axis magnetometer can be used. It has an I^2C interface, and can therefore be directly connected to the MPU6050. It has been tested on the robots, and proves to be a good choice for measuring the orientation of the robot.

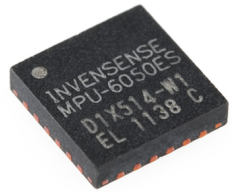

Figure 3.9: MPU6050 is a microcontroller with gyroscope and accelerometer. Image provided by sparkfun.com

Figure 3.10: HMC5883L is a triple-axis magnetometer. Image provided by sparkfun.com

3.6 COMMUNICATION

For the robots to act together, they need to be able to communicate with each other. There are mainly three ways of communicating wirelessly: WiFi, Bluetooth and ZigBee. With the constraints in Section 2.2 in mind, following are the types of wireless communication discussed.

- **WiFi** WiFi is a wireless communication which use radio waves and the IEEE 802.11 protocol. The WiFi standard is widespread today, and the WiFi chipsets that are manufactured are getting smaller and cheaper. There are many advantages with a wireless network, where one is that they are able to communicate through the internet. This gives a human an opportunity to communicate with the robots even on the other side of the planet. It also gives the robots an opportunity to download new updates or missions from a server. Some disadvantages is that WiFi is not a mesh network. This means that all the robots have to have communication with a host robot in order to work. If the host malfunctions, the network is lost.
- **Bluetooth** Bluetooth is a wireless communication protocol for use in PAN networks. Bluetooth was first used for communication between telephones, but has spread to many devices and is now a very common protocol for wireless communication. Bluetooth Low Energy(BLE) is a feature that came with the Bluetooth 4.0 protocol. Although it has less data throughput and range than the Bluetooth 4.0 protocol, it has a significant reduced energy consumption. Bluetooth Low Energy is a good candidate for this project, but the protocol does not have support for mesh networks yet. It has a master-slave structure. There exists algorithms for creating bluetooth mesh networks, but there is no widespread variant. Another problem with BLE is that there are not many modules available, meaning that this has to be made, which cost time.
- **ZigBee** ZigBee is a wireless communication protocol based on the IEEE 802 protocol. It is specifically created for mesh networks with low energy consumptions. These are advantages that suit the constraints very well. The Zigbee protocol was designed because many foresaw that Bluetooth and WiFi would be unsuitable for wireless sensors and self-organizing networks. Zigbee has many low-cost modules on the market, which makes ZigBee a good candidate for this project.

3.6.1 XBEE

The XBee is a ZigBee module made by Digi International. It comes with a broad range of antennas, with many different characteristics. The XBee 2mW Chip An-

Figure 3.11: The XBee Series 2 is a module using ZigBee as a protocol for the wireless communication. Image provided by sparkfun.com

tenna 2 series has been tested in this project. The chip has the full ZigBee protocol built in, with a 3v3 UART interface. It is cheap and easy to get a hold of. A XBee Shield suited for Arduino can be bought which creates a compete ZigBee communication between Arduino boards without any specific requirements. There are also modules for connecting the XBee to an USB interface so that it can be plugged into computers, giving communication between the robot mesh-network and a computer. As mentioned above, the Zigbee protocol has a mesh-network feature. This feature has not been tested on the X-Bee yet, but research show that it is not hard to implement [Faludi, 2011]. The X-Bee though, has a master-slave system, which is not desirable, but is sufficient enough for the purpose of this project. A further improvement would be to have a ZigBee module without this feature.

3.7 Ultrasonic sensor

In Chapter 5 it will be an explanation of why ultrasonic sensors should be used. There are not many cheap ultrasonic instruments on the market. This means that circuits for transmitting and receiving ultrasonic signals has to be made. A basic signal generator for transmitting, and a signal amplifier for receiving should be sufficient. The frequency to be used can be discussed, but there is no right answer here.

3.7.1 400SR100 and 400ST100

The 400SR100 and 400ST100 are respectively a ultrasonic receiver and transmitter provided by Pro-Wave. They have a frequency of 40 MHz . These were mainly

Figure 3.12: The 400SR100 and 400ST100 are respectively a ultrasonic receiver and transmitter in the 40 MHz range. Image provided by prowave.com

chosen because they were cheap. Initial tests using a signal generator on the transmitter and an oscilloscope on the receiver has proven that a signal can be detected within a fair range. The receiver has noise at 320 mV , which is persistent. All voltages higher than this may therefore be detected as a signal. To be able to transmit a signal over 200 cm it only is required an amplitude 10 V. Creating a signal generator at 10 V_{rms}can therefore be made for transmitting signals.

CHAPTER 4

OBSTACLE DETECTION

Obstacle detection is a problem when robots are moving through environments. Obstacles can be dynamic or static, and detection of these obstacles usually requires lots of resources and/or expensive sensors.

In this chapter there will be a brief description of previous works on solutions, and why these are not suitable. This will be followed by a description of the Inverted Particle Filter, which can be described as a brute force method for detecting obstacles. In the end of this chapter there will be an implementation of the filter

4.1 INTRODUCTION

It is not hard for a robot to detect obstacles. This can easily be done with a laser or a bumper with a sensor on it. It is to use this information to manoeuvre around that is the challenge.

Using localization and obstacle detection combined is a hot topic to research. The leading method is SLAM, Simultaneous Localization And Mapping. Other methods available are forms of particle filter. All these methods are complex and use a lot of memory resources and processing power. It is therefore required to either tweak existing methods or make a new one.

In this project there are not much room for complexity. This is also the point of the project. The robots only need to create a local map around themselves so they can interact with their surroundings, although a global map would be beneficial. Methods like guidance and waypoint tracking are required to make the robots do a task, but these require a map or some sort of reference. SLAM solves this by building a map while exploring the environment, but then again SLAM is

very resource consuming and the robots does not need a full map. It looks like this project is in a grey zone of this research field, where alternative solutions are needed.

4.2 PREVIOUS WORK

4.2.1 SLAM

SLAM, short for Simultaneous Localization And Mapping, is a fairly new field for exploration with robots. The point of SLAM is that while the robot moves, and tracks its movement, it records features it senses. These features then correct the error in the movement the robot tracks, while simultaneously the tracked movement corrects the error in the features' position. The tracking mainly use an extended Kalman filter or a particle filter on top of some sort of IMU or GPS. There are many different methods for sensing features, and these mostly branch into different SLAM methods. SLAM is very suitable for exploring and mapping an area, where creating maps is the main task of the robot. Here are some of the SLAM methods, both well-known and recently developed. Note in these methods, that pose is often used. A pose is a position and orientation of an agent at a given point in time.

GRIDSLAM

In GridSLAM the map is set up in an opacity grid, where each cell has a value of opacity telling the likelihood of an obstacle within the cell[Chae and Yu, 2010]. The main problem with GridSLAM is discretization. Depending on the grid size the map either consumes large amounts of resources or has poor resolution. The grid can be static in position and dynamic in size, where the robot moves around the grid to observe more data. This will eventually lead to a large grid and therefore large resource consumption. To prevent this the grid can therefore be dynamic in position and static in size. The grid will therefore shift whenever the robot moves. The problem here is that a shift of a large array like this will require much processing power.

GRAPHSLAM

GraphSLAM[Thrun, 2006] uses a matrix Ω and a vector ξ to create a map. GraphSLAM requires a feature detection. This means that the robot is required to detect the range from it to a feature more than one time. A robot using GraphSLAM

therefore usually use stereo vision to detect features. The Ω matrix is an $n \times n$ matrix, where n is the number of poses the robot has measured plus the number of features it has detected. The ξ vector is a n vector. Inside Ω is the relationship between the poses and features, while in the ξ vector the sum of ranges for one pose or feature is placed. A finished Ω and ξ with three poses and one features might look like this:

$$\Omega = \begin{bmatrix} 1 & -1 & 0 & 0 \\ -1 & 1 & -1 & -1 \\ 0 & -1 & 1 & 0 \\ 0 & -1 & 0 & 1 \end{bmatrix}, \xi = \begin{bmatrix} -5 \\ 0 \\ -4 \\ 9 \end{bmatrix} \tag{4.1}$$

Whenever the robot needs to calculate the most likely ranges to other poses (i.e. the motion of the robot) and the ranges of the features (i.e. the position of the features), then the simple equation for creating the best estimates of the variables are used:

$$\mu = \Omega^{-1}\xi \tag{4.2}$$

The problem with this method is that the size of the Ω matrix is depending on the number of poses and features the robot registers. This means that the Ω matrix potentially could have hundreds of columns and rows. To add to this, the matrix has to be inverted to get any usable results. Even when using Householder Reflection, a matrix of this size would require a lot of processing power to invert.

MonoSLAM

A fairly new method is called MonoSLAM [Davison et al., 2007], short for Monocular SLAM. This is a SLAM method that only use one camera and an IMU. It takes a picture and extracts the features from it (using either SIFT, SURF, FREAK or other algorithms). Each picture is then compared, and the movement from one picture to the next is calculated. This together with IMU data gives the motion of the robot, while also giving a map of the area. MonoSLAM has only been implemented on hexacopters with the camera facing downwards. It is therefore good for mapping and exploration, but not for moving around on the ground. Another problem is that the feature detection algorithms available require huge amounts of resources and processing power.

The main problem with SLAM is that it requires a lot of resources and processing power. SLAM is also more focused on mapping rather than localization. The main focus for the robots in the projects is to localize themselves and each other, and come around common obstacles located around them. There is no need for a

complete map or complex solutions. This again means that a solution like SLAM might not be necessary.

4.2.2 PARTICLE FILTERS

Particle filters, also known as sequential Monte Carlo method, is a model estimation technique. The method is mostly used in robotics to get the location of a moving robot. Particles are scattered around an area with the same properties as the robot using the filter. As the robot moves, the particles moves in the same manner. For each measurement the robot takes, each particle does their own simulated measurement. The weight of the particles are then updated based on their measurement compared to the robots measurement. If the particle's measurement is consistent with the robot's measurement the weight is large, while if it is not consistent the weight will be very small. After this a resampling phase occurs. Here the particles will be selected randomly based on the weight, giving a new set of particles. The result will be particles that are more accurate to the location of the robot. After many cycles of this method, only a few particles that are near the robot's location will be present.

The heart of the particle filters is the resampling phase. The implementation of this phase can make or break the filter. The phase is meant to sort out good from bad particles, which means that if the phase sorts wrongly the filter converge to the wrong particles. Each particle is has a weight property that determines whether it follows the current measurements of the robot. However this does not mean that the particles with the highest weights are the best. The measurements might be noisy or completely wrong, and there might be many errors in the simulated model of the filter.

The most used resampling method is called the resampling wheel. It takes a random selection of particles based on weight. Each particle represents a slice equal to its weight on a wheel. Particles with a small weight gets a small slice, while particles with large weight gets a larger slice. The algorithm 4.1 starts by picking the index of a random particle. It then finds the current maximum weight of all the particles. The algorithm then starts to pick N random particles. This is done by choosing a random number β. If the value of β is higher than the current particle weight pointed to by the index, the weight is subtracted from the β, and the index increased by one. This continues until the β is within a slice. The particle owning the slice is then picked.

```
1 // pick random number
2 index = random[1..N]
3
4 // Get the current maximum weight
5 max_weight = max(weights)
```

```
6
7  // for each particle
8  for(i = 1..N)
9  {
10   // choose a beta that is between
11   // 0 and the maximum weight * 2
12   beta = random[0..2 * max_weight]
13
14   // while the beta is outside
15   // the current slice pointed to
16   while weight[index] < beta
17   {
18     // remove the slice from beta
19     beta -= weight[index]
20
21     // move to the next slice
22     index = (index + 1) % N
23     }
24
25   // pick the particle pointed to by the index
26   pick particle[index]
27 }
```

Listing 4.1: **Resampling wheel algorithm** for re-sampling a particle filter

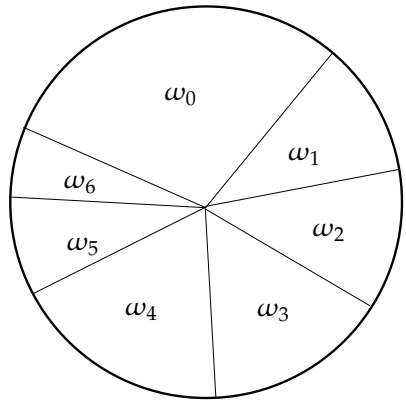

Figure 4.1: A graphical representation of the resampling wheel. Here each particle has a slice based on the weight ω. The ω_i is the weight of the particle i

The resampling wheel covered earlier is only one way to implement the resampling phase. There are many others which are more suitable for special cases. The most common problem in designing a resampling method is called particle

depletion. The problem occurs because the resampling phase picks multiples of some particles, and removes others. As the multiples grow in number, they may be overrepresented, and the particles with low weight will be killed of. This will lead to an overconfident result and the accuracy of the filter will drop. There are no good solutions on how to fix this problem, but the most common solution is to "mutate" each particle as they are picked, so that none of the particles are the same.

There are many types of particle filters, because a particle can be defined for different purposes. The only requirement a particle has is to have a weight and a resampling method comparing the weights of each particle and selecting particles. Examples of these are Rao-Blackwellized particle filter, particle particle filter and Gaussian particle filter.

4.3 INVERTED PARTICLE FILTER

The problem with the solutions presented above is that they require a lot of processing power. It is therefore presented here an alternative solution, which is more a brute force method for detecting obstacles. The method is called the Inverted Particle Filter, and the idea is not to find the location of the robot using the environment (like a particle filter), but to find the obstacles in an environment using the location of the robot. Each particle in the filter represents a point on the real world relative to the robot, and which may or may not be part of an obstacle.

The particles in the filter forms a circular dynamic map, with a fixed radius and the robot always in the centre (Figure 4.2). Each particle is spread out uniformly over the map, and as the robot moves each particle moves in the opposite direction. This represents that the obstacles "move" in the opposite direction of the robot, see Figure 4.3. If a particle moves outside the map the particle is removed and a new one is created on the other side of the map. Each time a laser measurement is taken, a beam is created from the centre of the map in the direction and with the length of the measurement. The particles close to the beam is less likely to be part of an obstacle, which will lower the weight of the particle, while particles close to the end of the beam are most likely to be part of an obstacle, which increases the weight.

Each particle has an opacity value, also termed weight, ranging from 0 to 1 which represents the probability of it being part of an obstacle. A value of 0 represents that there is probably not an obstacle where the particle is, while a value of 1 represents that the particle is definitely part of an obstacle. Each movement the robot has makes the value of each particle converge to 0.5. This represents that each time the robot moves, the information of the particle depletes due to the

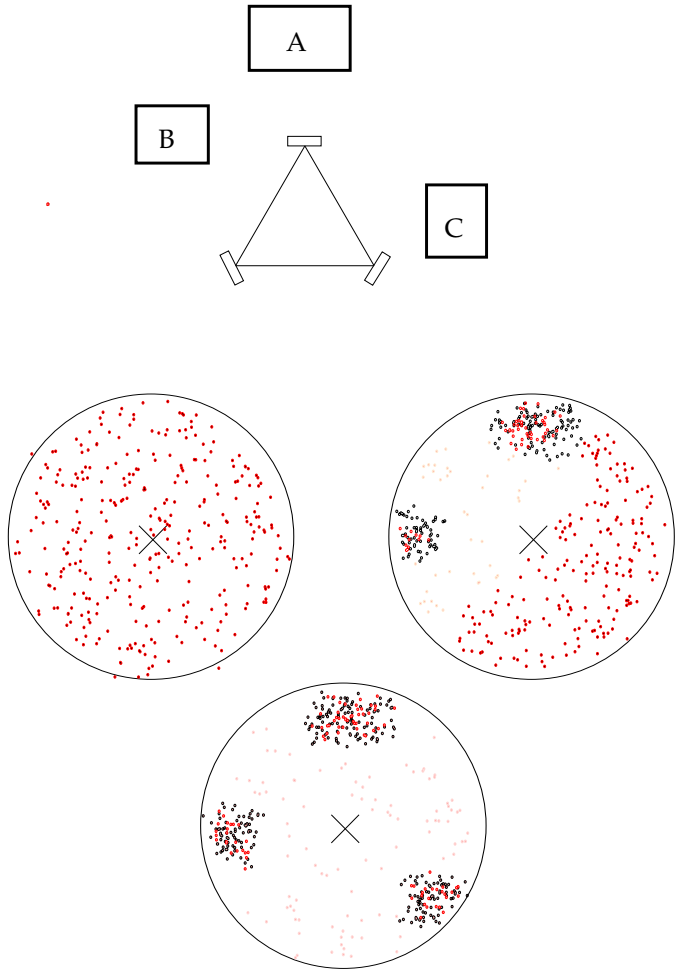

Figure 4.2: The principle of the Inverted Particle Filter. On the top of the image: The robot is placed beside obstacles A, B and C. The image on the bottom represents the map of the Inverted Particle Filter at three different times. The first map is the initial map of the particle filter where all the particles are scattered, and they all are red (weight of 0.5). The second map is when the robot has rotated and gained information about the surroundings. Both obstacles A and B are detected and the particles are black (weight of 1). The third map is when the robot has all the information about the environment, and all the obstacles are detected. The pink particles are particles with weights close to zero.

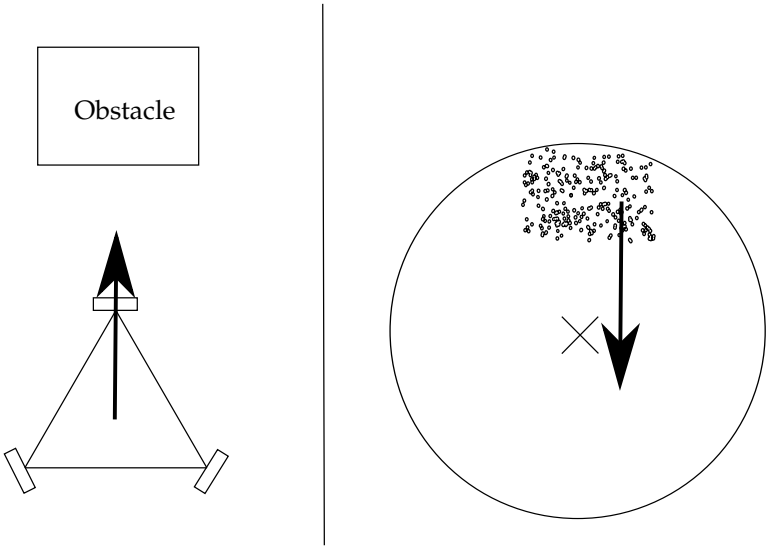

Figure 4.3: To the left: A robot moving towards an obstacle. To the right: the map created by the Inverted Particle Filter. It can be seen that the particles represents the obstacle in front of the robot, and as the robot move towards the obstacle, the particles move towards the X (representing the robot)

inaccuracy in the movement. When a particle has the value of 0.5 it is not possible to determine if the particle is part of an obstacle or not.

An example of the Inverted Particle Filter can be seen in Figure 4.2. The robot here is situated between three obstacles A, B and C. Each circle represents the map formed by the individual particles in the filter, at different times. Each particle has a weight which ranges from 0 to 1, represented in the picture as ranging from white to black, with red as 0.5. The first circle represents the initial particle filter. As described above, the particles are uniformly scattered in a circular map, and with an initial weight of 0.5. The map does not give any information about the surrounding obstacles. After some time the robot has rotated to the left, and acquired information about the surroundings. Recall that the robot uses a laser measurement to determine the weights of the particles. The next circle represents the state of the particle filter after the robot has turned for a while. It can be seen here that the particle filter has detected obstacle A and B. This can be seen by the particles being black (representing a weight of 1, and therefore a high probability of it being part of an obstacle.). The third map is taken after the robot has turned a full turn, the whole environment has been scanned, and the Inverted Particle Filter shows three probable obstacles around the robot.

The Inverse Particle Filter is loosely based on the GridSLAM method, but instead of having a grid with a fixed size which would cause a discretization problem, the particles are more continuous. This means that a robot is not limited to traverse through a grid as in GridSLAM, but can move freely around with as long as the particles complies with the movement.

Particle filters also have a feature that is very beneficial when it comes to limited memory resource. Particle filters have a fixed number of particles, and the processing capacity and memory resources are linearly dependant on the number of particles. This makes the particle filter easy to use on small microcontrollers where both processing speed and memory are limited.

4.4 Implementations

A simulation was created for this implementation to work. A map consisting of squares with a width of 50 units as obstacles was created. The map can have variable sizes and variable amounts of obstacles.

A robot can roam freely in the map and off the map (where there is infinitly nothing). It may only move forward and turn, and has a laser in front, which returns a range from 0 to 80 units depending on if there is an object in front it.

In this simulator an Inverted Particle Filter was implemented. Particles (ranged

from 1000 to 10 000) were scattered around the robot in a circle with radius of 200 units. Each particle initially has a weight 0.5 and can range from 0 to 1.

For each laser measurement the laser is represented as a line from the robot with the length of the measurement. All the particles which are 1 unit away from the line will get a weight close to 0. This representing that a particle close to the laser probably is not a point on an obstacle. At the end of the line, the particles get a weight close to 1, representing that that particle is probably part of an obstacle.

After each movement the robot has, it sweeps by first turning 45°to the right and then 90°to the left and back 45°to the right. All this while taking laser measurements. This will give the robot a field of view of ±45°. Also after each movement the robot goes through a resampling phase where every particle with weight under 0.5 is resampled using the resampling wheel.

Notice that the resampling technique used in the resampling phase is based on the resampling wheel provided above, and is not optimal. It mostly removes the particles with weights under 0.5, which then again leads to particle depletion. This can be seen in the figures below by the huge blank spaces. This is good for static obstacles, but will be a problem when dynamic obstacles are introduced.

Another thing to notice is that particles that are farther away than 200 units should be removed and a new one should appear on the other end of the map. This is currently not implemented. Therefore in Figure 4.5 where the robot moves the particle "cloud" will not have the origin as centre, and not have newly generated particles coming from outside. This does not affect the particle filter itself, but must be dealt with when implementing on a physical robot.

4.4.1 SCENARIOS

The current map in the scenarios presented below is a map containing two square obstacles with a width of 50 units. The centre of these two obstacles are (25, 75) and (75, 25).

ROBOT TURNING ON THE SPOT

As seen in Figure 4.4, the robot starts in the origin with heading $\frac{\pi}{4}$. The robot turns 4 radian and takes measurement with the laser every 0.1 radian turn. It can be seen in the figure that the robot clearly detects two obstacles in the upper right corner. It can also be seen that there are unexplored areas to the left of the robot since it has not done a full turn. This is an example of how simple but effective the inverted particle filter is.

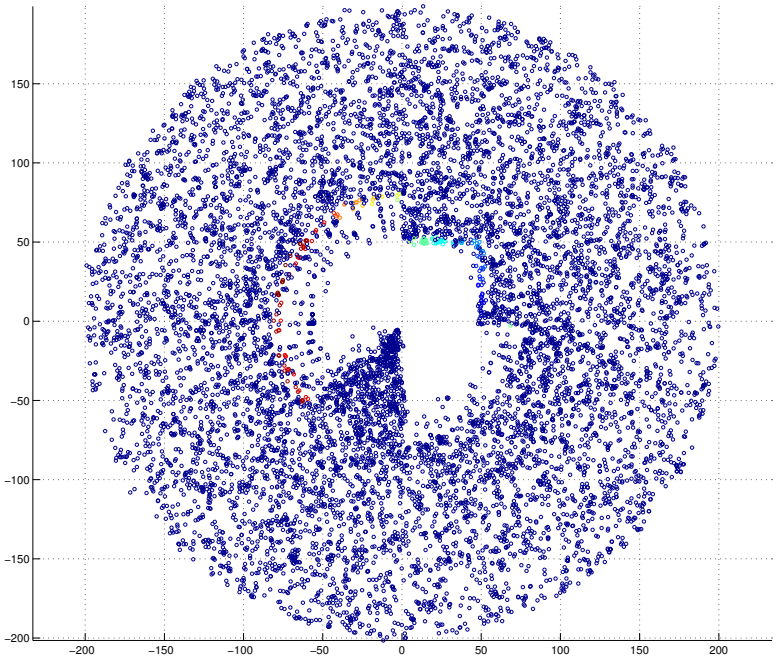

Figure 4.4: A robot turning 4 radians, starting from $\frac{\pi}{4}$ heading at position (0, 0).

Note that in the figure it can be seen that there are lines of particles within the blank area. This is due to slow sampling rate on the range measurement. The blank spaces between the lines are spaces where laser has taken a measurement and the lines are particles that has not yet been detected by the laser.

ROBOT MOVING IN CIRCLE

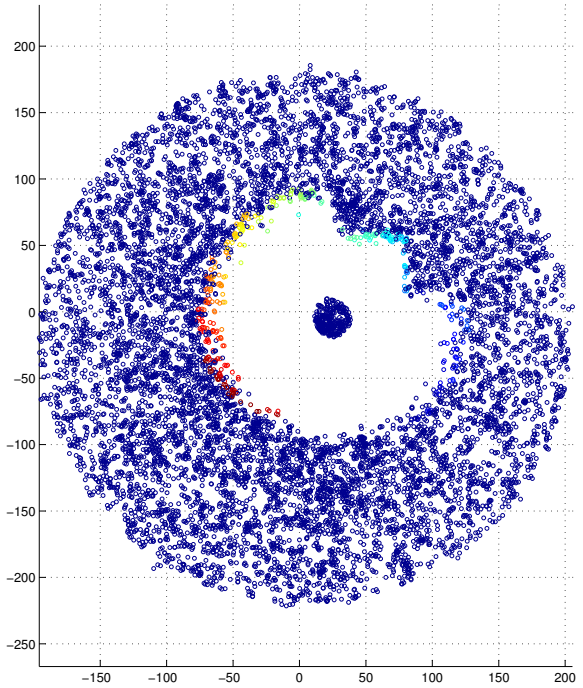

Figure 4.5: A robot moving in a circle, starting from (-25, -25) with a radius of 50 units, moving counter clock-wise. The map is set up with two obstacles. One with centre (50, 75) other with centre (100, 25. Particle count: 10 000)

As seen in Figure 4.5, the robot moves in a circle starting from (-25, -25) with a radius of 50 units, moving counter clock-wise. The blank space forms a doughnut shape. This is due to the unexplored area outside the blank space and in the centre of the map. The blank space contained particles, but where removed in the

resampling phase. The indentions in the doughnut shape are obstacles detected by the laser. Looking closely on the figure, it can be seen that the colour of the particles on the edge of the doughnut shape range from red to yellow to green to blue. Red is here represented as weight of 1 and blue as 0.5. It can therefore be seen that a particle detected recently is red, while older particles tend to be blue.

CHAPTER 5

RELATIVE POSITIONING

As mentioned in Section 2.4 it would be wise to use relative positioning to solve the positioning problem. This requires the robots to be synchronized, and some sort of sensor to measure the position.

This chapter first introduces the concept of relative positioning. Section 5.2 discuss the advantages and disadvantages of a few sensors commonly used for positioning. It is then presented a method using ultrasonic sensors to measure the relative position, followed by a protocol which could be implemented. Section 5.5 elaborates the need of synchronized clocks for the system to work.

5.1 INTRODUCTION

Relative positioning is mostly not used in robotics. This due to GPS and other systems being able to give a very accurate measurement of the position of a robot. By using GPS, some odometry, an accelerometer and other sensors and putting the information into a Kalman filter or other types of position estimators, the problem is solved. The output is roughly a position of the robot or vehicle, which can be sent out to other robots requiring this information. Mostly robots and vehicles are also very large in size and slow in movement. This gives the estimator time to create a good enough estimation, and the robots/vehicles time to handle this information.

If the robot/vehicle is situated inside, the GPS will not give any signal at all. This is not a problem if the robot/vehicle is alone. The robot/vehicle is then only required to know the position of it's origin or goal relative to itself to navigate around the environment. If there are more robots/vehicles that are cooperating, the easiest method is to create reference points that the robots can measure. This

will give a position relative to the reference points and therefore a relative position to each robot. This is usually done by using a differential GPS [Vik, 2012] or laser towers[Sperre et al., 2012]. Differential GPS (DGPS) is mostly used to enhance GPS accuracy and removes common errors. Relative positioning with DGPS works by having two or more GPS senders stationary on ground and having the robot/vehicle receiving the GPS signal. This is also called local GPS, or LPS. Laser towers are used by having a spinning laser tower on each robot, shooting two laser beams. Three or more stationary receiver towers looks for the laser beams and measures the distance to them. The robot then uses triangulation to find its position.

Another more widespread method using reference points is to use one or several cameras and a way of identifying each robot (either using lights with different colours, or unique symbols printed on each robot). A computer calculates the position of each robot using input from the cameras, and tells each robots what their position is.

These methods will not create a robust system. GPS is not accurate enough to estimate a good enough location when the robots are as small as they are, and GPS is useless inside. The problem with using reference points is that the robots are relying on sensors outside themselves. This is a huge risk, because if the reference points have errors or are not active, the robots will be immobilized. It also requires the reference points to be deployed before any tasks can be done. This not only takes time, it also limits the area the robots can operate. Using an outside computer to calculate the robots' positions is not a scalable method, as it is only a limited number of colours and symbols to be used. The robots are acting more as dumb agents being told what to do, and not a cooperative team. It also requires more set-up for the robots to work, so much set-up that it usually requires a human team to install the equipment. This is very time and cost consuming, and is almost considered cheating.

Making each robot able to only detect the relative position of its neighbouring robots purely on its own, is not easy to implement, but gives many advantages:

- **Scalability** Making the robot only able to detect its neighbours gives each robot a finite number for neighbours to deal with. This because there will at one point not be enough room in a robot's "neighbourhood". This will create a mesh network between the all robots, with the robots able to communicate through routes in the neighbourhood. The system is therefore very scalable. The problem in these mesh networks is that information have to go through many nodes to go from one end of the network to the next, thus being very slow. This is not a huge problem as a robot does not need much information about all the robots in the network (only its neighbours), and therefore does not need to send or receive information throughout the whole network.

- **Master- and slaveless** If the robots only detect their neighbours, there is no need for a master/slave system. The main problem of a master/slave system is that if a master is disabled, the system will need a method for electing a new one. If the network is spilt for some reason, there will be elected two masters, and a merge between two networks will need another re-election. In fact every change in the network will require a check in who is master in the system. The master is also required to communicate with the whole network, inhibiting scalability.
- **No reference points** As explained earlier using reference points to find the position of the robots is risky. A fault in the reference point, or a lack of reference point may potentially immobilize the robots. It also requires deployment, which can not be done in all environments. It also restricts the area the robots can operate.
- **Environment** Being able to behave consistently in different environments is key. To have one robot able to handle all sorts of environments without needing to switch out sensors or other hardware is very cost efficient. This makes a robust system. If the robot does not require others than itself to detect other robots to cooperate with, it can easily do so in all sorts of environments.

If you look towards nature, and especially insects and birds, it can be seen that relative positions are used. For locusts swarms and bird flocks to be able to fly as a team, there has to be a way for them to detect the position of everybody within the swarm. It would be hard for a locust to have a master or a reference point, and the only requirement a cricket has is the position of its neighbours for the swarm to work. The Boids algorithm[Reynolds, 1987] is an algorithm able to create swarm behaviour by only using relative position and velocity. The algorithm is easy to implement, but lacks cooperative properties. Birds and insects are not able to cooperate without having a social protocol, and are therefore unable to do tasks like picking up heavy objects as a team. Nevertheless it can be seen that relative positioning is the most robust method when using cooperative control.

5.2 Measuring relative positions

Using relative positioning, and having every robot able to detect its neighbours requires both distance measurements and triangulation. The cheapest and most used methods to measure distances are either by light, sound or imaging.

Imaging is getting more popular to use. Following here are some advantages and disadvantages to using it:

ADVANTAGES

- **Detects more** Using image processing, it is easier to detect the position of other objects as well as a robot's neighbouring robots. The amount sensors needed will therefore be less, and therefore more robust against noise.
- **Identification** It is easy to place symbols, lights or colours on each robot, giving them a unique identification. This can then be detected by imaging, making it easy to identify which robots are neighbours.

DISADVANTAGES

- **Heavy processing** Image processing requires much processing and memory resources to work. This is due to the analysing of images required to get the wanted information. This is not suited for this project, because of the constraints in processing and memory capacity (see Section 2.2). This might also make the microcontroller to slow to be able to handle the other deadlines that it has.
- **Field of view** Cameras have a restriction in the field of view, which is a problem when a robot is trying to detect everything around itself. This can be fixed by either limiting the movement of the robot to only moving forward and turning, which causes the robot to detect anything it is walking towards. It can also be fixed by making the camera sweep or spin. This will make it harder to detect moving objects, since after taking one measurement the camera has to wait for a whole sweep or spin to take another measurement at the same angle. The movement an object has had between these two measurement is unknown. The last method of fixing this problem is to use many cameras, which again raises the number of the sensors, and the power supply of the robot.
- **Learning algorithm** Image processing requires some sort of learning algorithm to work. The problem with using a learning algorithm for detecting objects is that the algorithm might not converge fast enough or not at all. This leads to the robot not detecting the objects. It is more reliable to use methods that have a proof – proving the method gives the desired result.

It can be seen here that imaging is not suited for solving the relative position problem for this project. It is more used in robots or computers capable of handling heavy computing, and mostly used for single robots.

The more unusual sensor for sensing other robots is by using light. Sensing distances with light implies adding two or more lights (preferably LEDs) and photodetector on each robot. Each robot then uses the photodetectors to detect the

strength of nearby light sources.

ADVANTAGES

- **Continuous detection** Since light travels at the speed of light, there is no delay from the light being emitted to it being detected. This means that each robot can be detected continuously without delay.
- **Detecting angles** It is easy to detect the angle of the light source by using many photodetectors and knowing the distance between them. This can be done with simple triangulation.

DISADVANTAGES

- **Identification** There is no way to distinguish one light source from another without them having different frequencies. This is a problem since this gives a limiting number of robots able to work at the same time. This also means that the light must be preprogrammed with a unique frequency, or have a unique diode soldered in.
- **Detecting distances** Detecting distances is hard using light. To detect the distance of an object emitting light, it requires the object to have more then one light source, and a known distance between the light sources. When viewing two light sources on a robot from the side, the distance between the sources vary if viewed from different angles, see Figure 5.1. This means that the angle of the robot must be known to be able to detect the distance from it.
- **Frequency** Electromagnetism has a disadvantage when it comes to frequency. If the robots who are to be detected emits visible light, then any light source in a room interferes with the sensor. This also applies to infrared and ultraviolet light when used outside. Higher frequencies like X-rays and gamma rays will be harder to produce, and potentially dangerous. The sensors are therefore limited to only using frequencies lying in the microwave and radio wave region.

Detecting using light is not easy without having information about the robots angle, and even then it would be hard. It can be argued that by only using the strength of the light source to detect the distance, and having many photodetectors to detect the angle. This is an easy and possible solution, but it still has a problem with identification. There is another way of using light. This is done by having two lasers with a known distance apart, spinning with a constant speed, on top of each robot[Sperre et al., 2012]. As the lasers travel the photodetectors

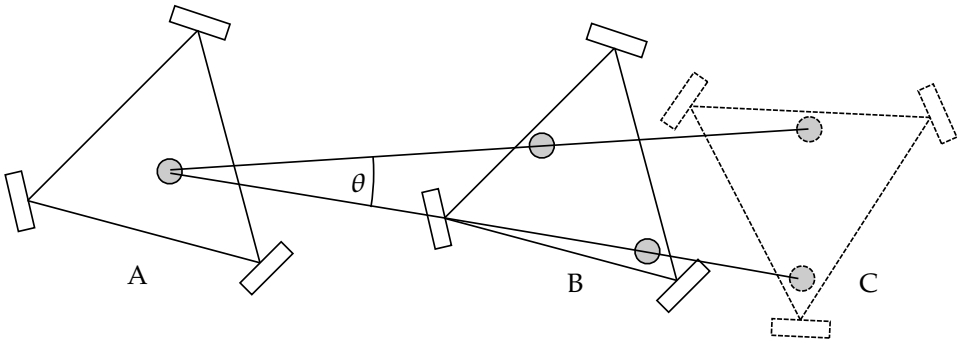

Figure 5.1: Robot viewed from above. Here an angle θ between two light sources on robot B are measured by robot A. It can be seen that because robot B has an angle, robot A can not differ if robot B is positioned at B or C

on the other robots detect two blinks that comes from the lasers as they sweep by. The time between the two blinks will give a distance, and using two or more photodetectors will give an angle. This is a good solution, but it requires much more complex measurements and electronics, and it still has the problem of identification.

Sound on the other hand has properties that makes it easy to use for measuring positions. By placing one sound transmitter and three or more sound receiver, with known distances between them, on each robot, the relative distance and angle can be calculated. This will be explained in Section 5.3. Here are some advantages and disadvantages by using sound for detecting relative positions:

ADVANTAGES

- **Speed of sound** The speed of sound is 340.29 $\frac{m}{s}$. When two microphones are placed beside each other and sound wave travels by them, the time difference between each detection is measurable with a good enough sample rate. Therefore by using triangulation and three or more microphones the angle of a sound wave can be measured easily and precisely.
- **Identification** Unlike light, the energy of a sound wave does not vary with the frequency. This means that one sound wave at 40 kHz and one at 41 kHz have the same energy given the same amplitude. Different frequencies can therefore give different identifications. Another way for identifying each robot is presented in Section 5.4.
- **Noise** Outside 20 kHz there are not many sounds in everyday life. Sounds

higher then 20 kHz are usually created by motors and electrical appliances, but at very low amplitudes. This means that there is not much measurement noise when using high frequencies.

DISADVANTAGES

- **Detecting distances** When detecting distances using sound, the time between emittance and measurement is used. This means that the time between one robot emitting a sound wave and another detecting it needs to be known to be able to calculate the distance. This requires synchronized clocks and a communication channel telling when a sound wave was emitted and when it was detected.

The simplest solution is by using sound. It is the most robust type of sensor, and the only requirement is a synchronized clock and a communication channel. The communication is already meant to be implemented which only leaves synchronized clocks.

5.3 ULTRASONIC SENSING

Placing one ultrasonic transmitter and three ultrasonic receivers on each robot, the problem with relative position can be solved. The ultrasonic transmitter is placed in centre of the robot, shown in Figure 5.2, facing such that a pulse from the transmitter creates a circle of sound waves on the horizontal plane, i.e. a water drop hitting a water surface. This can be accomplished by placing the transmitter upwards, and with a pyramid shape on top of it - spreading the sound wave, see Figure 5.3. Each receiver is placed with equal distance apart and equal distance apart from the transmitter. Applying synchronized clocks and a protocol, the relative position of every robot can be detected. The sound waves emitted are in the ultrasonic range since there is little noise in these frequencies.

For detecting a single robot, the robot only needs to send a short pulse with its transmitter, and record when it was emitted. The surrounding robots eventually receives the sound wave, and records when each of its receivers senses the sound wave. The distance from the emitting transmitter to each of the receivers can then be measured using the time recordings Figure 5.4. Here synchronized clocks are necessary.

For detecting multiple robots multiple frequencies can by used, but an alternative solution is presented in Section 5.4.

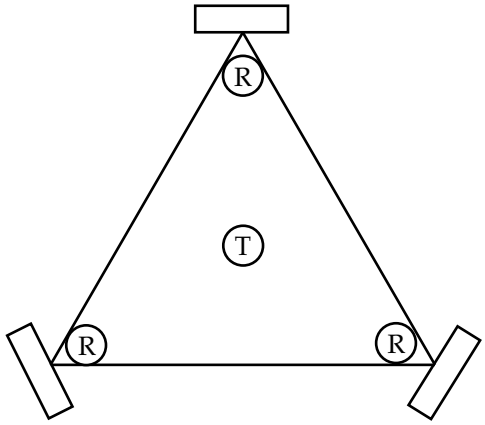

Figure 5.2: Robot viewed from above. T represents a ultrasonic transmitter, and R represents a ultrasonic receiver

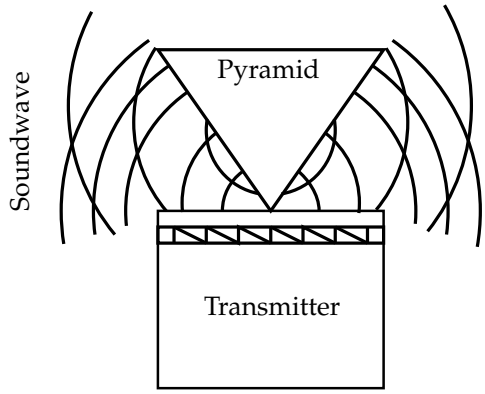

Figure 5.3: Transmitter viewed from the side. As seen a sound wave emitted is spread out into a circular wave.

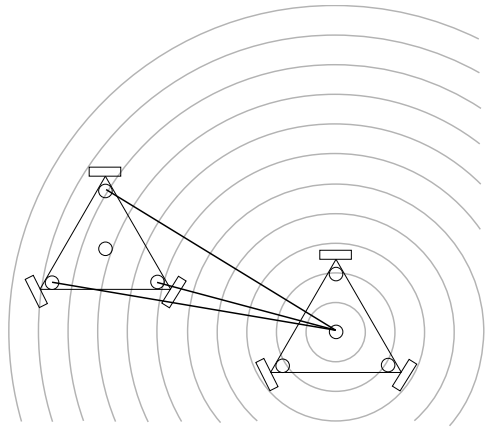

Figure 5.4: A sound wave is pulsed from one robot and the three receivers on another robot measures the pulse

5.4 PROTOCOL

To use relative positioning the robots are required to differentiate each other from another. When using sound for detecting, the first method that comes to mind is to differentiate using different frequencies. The problem with this is that it usually requires complex hardware circuits and/or fast Fourier transform for filtering the sounds. This also require the sensors and transmitters to be able to sense and produce different frequencies.

Another solution to identifying is by using a communication protocol. Since synchronized clocks has to be implemented if sound is used to measure the relative position, it would be beneficial to use it for identifying which robot emits which sound.

By implementing a time slot for when each robot is allowed to emit a sound wave, the identification problem can be solved. The robots agree on which time slot belongs to which robot, and each robot must then emit a pulse within this time slot.

As seen in Figure 5.5, there are 4 robots interacting with each other. When they enter the first time slot, the first robot sends an ultrasonic pulse. This is received at different times by each robot[1]. When the robots enters the second time slot, the second robot sends a signal. Since each robot has been given a unique time slot,

[1]Note that the robots actually receives three signals, one from each ultrasonic receiver it has. These signals are then used to triangulate the position. This has been abstracted away in the figure.

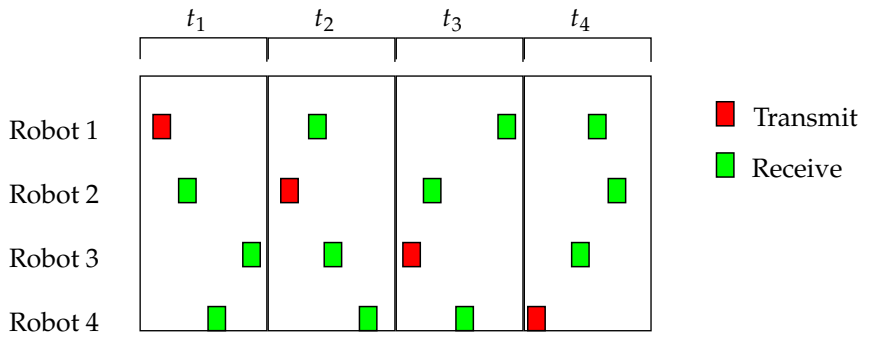

Figure 5.5: A graphical representation of the protocol. Each robot transmits a signal at the beginning of their time slot. The other robots then receives the signal. Since each robot has a given time slot the other robots know which robot sent the signal.

the other robots can identify which robot sent the signal depending on which time slot they are in.

EXAMPLE

6 robots have synchronized clocks and ultrasonic sensors as described above. In this example a time slot can have a period of $\frac{1}{6}$ seconds. This so that each cycle is completed within one second, which is sufficient for this example. When the robots enter the first time slot, the first robot emits a sound pulse and the others receives and calculates the distance to it. The sound emitted travels at a maximum of 56.72 meters before it "enters" the next time slot, which is farther then the robots normally would move away from each other.

The questions raised in the above example is what happens if one robot move to far from the other robots so they receive the sound wave outside the time slot? And what happens when there are to many robots for the time slots to be efficient?

For addressing the distance question first. The sound wave will probably diminish so it will not be detected by the other robots. If this is not the case, then there is a risk of the robot being so far away from the other robots that when its signal is received it might be received to late (i.e. some one else's time slot). This will cripple the whole system, and the other robots will not be able to identify the right robot. Having different time slots on different frequencies solves this problem. Using the example: Having one time slot at 40 kHz with a period of $\frac{1}{6}$ s and another time

slot at 41 kHz with a period of 0.5 s . Now the 40 kHz frequency can be used to detect nearby robots, while the 41 kHz frequency can be used to detect robots far from each other (up to 170 meters). It can be seen in Figure 5.5 that Robot 1 and 3 are dangerously far away from each other. A small delay in the transmission will make them miss the deadline. Here it would be wise to implement a double time slot as mentioned. There is also worth mentioning that detecting robots very far away from each other is neither necessary nor interesting. These robots might be out doing other objectives, and there is no risk in colliding with them either.

What happens when there are to many robots for the time slots to be efficient? An example of this problem might be if there were 100 robots trying to detect each other. If the whole time slot cycle still were to be 1 second, the time slots would have to be 1/100 s. It would be hard for the robots to keep the calculation deadlines, and the maximum travel range of a sound wave would only be 3.4 meters. If the time slots were to have a period of 1/10 s, the robots would be able to hold up, but the complete cycle would be 10 seconds. This means that the update rate of the positions would be 10 seconds, and is very inefficient and leads to potential collisions. The solution is again using different frequencies and different time slots. Having 10 time slots with frequency range from 40 kHz to 50 kHz with a step of 500 Hz , and a period of 1/5 s , each robot would be able to detect each other within a range of 68 meters. This means that 10 robots transmits a signal within the same time slot, but at 10 different frequencies. It is worth noting that with this amount of robots there would be other potential problems that needs to be solved before this problem occurs.

5.4.1 MULTIDIMENSIONAL SCALING

A problem that arise when measuring distances is that the measurements differ from each of the robots (Figure 5.6). This occurs because the clock on each robot differ. The error in time leads to an error in distance when detecting transmitted sound. This problem is called multidimensional scaling (MDS) when this goes over multiple dimension (i.e. multiple measured distances). It is therefore needed to calculate the most plausible distances between them. The problem is formulated as

$$\delta_{i,j} = \text{robot } i'\text{s measured distance to robot } j \tag{5.1}$$

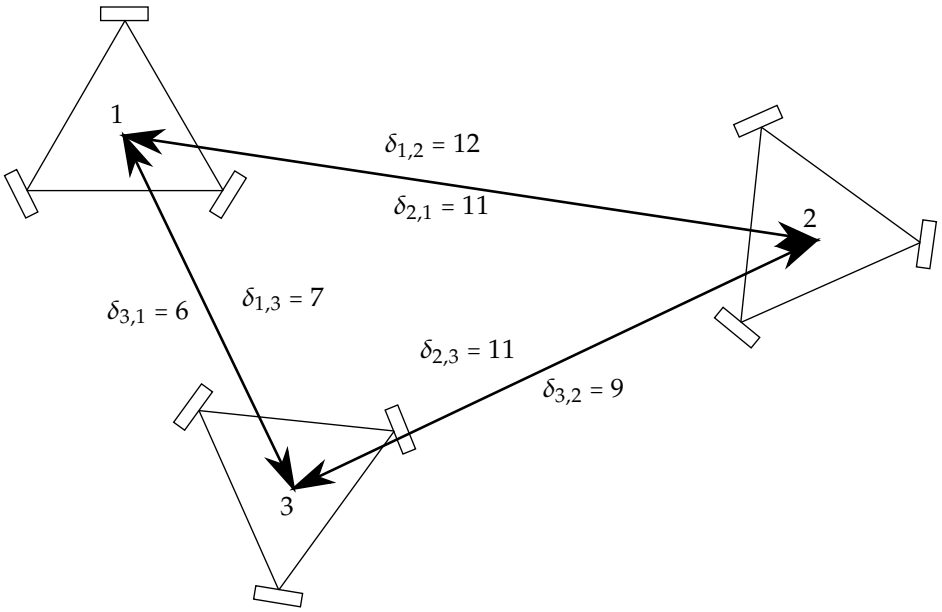

Figure 5.6: Three robots with their distances measured between each other. Equation 5.1 describes the distance δ. The distance measured from robot 1 to 2 is not the same as measured from 2 to 1

By putting all the measurements in a *dissimilarity matrix*

$$\Delta = \begin{bmatrix} \delta_{1,1} & \delta_{1,2} & \delta_{1,3} & \cdots & \delta_{1,n} \\ \delta_{2,1} & \delta_{2,2} & \delta_{2,3} & \cdots & \delta_{2,n} \\ \delta_{3,1} & \delta_{3,2} & \delta_{3,3} & \cdots & \delta_{3,n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \delta_{n,1} & \delta_{n,2} & \delta_{n,3} & \cdots & \delta_{n,n} \end{bmatrix} \tag{5.2}$$

The multidimensional scaling problem can be converted into a metric multidimensional scaling problem(MMDS), being a simpler variant of MDS. In order for this, the entries in the matrix must hold two properties: *non-degeneracy* and *triangular inequality*. Non-degeneracy can be formulated as:

$$\delta_{i,i} = 0, \quad 1 \leq i \leq n \tag{5.3}$$

While the triangular inequality can be formulated as:

$$\delta_{i,j} + \delta_{i,k} \leq \delta_{j,k} \tag{5.4}$$

Since the entries on the matrix are distances, this can easily be proven. There is also an unfortunate property in this matrix, which is:

$$\delta_{i,j} \neq \delta_{j,i} \tag{5.5}$$

There is no straight forward method for solving this problem, but a cost function for a least square problem can be formulated as:

$$\min_{x_1,\ldots,x_n} \sum_{i<j} (\|x_i - x_j\| - \delta_{i,j})^2 \tag{5.6}$$

While this problem occurs with distances, this problem will not occur when measuring angles. This is because measuring angles are not dependent on the time the sound was emitted. Since it is not dependent on this, the synchronized clocks do not have any effect on the measurements, and an error between the clocks does not affect the measurement of angles.

5.5 Synchronized clocks

The problem with the clocks on microcontrollers is that the clocks may differ. Even clocks that are initialized the same, will eventually differ due to clock drift. No clocks are the same, and will therefore at one point in time be different. This is a big problem since many computer systems that interact with each other requires to be synchronized to function properly.

In regard to the relative positioning system above, synchronized clocks are vital. An example is given in Figure 5.7. Here two robots where robot A is one second earlier than robot B. Robot A transmits its ultrasonic signal at real time t_0 (which is t_0 for robot A and $t_0 - 1$ for robot B). At real time t_1 the signal is received at robot B. The time difference between t_1 and t_0 is Δ_t. When robot A communicates that it transmitted the signal at t_0 and robot B communicates that it received at $t_1 - 1$, they calculate the time difference is given as $\Delta_t - 1$. The distance between them would then be wrong by over 340 meters. When robot B then transmits, the difference calculated would be $\Delta_t + 1$, which is another 340 meters difference in the other direction for a total of over 680 meters. Getting the clock difference down to microseconds would therefore be desirable.

One might think that for the example above that robot A tells its time to robot B, and robot B corrects its time to this, would solve the problem. The problem with this is that the transmission time of the information and the computation time would contribute to the time difference. The time difference above would be reduced to milliseconds instead of one second, but it still persists.

Better algorithms exists, and many of these have different accuracies, properties and requirements. There are two main branches of solutions:

- **Centralized systems** are systems where there is one master dictating the time to the slaves. Algorithms that use this are for instance Christian's algorithm and Berkeley algorithm.
- **Distributed systems** are systems where there is no master nor slave, and the information is distributed. This means that an average of the time has to be created.

For this project, the robots does not care about the time of the outside world. This means that the correct time is not significant, only that the difference between the clocks is very small. Therefore neither centralized nor distributed algorithms is more beneficial then the other. Therefore any algorithm can be used, but one with sub-microsecond precision is fancied. The use of the Precision Time Protocol [Mills, 1985] should be looked at more thoroughly because of this feature. It was first defined in IEEE 1588-2002, but revised in IEEE 1588-2008 with better accuracy and robustness. The Precision Time Protocol needs a reference clock. This clock usually is an atomic clock. This clock then synchronizes with other computer clocks, also refereed to as server clocks. These server clocks are then used to synchronize with client clocks. As mentioned, the robots do not require an absolute time, but a small time difference. Therefore promoting one of the robots as the reference clock and synchronize the rest of the robots to it would be sufficient. The protocol is also highly scalable, by electing many clocks to be reference clocks, which also makes the protocol fault-tolerant.

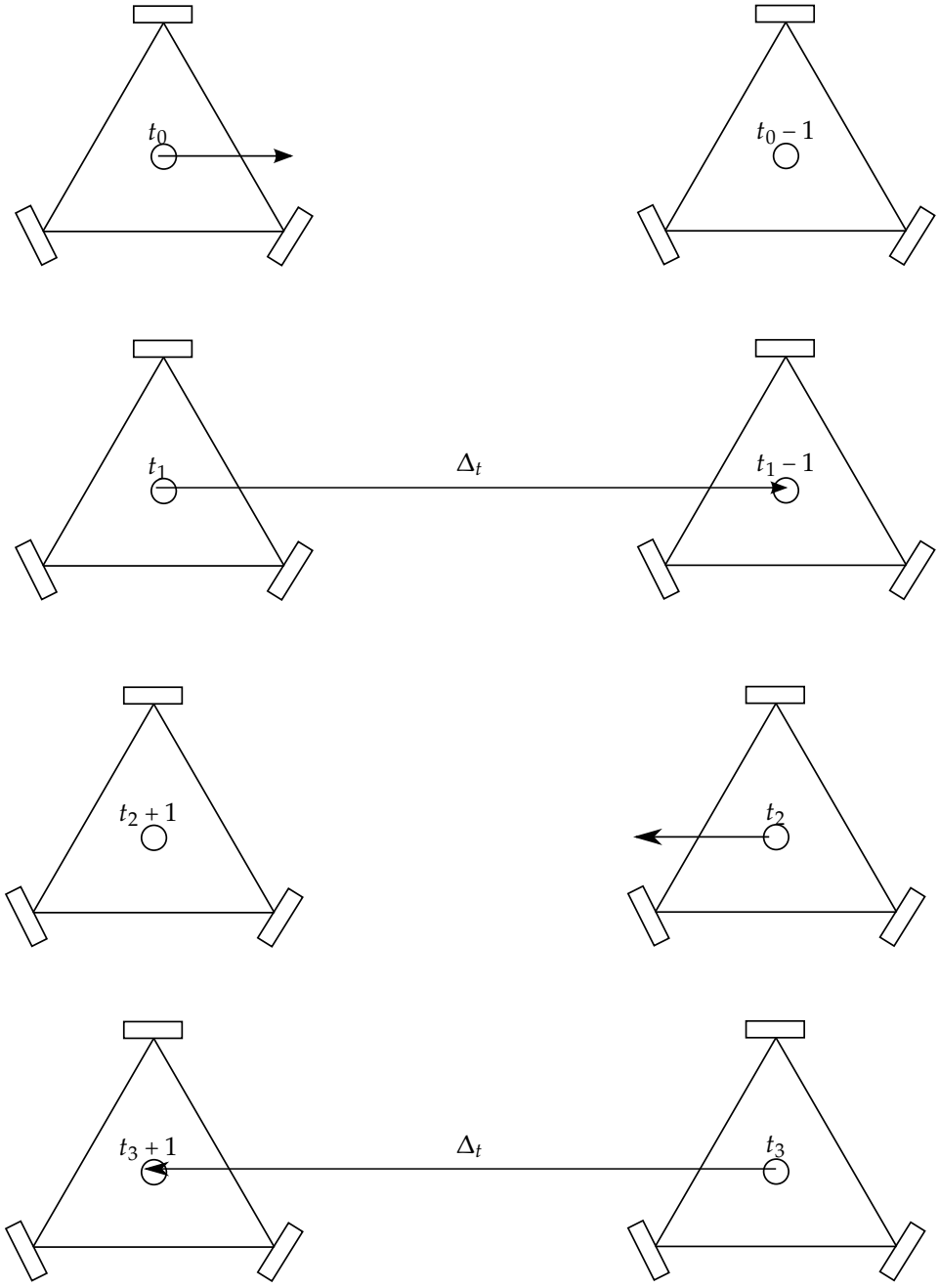

Figure 5.7: Problems with non-synchronized clocks. Robot A's clock is one second earlier then Robot B's. When Robot A transmits and Robot B receives, the real time between them is Δ_t, but the calculated difference is $\Delta_t - 1$. When Robot B transmits the calculated time difference is $\Delta_t + 1$

CHAPTER 6

DISCUSSION

This chapter will provide some hindsight on this project. In each section below each chapter will be discussed, and some hindsight and experience will be presented. After this an overall discussion of the project is presented.

6.1 PROBLEM DESCRIPTION

It could be discussed if the solution of using a mere laser and some ultrasonic sensors is sufficient for the project to work. It is true that using a microcontroller with a Linux distro and use an image processing library like OpenCL to give the robots camera vision, would be a plausible solution as well. It is true that such an approach is more applicable than the current solution, but it would require a lot more work get right. The current solution is also something which probably never has been done before, and the field of multi-robot systems would benefit from the knowledge acquired in this report. As mentioned in the introduction, this solution would be beneficial to see how simple a system could be and still have cooperative properties.

6.2 HARDWARE

The products presented in Chapter 3 can be discussed. There might be many modules able to perform better, but the time required to research this is too long for this report and maybe even this project. The proposed products are familiar to the participants of the project, and they are easily accessible.

None of the presented products have been thoroughly tested, although the Arduino Due is related to the Arduino Mega, which is a known platform by the participants of the project. The experiment on the IR-laser was disappointing (presumably not the laser's fault), and other than that, the only component tested is the X-Bee, which has yet only been tested for two-way communication.

Due to time constraints and a minimal budget, most of the presented hardware was not obtained, and therefore not tested. This is not desirable when deciding on an implementation, but the research on the components show positive results.

6.3 Obstacle Detection

It can be discussed if the approach of using a laser for obstacle detection is enough, and if only a local map is needed for the robots to be able to navigate and explore the environment.

As seen in the simulations (Section 4.4), a system consisting of only a laser and an Inverted Particle Filter gives promising results for a single robot. This is sufficient for obstacle detection, but a thought of reusability of maps rise. As discussed in Section 2.4.1 a merge of local maps would be beneficial for the total system. Since the Inverted Particle Filter creates a circular map, and the relative positioning system gives a distance between the robots, it can easily be checked for collision between the local maps. When two local maps overlaps the particles can be compared. This can be accomplished by temporarily adding the overlapping particles to the maps, and let the resampling function keep the particles if they add information to the map.

The Inverted Particle Filter (presented in Section 4.3) has its flaws when it comes to dynamic obstacles. This is due to the resampling function. The resampling function can however be constructed for any specification, and can therefore be used in this project. A robot might be able to move around with the current implementation of the particle filter, but it would require some more trial and error for the robot to move efficiently. There has yet to be anything indicating that a resampling function is not able to handle dynamic obstacles, so this study will be done at a later stage of this project.

Another implementation that would be beneficial to the obstacle detection system is to implement the obstacles as single objects rather then a set of particles. This can be done with some simple line or corner detection algorithms on top of the Inverted Particle Filter.

6.4 Relative positioning

The method for detecting relative position using ultrasound that was presented in Chapter 5 is only theoretical. No implementation of the method exists yet, so if it works remains to be seen. The test presented in 3.7 reveals promising results for detecting sound waves, but an experiment in triangulation has not been conducted yet. If the experiment gives negative results, the positioning system must be re-evaluated. Neither the protocol for synchronized clocks or identification has been tested, nor has the multidimensional scaling problem been looked thoroughly into. There are potentially many problems in the implementation of this method that has not been discovered yet, and only time will tell if this method is plausible, though the research is promising.

6.5 Overview

The hardware platform presented in Chapter 2 seems plausible to implement. Sadly the presented solution is highly theoretical, and potential problems may arise. The relative positioning system is purely theoretical, and the obstacle detection system has some flaws in the Inverted Particle Filter.

Though there are many theoretical aspects of this report, most of the experiments done have given promising results. There has not yet been any evidence that the presented solution would not work, and therefore the next step is to test the system in a practical manner. By investing in the given hardware components and testing them individually and collectively, the plausibility of the system will strengthen. While the project might stumble upon many problems in the future, the research conducted supports the possibility of successfully implementing the current solution.

CHAPTER 7

CONCLUSION

A project started fall 2012 with the purpose of creating a functional multi-robot system for demonstrations and educational purposes. In this report there has been an investigation in how simple a multi-robot system can be and still be able to handle different cooperative control systems.

A solution that is plausible has been presented, and different hardware components, methods and algorithms that supports the solution has been presented as well.

The solution presented contains a planar holonomic robots moving using omni-wheels. The robots have an IR-laser which measures the range to obstacles, complimented with an Inverted Particle Filter, to create a local map of these obstacles. The robots also use ultrasonic transmitters and receivers to calculate the relative position of each other for cooperation.

The hardware components for the platform has been discussed. The components that are key for creating this platform, and the requirements for each of them has been thoroughly discussed. Recommended products for each component, which support the requirements has been presented. Tests on some of these products has been done, and experiments that support the use of some products has been conducted. All this is present in the report.

In this report it has been a research on how the robots should interact with obstacles. The obstacle detection system operates using IR-laser to measure the range from the robot to an obstacle, and a new method called the Inverted Particle Filter, which creates a local map of the robots surroundings. The Inverted Particle Filter use the input from range-measurements to create this map. This method has also been tested in a simulator and the results are positive, though some further work is needed to improve the implementation.

The positioning system has been discussed, and a conclusion to use relative position has been taken. This is due to the constrains of the project and the scalability of the system. Different methods and sensors used to solve the problems of relative positioning have been presented and discussed. It has been concluded to use a ultrasonic transmitter, three ultrasonic receivers, synchronized clocks and triangulation to find the distance and angle between each robot. This method is supported by the use of a protocol for sending sonic pulses, so that the robots can identify each other. A short look upon Multidimensional Scaling for comparing the measurements and calculate the positions has been conducted.

A deeper understanding of what problems lie ahead, when trying to implement this multi-robot system has been achieved. It is possible to make a multi-robot systems which is very simple in implementation and low in cost. A fully cooperative control system that has both obstacle avoidance and formation control can be implemented on this platform. The platform can provide a positioning and obstacle detection system which a cooperative control system has an interface to. The hardware platform is simple and to many degrees scalable, but still highly theoretical.

Some of the methods discussed in the report might only be suitable for this project, and larger robots with fewer constraints and other purposes would probably benefit more from other methods. With larger robots it would be more beneficial to use faster computers and more complex sensors, making it possible to use more intricate methods for obstacle detection and positioning. It would also be wise to use multifarious methods to make the system more fault-tolerant.

This report gives insight on how simple a functional multi-robot system can be, and provides such a solution.

CHAPTER A

APPENDIX

A.1 MPU6050 BREAKOUT BOARD

For the use of the MPU6050, it was required to create a breakout board. The board was based on the "Tripple Axis Accelerometer 6 Gyrp Breakout - MPU6050" board provide by SparkFun. Most of the schematics are the same, but it is configured to better suit the HMC588L Magnetometer. The board is smaller in size than the one provided by Sparkfun, and has a GND and VCC out with the additional I^2C so that the HMC588l Magnetometer could fit right on top of the breakout board. The board was produced using the ProtoMatS62, but an unfortunate error in the set-up gave a malfunctioning board. Due to time constraints a new board has not been produces.

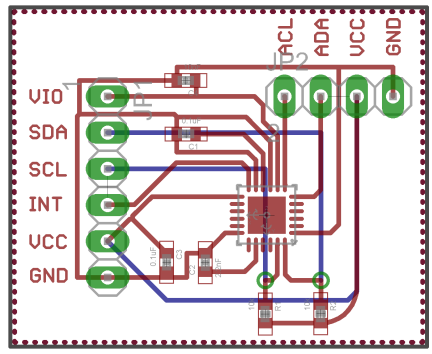

Figure A.1: The MPU6050 breakout board

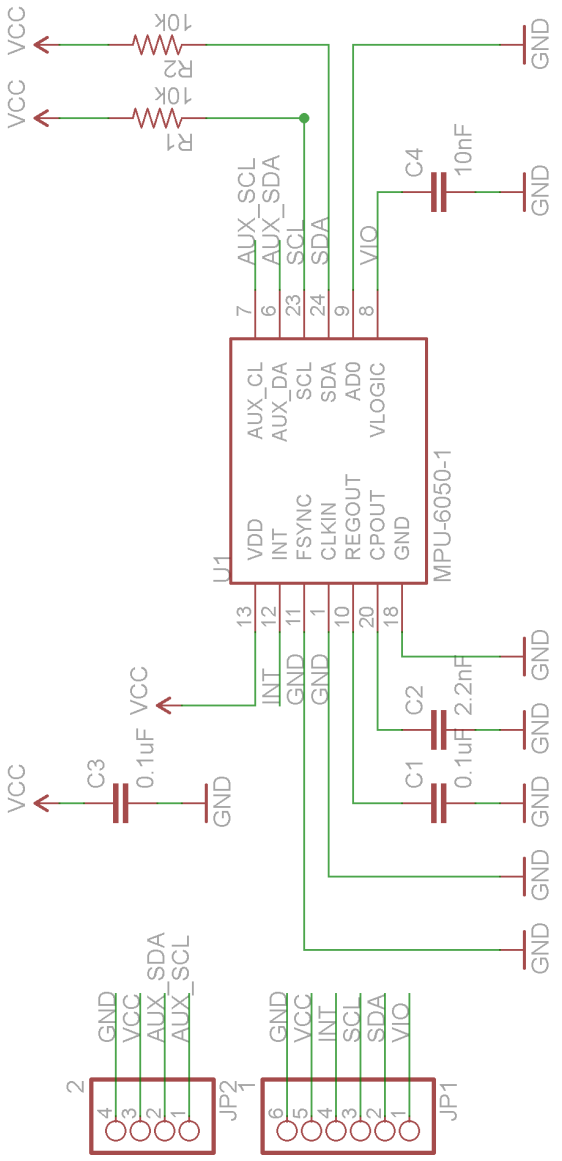

Figure A.2: The schematic of the MPU6050 breakout board

BIBLIOGRAPHY

[Arcak, 2007] Arcak, M. (2007). Passivity as a design tool for group coordination. *Automatic Control, IEEE Transactions on*, 52(8):1380–1390.

[Burgard and Moors, 2000] Burgard, W. and Moors, M. (2000). Collaborative multi-robot exploration. *. . . and Automation, 2000 . . .*, (April).

[Chae and Yu, 2010] Chae, H. and Yu, W. (2010). Artificial landmark map building method based on grid SLAM in large scale indoor environment. *2010 IEEE International Conference on Systems, Man and Cybernetics*, pages 4251–4256.

[Davison et al., 2007] Davison, A. J., Reid, I. D., Molton, N. D., and Stasse, O. (2007). MonoSLAM: real-time single camera SLAM. *IEEE transactions on pattern analysis and machine intelligence*, 29(6):1052–67.

[Faludi, 2011] Faludi, R. (2011). *Building Wireless Sensor Network*. O' Reilly.

[Klausen, 2012] Klausen, K. (2012). *Coordinated and cooperative control of a multi-robot system*. Student project, NTNU.

[Kraft et al., 2008] Kraft, M., Schmidt, A., and Kasinski, A. (2008). High-speed image feature detection using fpga implementation of fast algorithm. *Proc. VISAPP (1)*, pages 174–179.

[Lucidarme, 2002] Lucidarme, P. (2002). Implementation and evaluation of a satisfaction/altruism based architecture for multi-robot systems. *Robotics and Automation, . . .*, (May):1007–1012.

[Mills, 1985] Mills, D. (1985). Network time protocol (NTP). *Network*, (September):1–15.

[Mullane et al., 2011] Mullane, J., Vo, B.-N., Adams, M., and Vo, B.-T. (2011). *Random Finite Sets for Robot Mapping and SLAM*.

[Pagello et al., 1999] Pagello, E., D'Angelo, A., Montesello, F., Garelli, F., and Ferrari, C. (1999). Cooperative behaviors in multi-robot systems through implicit communication. *Robotics and Autonomous Systems*, 29(1):65–77.

[Reynolds, 1987] Reynolds, C. W. (1987). Flocks, herds and schools: A distributed behavioral model. *ACM SIGGRAPH Computer Graphics*, 21(4):25–34.

[Sperre et al., 2012] Sperre, A. H., Halvorsen, A., and Myren, S. R. k. (2012). *Eurobot NTNU 2012*. Master thesis, NTNU.

[Studi et al., 2006] Studi, D., Cassino, D. I., Universit, A., and Arrichiello, F. (2006). Coordination Control of Multiple Mobile Robots. (November).

[Thrun, 2006] Thrun, S. (2006). The Graph SLAM Algorithm with Applications to Large-Scale Mapping of Urban Structures. *The International Journal of Robotics Research*, 25(5-6):403–429.

[Tuci et al., 2006] Tuci, E., Groß, R., and Trianni, V. (2006). Cooperation through self-assembly in multi-robot systems. *. . . and Adaptive Systems . . .* , 1(2):115–150.

[Vik, 2012] Vik, B. r. (2012). *Integrated Satellite and Inertial Navigation Systems*. Department of Engineering Cybernetics, NTNU.

[Wiggins, 2006] Wiggins, R. (2006). Myths and realities of Wi-Fi mesh networking. *Yankee group report*, (February 2006).