



NTNU – Trondheim
Norwegian University of
Science and Technology

Hardware Platform for a Multi-Robot System

Research and implementation of a
multi-robot system with a cooperative
behavioral control system

Adam Leon Kleppe

Master of Science in Engineering Cybernetics

Submission date: June 2013

Supervisor: Amund Skavhaug, ITK

Norwegian University of Science and Technology
Department of Engineering Cybernetics

Assignment

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

- Study theories and current implementations of multi-robot systems.
- Research requirements and constraints for the multi-robot system developed at ITK, and propose a solution for this system.
- Propose a hardware platform for the multi-robot system, and as necessary investigate the different hardware components required to implement the platform.
- Research and develop an obstacle detection and mapping method for the multi-robot system.
- As deemed, conduct experiments and simulations to investigate the feasibility of the implementation.
- As time permits, implement the proposed system.
- Evaluate the implemented system.

Abstract

A small demonstration lab with a multi-robot system is being developed at ITK. This thesis considers the development and implementation of a hardware and software platform for this multi-robot system. An overview of the project and system is presented, followed by a walk-through of the hardware components required and used in this system. A brief overview of the software system is presented. The problems within, requirements of and solutions to obstacle detection methods is presented, followed by a conference paper, written by the author of this thesis, introducing the Inverted Particle Filter, which is an obstacle detection and mapping method developed for resource constrained systems. Previous work on positioning systems is presented and the positioning system for this multi-robot system is proposed. The overall project and system is discussed and evaluated.

Sammendrag

Et liten demonstrasjonslaboratorium med et multi-robotssystem er under utvikling hos ITK. Denne avhandlingen tar for seg utviklingen og implementeringen av en maskinvare- og programvareplattform for dette multi-robotssystemet. En oversikt over prosjektet og systemet er presentert, fulgt av en gjennomgang av maskinvarekomponentene som kreves og er brukt i dette systemet. Et kort overblikk av programvaresystemet er presentert. Problemene rundt, kravene til og løsningene for hinderdeteksjonsmetoder er presentert, fulgt av en konferanseartikkel, skrevet av denne avhandlingens forfatter, som introduserer Inverted Particle Filter metoden, som er en hinderdeteksjons- og kartleggingsmetode utviklet for ressursbetingede systemer. Tidligere arbeid av posisjoneringssystemer er presentert, og posisjoneringssystemet for dette multi-robotssystemet er foreslått. Det totale prosjektet og systemet er diskutert og evaluert.

Acknowledgement

I want to thank my family, friends and especially my girlfriend for encouragement and support.

Thanks to everybody at NTNU for help and guidance, and for making NTNU a home for me.

I would also like to thank Amund Skavhaug, supervisor, mentor and friend.

Adam Leon Kleppe

Contents

Contents	ix
List of Figures	xiii
List of Tables	xvii
1 Introduction	1
1.1 System overview	3
1.2 Project overview	5
1.3 Contribution and scope of this thesis	5
1.4 Organization of this thesis	6
2 Overview	9
2.1 Motivation	9
2.1.1 Constraints	10
2.2 Proposed solution	11
2.2.1 Movement	12
2.2.2 Computational unit	12
2.2.3 Communication	13
2.2.4 Obstacle detection and positioning system	13
2.3 Implemented solution	13
2.4 Future work	15
3 Hardware Components	17
3.1 Computational Unit	17
3.1.1 Candidates	18
3.1.2 BeagleBone	21
3.2 Movement	23
3.2.1 Omniwheels	23
3.3 Communication	23
3.3.1 XBee	25

3.4	Obstacle detection	26
3.4.1	Sharp GP2Y0A21YK IR Range Sensor	27
3.5	Position estimation	28
3.5.1	IMU	28
3.5.2	Ultrasonic sensors	29
3.5.3	External camera	30
4	Software systems	31
4.1	Overview	31
4.2	Obstacle detection system	33
4.3	Positioning system	33
4.4	Motor system	33
4.5	Communication system	33
4.6	Control system	34
4.7	Manager system	35
5	Obstacle detection system	37
5.1	Previous work	37
5.1.1	SLAM	39
5.2	Inverted Particle Filter	46
6	Positioning system	63
6.1	Previous work	63
6.2	Relative positioning	65
6.2.1	Measuring relative position	66
6.3	Relative positioning system	70
6.3.1	Trilateration	72
6.3.2	Multidimensional scaling	73
6.4	Identification	76
6.5	Synchronized clocks	80
6.6	Implementation	83
6.6.1	Distance measurement	83
6.6.2	Visual system	85
7	Discussion	89
7.1	Overview	89
7.2	Hardware components	90
7.3	Software systems	90
7.4	Obstacle detection system	91
7.5	Positioning system	91
7.6	Final notes	92
8	Conclusion	93

Bibliography	95
---------------------	-----------

List of Figures

1.1	The relationship between the different subsections of the project	4
1.2	The flow between the different subsections of the project	4
2.1	An overview of the proposed solution.	11
2.2	The robot viewed from the top displaying the location of the main components	12
2.3	The implemented solution	14
2.4	Images of the first prototype and implementation of a robot	14
3.1	The Arduino Mega 2560 viewed from the front. Image provided by arduino.cc	19
3.2	The UC3-A3 Xplained viewed from the front. Image provided by atmel.com	20
3.3	The Raspberry Pi Model B viewed from the front. Image provided by digitaldiner.blogspot.no	21
3.4	The BeagleBone viewed from the front. Image provided by digitaldiner.blogspot.no	22
3.5	The omniwheels used by the robot. Image provided by kornylak.com	23
3.6	Characteristics of the omniwheels	24
3.7	The XBee Series 2 is a module using ZigBee as a protocol for wireless communication. Image provided by sparkfun.com	25
3.8	The Sharp GP2Y0A21YK IR Range Sensor is the range sensor used on the robot to detect obstacles. Image provided by sparkfun.com	27
3.9	HMC5883L is a triple-axis magnetometer. Image provided by sparkfun.com	29
3.10	The 400SR100 and 400ST100 are respectively a ultrasonic receiver and transmitter in the 40 MHz range. Image provided by prowave.com	29
3.11	The Kinect motion sensor. Image provided by pressfire.no. Some content of this image has been removed	30
4.1	Overview of the software systems	32

4.2	A diagram of the reactive paradigm	34
5.1	Example of loop closure. The orange arrow shows the actual path of the robot, while the green is the path measured by the robot. When the robot measures that it is back to its start location, it is not actually in the start position.	40
5.2	A low and high resolution map	41
5.3	An example of a problem with shifting arrays. The robot moves, causing the array to have to shift accordingly. This example with 16x16 cells and shifting a total of 7 units results in a total of 1792 shift operations.	42
5.4	A robot moving from x_0 to x_1	43
5.5	A robot moving from x_1 to x_2	43
5.6	An obstacle being detected at each of the three poses. Note that in this example the obstacles and poses is on an one-dimensional line, while the image is portrayed in two dimensions to enhance clarity .	44
5.7	A quadcopter capturing images of a terrain. The quadcopter uses monoSLAM so each image helps create a map which also estimates the position of the quadcopter	45
6.1	An angle θ between two light sources on robot B are measured by robot A. If the angle of robot B is not known, robot A cannot determine if robot B is in position A or B	69
6.2	Robot viewed from above. T represents a ultrasonic transmitter, and R represents a ultrasonic receiver	71
6.3	The set-up of the transmitter	71
6.4	A sound wave is pulsed from one robot. The three receivers on another robot measures the pulse	72
6.5	An example of trilateration. The coordinate of the black cross is the intersection of the three circles created by the distances measured from each of the known coordinates (red, green, blue) to the black cross	74
6.6	Three robots with their distances measured between each other. Equation 6.4 describes the distance δ . The distance measured from robot 1 to 2 is not the same as measured from 2 to 1	75
6.7	Each robot transmits a signal at the beginning of their time slot(Red square). The other robots then receives the signal (Green square). Since each robot has a given time slot the other robots know which robot sent the signal.	77
6.8	Example of identifying robots when the robots are to far away from each other. Robot 1 and 3 do not listen to each other in Schedule 1, but do in Schedule 2. Robot 2 and 4 do not listen to Schedule 2 . .	79

6.9	Problems with non-synchronized clocks. Robot A's clock is one second faster than 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 $\Delta_t + 1$	81
6.10	A synchronization action with the Precision Time Protocol	83
6.11	The experiment conducted of measuring distances. The Arduino Mega sends a pulse to the signal generator circuit, which generates an ultrasonic pulse. This pulse is received by the ultrasonic receiver and amplified by the amplifier circuit. The amplified signal is measured by the Arduino Mega. The time between when the pulse is sent and received is measured.	84
6.12	The orientation of the robots are found by calculating the angle between the centre of the green and yellow colour	85
6.13	The camera views the different robots, and identifies each robot with their colour, and use the yellow colours to measure the orientation .	86
6.14	Images of the contours of the coloured paper	87

List of Tables

6.1 Segment of measurements from the conducted experiment	85
---	----

Chapter 1

Introduction

In recent years there has been a huge increase in the need of autonomous robots for exploration. An recent example of this is Curiosity Rover from NASA[NASA, 2012]. 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 are no easy solutions to this problem. A robot requires many sensors, mobility suitable for the environment, and algorithms and controllers for it to be able to complete a mission on this scale. To further complicate this problem, 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 such an operation, it would require heavy computing, and numerous sensors, making the total system very complex. Therefore a new branch in solutions has 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 system is more efficient than one robot, but it also introduces more problems: The robots within the multi-robot system would need to be able to communicate and coordinate with each other in an efficient manner.

There are many solutions to multi-robot systems ([Burgard and Moors, 2000], [Davison et al., 2007], [Lucidarme, 2002], [Thrun, 2006], [Pagello et al., 1999], [Tuci et al., 2006]), each with different purposes, tasks and implementation methods. Multi-robot systems can roughly be split into two categories: *Swarm systems* and *autonomous cooperative systems*.

A swarm system is a system composed of individual robots. Normally these robots are identically build and have the same properties. The robots in a swarm system

do not normally cooperate with each other, but rather focuses on doing individual tasks in parallel. Examples of such a system could be robots tracking animal population or mapping environment by taking image samples while flying over large area of terrain. The robots in a swarm does not need to cooperate, since they do not have to handle the images taken by each other. A robot only needs to take an image and estimate the position it took the picture. An external computer could collect the images and inform which part of the terrain has not been covered, commanding which robot should go where.

Autonomous cooperative systems cooperate on tasks, and has to communicate with each other to complete a task. An example task would be holding a formation, where the robots need to figure out where to move in order for the formation to be held. The autonomous cooperative systems can have many tasks that need to be cooperated upon, and the combination of these tasks can potentially make the robots able to perform complex missions. An example of such a system is a robot cooperating on performing a rescue mission. Some of the tasks that are required in such a mission is obstacle avoidance, formation holding, searching, going to a target and maybe a retrieve task. In this thesis, only the autonomous cooperative system will be covered, and the term multi-robot system in the thesis will mostly focus on the autonomous cooperative aspect.

The complexity of multi-robot systems escalate quickly. One cause of this is that each robot require sensors for manoeuvrability and collision avoidance, and many of these sensors on one robot may create disturbance and/or faulty measurements on the other robots' sensors. Each robot are also required to communicate state and sensor data to each other. The more robots that are in the network the larger the throughput of information is. The sensors combined with the communication system can quickly become complex, costly and potentially be high maintenance. It is therefore desirable to look at how low in complexity such a system can be and still have cooperative properties. This will be more cost and power efficient, as well as making room for bigger and more flexible system.

An educational project was established at NTNU in fall 2012 to create a multi-robot system. The goal of this project was to see how low-cost a multi-robot system could be, and how to reduce the complexity of a multi-robot system. 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. The robots are also going to be used as a demonstration tool, which limits both the size of each robot, and time and space required to set up the demonstration. The goal was to create 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.

The goals of the project can be split into two. The main goal is to study and

develop a multi-robot system focusing on cooperation. The secondary goal is to design the multi-robot system so that it can be used in demonstrations. The secondary goal influenced the development and concept of the multi-robot system with both desired features and restrictions to the system. In the end though, the secondary goal cannot be complete unless the main goal is. This means that in order for the main goal to be finished in time, some of the features and restrictions may be compromised.

1.1 System overview

There are several aspects in the multi-robot system developed in this project, which require special attention. It is therefore convenient to divide the system into smaller subsections:

- **Control System** A cooperative behavioural control system that controls the behaviour of each robot, making them able to cooperate on the desired tasks.
- **Platform** A hardware and software platform able to provide data and an interface, making it able to accommodate implementations of different cooperative control systems, and able to communicate with other robots or external computers.
- **Sensor Systems** The required sensor systems able to detect other robots and obstacles, which is fed as input to the control system.

Note that the subsections described above are to be implemented on each robot. Therefore each robot has a control system, a hardware and software platform and sensor systems. A visual representation of the subsections can be viewed in Figure 1.1. It can be seen in the figure that the sensor systems are part of the hardware platform. This is due to the fact that the hardware platform requires sensor systems to function. Also seen in the figure is that the software platform will have an interface to communicate with the control system.

A flow chart of the system can be viewed in Figure 1.2. In the figure it can be seen that the platform collects and processes the data from the sensor systems, it then feeds the input to the control system. The control system then sends control signals back to the platform. The platform applies the control signals to e.g. the wheels, and communicates the state of the system to the network.

There will only be a brief overview of the implemented control system in this thesis. For a more thorough discussion, the full implementation is described in [Klausen, 2013]. In this thesis there will rather be a thorough walk-through of the hardware platform with focus on the solutions and implementations.

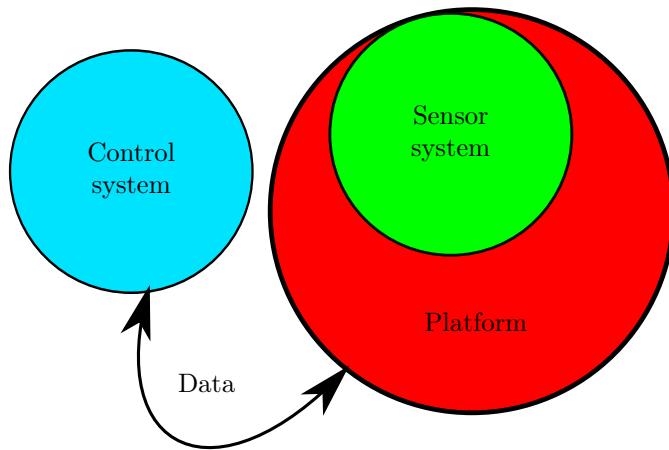


Figure 1.1: The relationship between the different subsections of the project

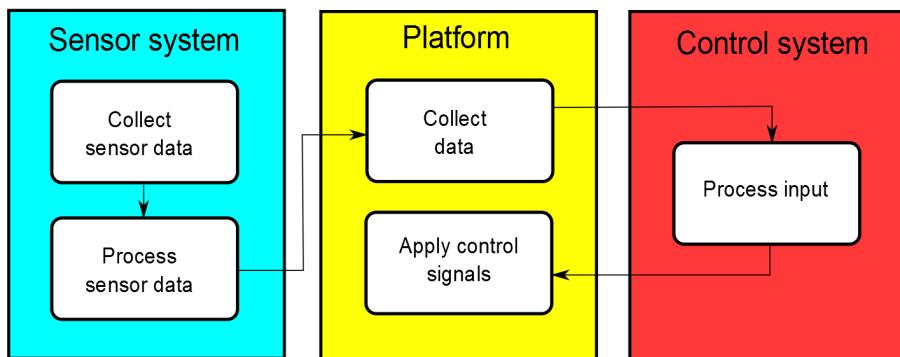


Figure 1.2: The flow between the different subsections of the project

1.2 Project overview

The project was started in fall 2012 with a pre-study[Kleppe, 2012] where the plausibility of the implementation of the multi-robot system, and the requirements for it to be implemented was presented. A simulator and different control systems were also developed to test which control system was suited for the project.

After the pre-study was conducted the main project started. There were a total of 8 participants in the project.

- One participant was responsible for developing a control system which could be implemented on the multi-robot system.
- One participant, the author of this thesis, was responsible for developing a hardware and software platform and sensor systems which accommodated the control system.
- Six participants were responsible for producing six robots with the given hardware specifications and develop a user interface which would be used in the demonstrations.

1.3 Contribution and scope of this thesis

This thesis will present a possible solution for a low-cost system platform for a multi-robot system. The proposed methods and approaches are preferential to the motivations and restrictions of the project. These methods and approaches will be presented with focus on their theoretical advantages. The implementation of each of the aspects in the hardware platform will be presented, followed by a discussion of these. Note that, though the proposed solution is *theoretically* preferred in this project, compromises had to be made to some of the implemented aspects. Alternative solutions had to be implemented in order for the main goal to be accomplished. This is detailed further at each relevant chapter.

As this thesis starts where the pre-study ended, the goal of this thesis is to present the implementations of the system that was proposed in the pre-study. The implemented system has been evaluated and compared to the solution presented in the pre-study. The main contributions of this work is within the following topics of the multi-robot system:

- **System platform** The pre-study proposed a solution for a multi-robot system. This system has been implemented to the extent it was possible. The results of the implementations and the changes that have been done will be presented.

- **Obstacle detection system** In order for the multi-robot system to detect and avoid obstacle, an obstacle detection system was required. A method for this was proposed in the pre-study, but was still very theoretical. In this thesis the full method and the conducted experiments are presented.
- **Positioning system** Due to time constraints the positioning system proposed in the pre-study was not tested. Some of the aspects in the proposed system has been developed and tested, but because of time constraints a different positioning system was developed.

Some of the contents from the pre-study report has been reworked, extended and included in this thesis. For convenience, the full report of the pre-study is included in the digital appendix.

1.4 Organization of this thesis

Since there are many topics discussed in this thesis, each topic is divided into smaller individual chapters to provide for better readability. Each chapter is to a certain degree self-contained, but for a more complete view of the system the reader is encouraged to read the chapters in the given sequence. Some chapters and sections contain discussions and results in order to enhance readability. Chapter 7 contains a discussion with a broader view concluding the resulting implementation of the system.

The thesis first introduces an overview of the proposed solution from the pre-study. Chapter 2 introduces the motivations and restrictions of the projects, followed by a brief overview of the proposed solution and the main aspects within. The chapter then presents the implemented solution, with a brief description of the inevitable complications that occurred during development, and why some of the aspects in the solution were changed. The chapter ends with a short description of the future work on the project.

Chapter 3 presents the hardware components used in this project. Each section features an aspect within the solution, and the hardware components that are relevant for this aspect. Some sections presents alternative hardware components with advantages and disadvantages to these.

Chapter 4 features the software systems and their contributions and roles within the the multi-robot system. Each section features an individual part of the software system. It is worth noting that not all of the presented systems reflect the actual implementation completely. Many of the features and subsystems have been abstracted away to enhance clarity.

The obstacle detection method will be introduced in Chapter 5. Alternative approaches and their advantages and disadvantages are discussed, followed by a conference paper¹ presenting a new obstacle detection method called the Inverted Particle Filter.

Chapter 6 features the positioning systems. Previous work and the main branches of solutions will be discussed, and the problems with these will be presented. A positioning system using ultrasonic sensors to detect the robots relative positions is then presented, as well as what is required to implement this system. The implementations and conducted experiments are presented at the end of this chapter.

This thesis ends with a chapter discussing the loose threads from the previous chapters and the overall implemented system.

The digital appendix included in this thesis contains the report from the pre-study conducted, the conference paper describing the Inverted Particle Filter, the source-code for the applications running on the robots as well as for the external camera application. It also contains schematics of different circuits used in this project and videos of the robots operating. For further detail read the included **README.txt** file.

¹Submitted to SafeComp DECS'13 <http://conf.laas.fr/SAFECOMP2013/?q=node/28>

Chapter 2

Overview

There are many solutions for creating multi-robot systems, and there are no limits on how these systems can be created. There are however, constraints in this project, which limits the possible solutions. This chapter discusses motivation and constraints to this project, the proposed solution from the pre-study and the implemented solution.

This chapter starts with a definition of what the main motivations regarding this project are, as well as the constraints within it. The chapter proceeds with presenting a possible solution to this project, the implemented solution and the differences between them. The chapter ends with a brief discussion on the future work of this system.

2.1 Motivation

As mentioned in Chapter 1, the main motivation for this project was to create an autonomous cooperative multi-robot system. The goal is to demonstrate the advantages of using a multi-robot system, and the requirements for its implementation. This means that the main priority is to create a hardware and software platform making it possible to implement a cooperative behavioural control system, and present the components required for such a system.

The secondary motivation is to make this system able to perform as a demonstration tool. It is this motivation that applies the most constraints to the project, limiting the solutions. While these constraints may be dominating, they may be compromised in order for the multi-robot system to work.

The solution proposed in the pre-study satisfies the given constraints, the implemented solution however, does not, though the implemented achieved the main goal.

2.1.1 Constraints

There exists many solutions for multi-robot cooperative systems. However the focus in this project is to see if a multi-robot system can be developed with low cost, and still have cooperative properties. Restricting the robots to be small in size and low-cost are the main priorities of the project. The size is restricted due to the robots being used as a demonstration tool. If the robots are too heavy or big, they will be hard to transport, and may have problems with manoeuvring in small areas. The restriction in cost is due to the budget. To produce six robots able to perform equally or better than one larger more equipped robot also means that each robot has to cost $\frac{1}{6}$ of one larger robot in order for it to be productive. Herein these constraints are more constraints, and below is a list describing each of the constraints on the robots:

- **Cost** The main motivation of this project is to create six robots being able to do tasks together as good or better than one robot. As mentioned, this means that the cost of the robots combined must be equal to the cost of one larger, more well-equipped robot.
- **Size and weight** The robots are meant to be small. This for making them easier to carry and handle. This also makes them able to perform in various small areas, which could be a case in demonstrations. They should also be light-weight, so that they are easy to transport.
- **Power Supply** Since the robots are small in size, their power supply needs to be small in size as well. This might lead to the power supply having low supply capacity.
- **Power Consumption** Since the supply capacity is limited, the power consumption needs to be low. This means that large motors or power hungry sensors cannot be equipped on the robots.
- **Computer size** Due to the constraint in size, the robots cannot have a large computer or microcontroller on board. Because of the restriction in power consumption, the computer cannot consume too much power either.
- **Memory capacity and processing power** The computer or microcontroller on each robot needs to be low-power and small in size. This often means that the memory capacity and processing power are limited as well. This encourages the usage of methods that have predictable and low resource consumption.

- **External references** Since the robots are going to be used in demonstrations, they often face various environments. This means that there may be few possibilities for deployment and set-up, and the robots should be able to work on the fly. To accommodate this the robots should not depend much on external references, if at all.
- **Sensors** Since the robots are low-cost, each sensor should be cheap. Also because of the low cost and small size there cannot be many sensors on each robot.

The list above will be frequently referenced in the later chapters when arguing for or against a component or method. It is therefore encouraged to take a note of this list.

2.2 Proposed solution

The project started in fall 2012 with a pre-study of the plausibility of a multi-robot system with the given restrictions. The conclusion was that it was possible to implement such a system and a proposed solution was given. Figure 2.1 shows the entirety of the proposed solution.

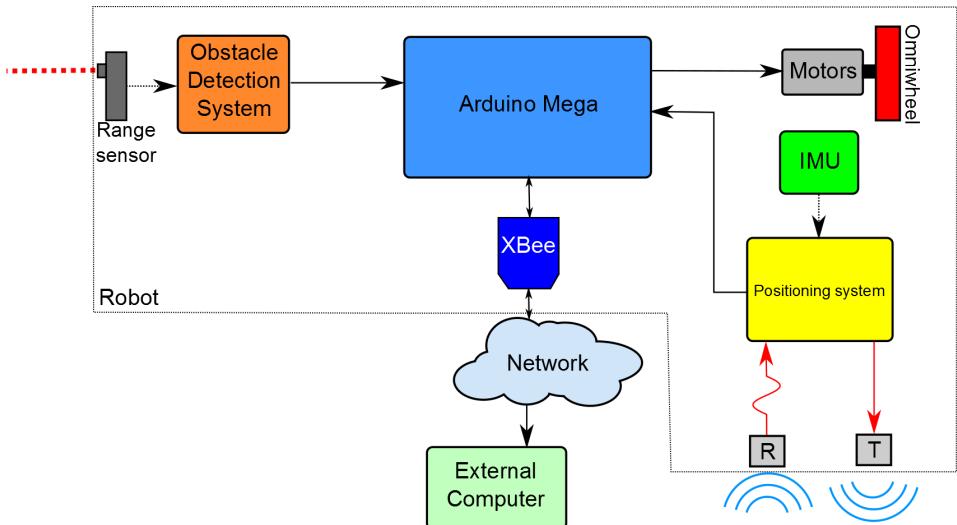


Figure 2.1: An overview of the proposed solution.

The proposed components and methods in Figure 2.1 were all tested, and they are

more thoroughly described in the later chapters. The decision of why they were used or not in the implemented solution will also be described later.

2.2.1 Movement

For simplicity the robots move on the ground, and will move on a plane floor without noticeable inclination or terrain. Having the robots move in terrain, on water or in the air would create an added complexity that would steer the focus away from the main project. To simplify the movement of the robots even further, the robots were also made holonomic. This is done by the use of omniwheels. A simple diagram of the robot can be viewed in Figure 2.2. The figure shows the hexagonal shape of the robot, and the location of the omniwheels. Using this shape and the properties of the omniwheels, the robot is able to move in the plane without any non-holonomic constraints. The mechanics of the movement can be read in Section 3.2.1.

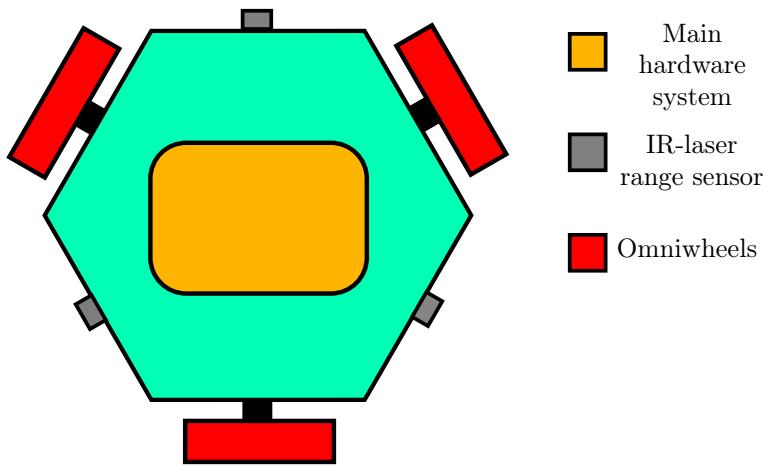


Figure 2.2: The robot viewed from the top displaying the location of the main components

2.2.2 Computational unit

As seen in Figure 2.1 the proposed system uses an Arduino Mega. This meant that the constraints in memory consumption and processing power were very dominating. This also meant that the whole system had to be small and that it would be harder to divide the system into subsystems. This choice of computational unit influenced the other choices of hardware.

2.2.3 Communication

It is required of the robots to communicate with each other and an external computer. The communication is done by broadcasting information wirelessly. The ideal solution uses a fully routed mesh network topology for this (not fully connected). This will make the robots able to communicate with everybody in the network without having to connect to everybody and without having a master-slave network. This enhances the scalability of the system. In this thesis, the term mesh network will mainly focus on fully routed mesh network.

The robots would therefore communicate with each other in a mesh network using an XBee module. This is due to the Arduino Mega does not have any good wireless communication interface other than with the XBee module. There are many advantages in using XBee, which can be read in Section 3.3.1.

2.2.4 Obstacle detection and positioning system

The robots require many sensors to manoeuvre around the environment. It is desirable to use low-cost sensors, and a minimum number of sensors. This to comply to restrictions in Section 2.1.1. Due to this, the obstacle detection method uses only an IR-laser range sensor, while the positioning system uses ultrasonic sensors together with an IMU.

To the authors knowledge, there are not many obstacle detection and mapping methods which have low resource consumption. Due to the constraints of this project, the obstacle detection system is within a grey zone of this research field, and alternative solutions had to be developed. A new obstacle detection and mapping method was developed for the obstacle detection system, called the *Inverted Particle Filter*. This method is especially designed for systems with limited resource consumption, which is a case with the Arduino Mega. The full description of this method can be read in Chapter 5.

The positioning system is described in Chapter 6 and uses ultrasonic sensors to measure the position. The methods used for measuring the position of each robot were never tested, but previous implementations of the methods conclude that the system is possible to implement[Rivard et al., 2008].

2.3 Implemented solution

After the pre-study was conducted, the implementation of the robots started. The resulting solution can be shown in Figure 2.3.

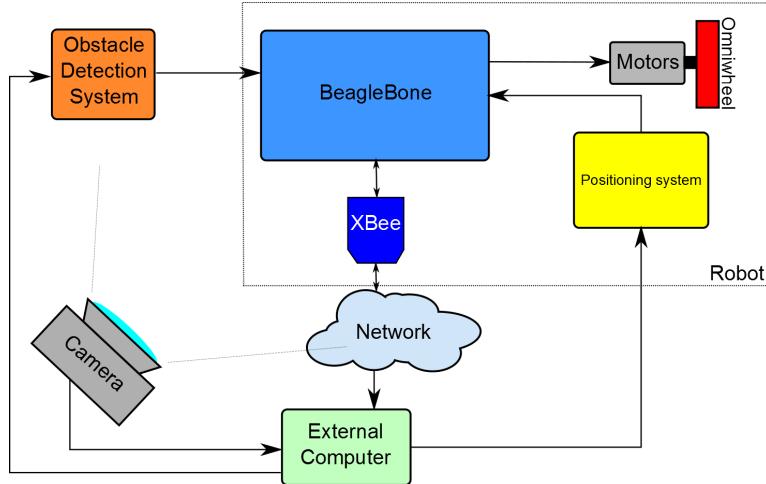
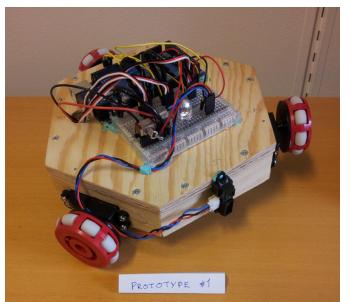
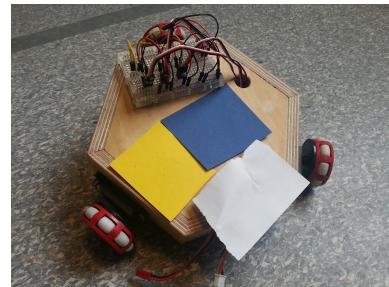


Figure 2.3: The implemented solution



(a) The first prototype of the robot



(b) The final implementation of the robot

Figure 2.4: Images of the first prototype and implementation of a robot

As the project progressed, many of the conclusions and assumptions stated in the pre-study were proven false. There were also complications in the development and construction which delayed the implementation. This meant that some of the aspects in the solution were altered or removed.

The first alteration to note is that the Arduino Mega was replaced by the Beagle-Bone. There are many reasons for this, and they can be read in Section 3.1.2. This means that the restrictions in memory capacity and processing power were much less dominating and that each subsystem could be divided into different applications. Unfortunately, this lead to the most dominating restriction was the power consumption.

There was a limitation in the broadcast function on the XBee module, making it unable to satisfy the required data throughput. This is more thoroughly described in Section 3.3.1. To fix this issue, a custom broadcasting function had to be developed, which lead to the network topology being a fully connected mesh network.

The obstacle detection system was not implemented on the robots. The IR-laser range sensor had a delay, making it unable to keep up with the movement of the robot. The experiments conducted showed that the implemented system worked, but had to run slowly in order for the sensor to be able to work properly. The limitations of the IR-laser range sensor can be read in Section 3.4.1, while the conducted experiments can be read in Chapter 5.

The positioning system also had to be switched out. There were problems when developing the hardware circuits used by the system (Section 6.6), which delayed the implementation of the rest of the system. It was at one point concluded that there was not enough time to finish implementation of the system.

Since the implementation of the positioning system was not going to be finished in time, the robots had to use an alternative positioning system. It was therefore concluded to develop an external camera vision system, identifying the different robots using colour. The full description of the visual system can be read in Section 6.6.2. Since the obstacle detection system was not implemented, the obstacle detection system was also replaced by the camera vision system.

2.4 Future work

Due to the problems that were encountered, the implementation of the robots is incomplete. The robots are able to perform in demonstrations, but not to extent desired. The robots show the concept of cooperative multi-robot systems, which was the main goal. In order for the robots to be complete there are only minor issues that has to be resolved.

If the IR-laser range sensor is replaced by a more precise measuring sensor without delay, the obstacle detection system only have some minor issues before it is fully functional. When it is replaced, the system still needs to be tested thoroughly before the conclusion about the suitability of the system can be established.

Most of the hardware circuits for the positioning system are developed, meaning only the implementation of the system is left. When the implementation is done, the system has to be tested. Since the camera vision system is fully functional, it will be easy to test the precision of the implementation, by comparing the results of the two systems.

Chapter 3

Hardware Components

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

In this chapter each of the main components of the hardware platform are presented. Each section features an aspect of the hardware platform and the components used within it. Some sections presents alternatives to the chosen components, and discusses the advantages and disadvantages of these.

The chapter starts by introducing the computational unit used in the robots. The chapter then proceeds with the components required for the movement of the robots, and continues with sections describing the hardware components for communication, obstacle detection and position estimation.

3.1 Computational Unit

The most crucial decision of hardware components is which computational unit to use. The computational unit is the heart and brain of a robot, and making the wrong decision can potentially cause the rest of the components to fail. Because of this, it was early established some requirements which the computational unit had to meet:

- **Speed** For the cooperative behavioural control system to work efficiently and for all the sensor data to be processed in time, the processor has to be fast

enough. Since neither the sensor nor system requirements are known to the full extent yet, the requirements of this property is hard to estimate.

- **Resources** The cooperative behavioural control system might require many resources, like memory. The system should be designed to the extent needed, so that it does not collect and store new data without replacing old data. This would make the resource requirements more predictable.
- **Programmability** The computational unit should be easy to program. The cooperative behavioural control system is most likely going to be implemented in SimuLink, using SimuLink Coder to generate the code. The computational unit should be easy to connect to the computer, download this code and run it.
- **Single board** The computational unit should either be a single board computer or microcontroller. This because it is not enough time to create a circuit board for the computational unit from scratch. In these days, single board computers and microcontrollers comes in many forms and qualities and are often very cheap.
- **Prototyping** It should be easy to prototype with the computational unit. If the computational unit is on a single board, with the I/O pins laid out, it will be easy to prototype with it, e.g. adding and removing sensors.

Since there is a aim to have a platform for prototyping with an I/O interface, the search has been aimed on single board computers and single board microcontrollers.

3.1.1 Candidates

There were three other candidates that were considered before the final decision landed on the Beaglebone. The three candidates were the Atmel UC3-A3 Xplained, the Arduino Mega and the Raspberry Pi. All of the candidates were tested both by developing simple applications on them, and testing how the interface towards other hardware components were. Below will be a brief description of each of them and why they did not fit the requirements.

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 pins 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 is not sufficient for this project, and may cause a problem

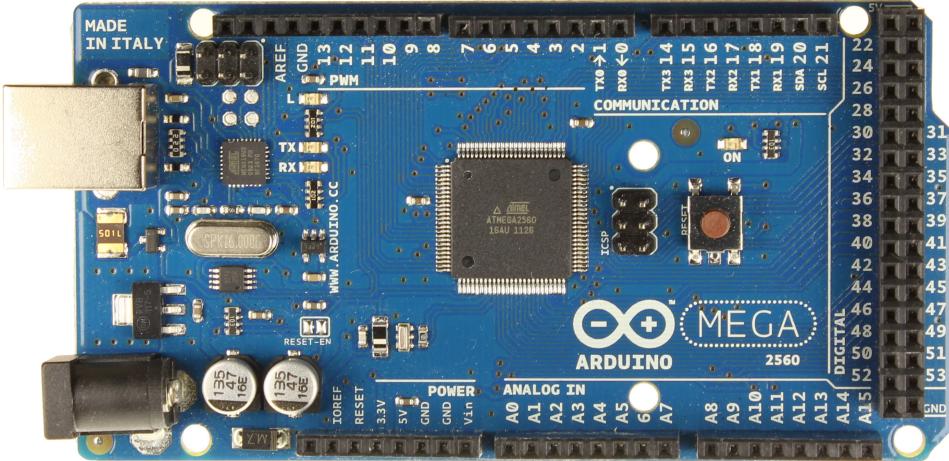


Figure 3.1: The Arduino Mega 2560 viewed from the front. Image provided by arduino.cc

in the end. 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++ code and libraries on the Arduino, making it possible to implement the required functions in C++ libraries. This was very beneficial, because then you could create simulators in either C++ or Simulink and test implementations. The implementations could then be almost directly ported to the Arduino. The Arduino Mega has an inbuilt bootloader, meaning it does not need any external programmer for it to be programmed, but also which means that it doesn't have a debug function. Arduino is designed for hobby projects and prototyping, and is not well suited for a larger systems or systems with real-time requirements. This was a problem if the full system were to be implemented. The Arduino was the best alternative next to the Beaglebone.

Atmel UC3-A3 Xplained

The UC3-A3 Xplained is a single board microcontroller 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 runs at 12 MHz, but can run at up to 66 MHz. The board only has 32 I/O pins, 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, but which means that programming all the robots

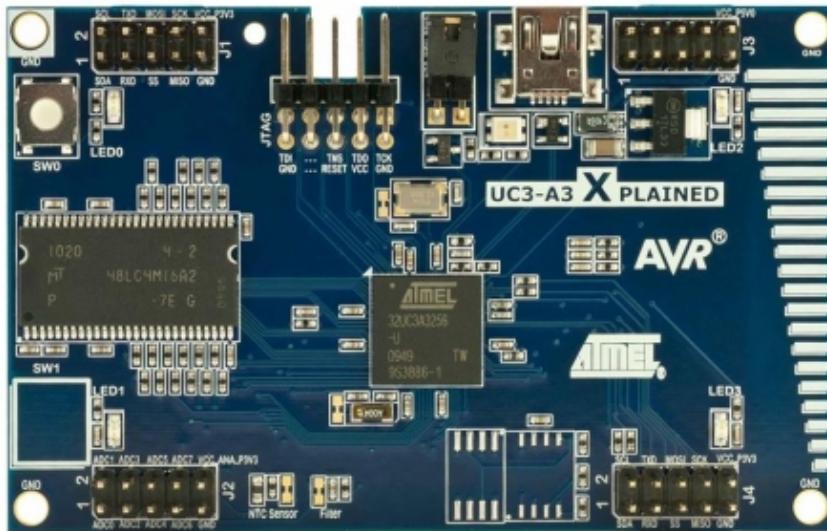


Figure 3.2: The UC3-A3 Xplained viewed from the front. Image provided by atmel.com

requires lot of time, since each has to be plugged into the external programmer when programmed. It is a 32-bit microcontroller, and is both faster and bigger with more memory than the Arduino Mega. The microcontroller is programmed in C, and the most necessary external libraries available are, provided by Atmel. Since there are less libraries available for this than the other candidates 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. Though the UC3 was more powerful than the Arduino Mega, it had an inconvenient interface which did not meet the requirements.

Raspberry Pi

The last alternative was the Raspberry PI. It is very fast and powerful having an ARM1176JZF-S with a clock speed of 700 MHz . It can also run different Linux distributions, which makes it possible to implement the system in many smaller applications, with a proper task manager, instead of one large program. It has an inbuilt GPU to do image processing, which could make camera vision an option for detecting objects. The robots, however, require many analogue and PWM signals, and the Raspberry Pi does not meet these requirements. The Raspberry Pi is also very picky in power supply. This meant that a voltage drop from the supply voltage

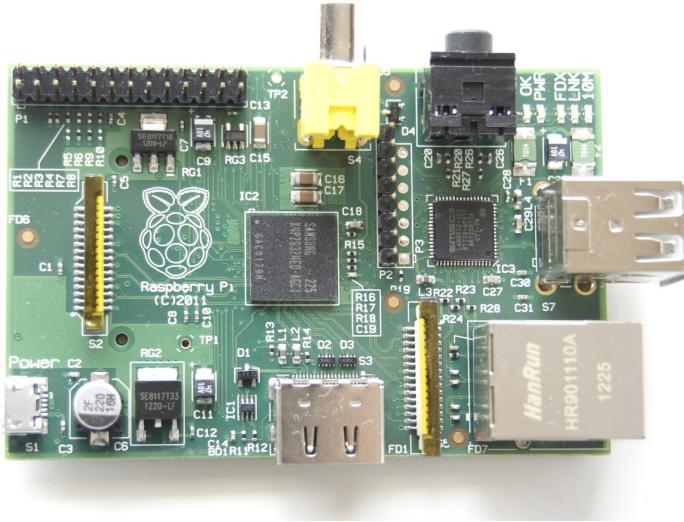


Figure 3.3: The Raspberry Pi Model B viewed from the front. Image provided by digitaldiner.blogspot.no

might cause the Raspberry Pi to write corrupted information to the memory, and in worst cases restart. This led to dropping the Raspberry PI since it was going to be powered by a battery.

3.1.2 BeagleBone

The computational unit that was settled upon is a low-cost single board computer called a BeagleBone[BeagleBoard.org, 2011]. The BeagleBone is equipped with a 720 MHz ARM Cortex-A8 processor and can run linux-based distributions. It has a large variety of I/O interfaces, which allows for the use of the required hardware components. This gives the BeagleBone the advantage of both being able to connect to the hardware components used by the robot, and providing an operating system. The operation system chosen was the Angstrom linux-distribution. This is a linux-based operating system designed especially for embedded systems, and can be installed on a wide array of computers, including both the Raspberry Pi and the BeagleBone.

One of the advantages of the BeagleBone is that it is possible to cross-compile code to it. This means that it is possible to produce code on an external computer, run tests and simulations on it before it can be compiled for the BeagleBone.

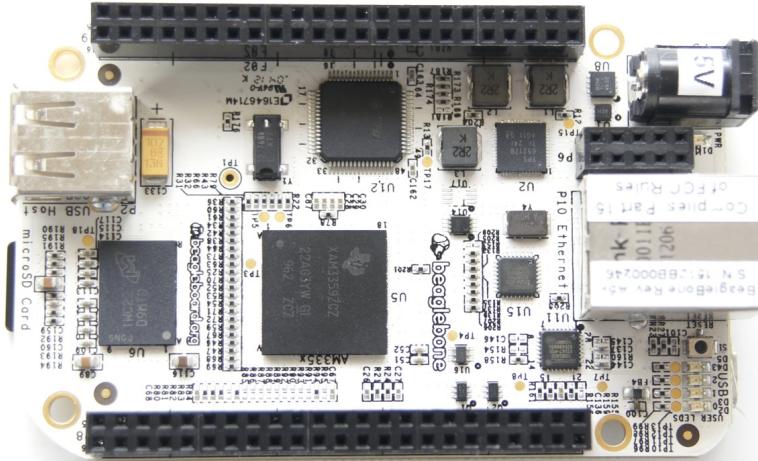


Figure 3.4: The BeagleBone viewed from the front. Image provided by digitaldiner.blogspot.no

Another advantage is that it can run Linux distributions. This means that it has features like a task manager, a file system, network sockets, among others. Therefore, each subsystem can be split into separate applications and run simultaneously. This also means that the different applications can be developed independently and that developers can work on the different applications simultaneously.

It also has the advantage of having an Ethernet port. This gives the opportunity to use SSH and other internet protocols, which makes it easy to cross-compile code and send the application to each of the BeagleBones in one click.

Since the BeagleBone runs Linux distributions and can use cross-compiled C++ code, the BeagleBone has the advantage of using many open-source third-party libraries. Libraries such as OpenCV[OpenCV, 1999] could be used on the Beagle-Bone so that it could utilize a camera for detecting obstacles.

The BeagleBone is costs only \$89 which, which is very affordable. At the time of writing, the BeagleBone has come in a new version: BeagleBone Black. The BeagleBone Black has the same interface as the BeagleBone, but with better memory and processing speed, and an HDMI port. The BeagleBone Black is also cheaper than the BeagleBone, at only \$45.

3.2 Movement

The only requirements for the movement of the robots were that they moved on a flat surface. As mentioned in Section 2.2.1 it was beneficial to make it able to move with three degrees of freedom.

3.2.1 Omniwheels



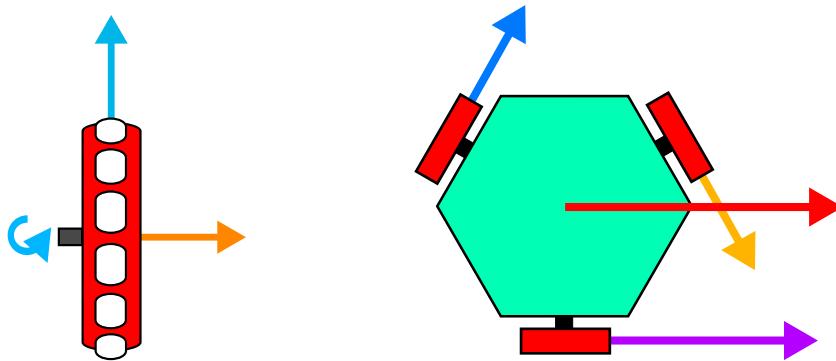
Figure 3.5: The omniwheels used by the robot. Image provided by kornylak.com

The omniwheels have a property that makes movement with three degrees of freedom possible. The wheels are designed so that they are able to roll forwards and backwards, and slide sideways. This is convenient in for instance conveyor belts when the wheels are stacked side by side forming an array of wheels. Objects can be transferred when the wheels rotate while also slide from side to side freely, making it possible to push or pull the object off the conveyor belt without damaging it.

With the wheels placed in the shape shown in Figure 2.2, the robot is able to move with three degrees of freedom. The combination of rotations on the wheels create movement in the robot, both planar movements and rotational movement. This is shown in Figure 3.6(a) and Figure 3.6(b). A complete description of the movement dynamics can be read in [Klausen, 2013].

3.3 Communication

For the robots to be able to cooperate with each other, they would require to have a communication channel. The most common form of wireless communication is by using radio waves. The most common protocols in computer networks are WiFi,



(a) The movement characteristics of the omniwheel. When the wheel turns (blue arrow) the wheel moves, it can also slide freely sideways (orange arrow)

(b) The movement of the robot. The combined rotation of the wheels creates movement on the robot

Figure 3.6: Characteristics of the omniwheels

Bluetooth and ZigBee. With the constraints in Section 2.1.1 in mind, the three protocols are discussed below.

- **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 WiFi, where one is that it is easier to connect to the internet by using a wireless router. 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. The main disadvantage with WiFi is that it is not very compatible with the BeagleBone. The best way to install WiFi on a BeagleBone is to use a WiFi dongle and plug it into the USB port. The issue is that the BeagleBone does not support many of these dongles, and even the ones supported do not work well. Another problem with WiFi dongles is that they drain much current. A BeagleBone using a WiFi dongle often drain up to 2.1 A, which would drain the battery substantially. The BeagleBone also requires a more stable power supply if the USB port is to be used. If the voltage drops below 5 V the USB port will be disabled, which leads to the WiFi connection being lost.
- **Bluetooth** Bluetooth is a wireless communication protocol for use in PAN networks. Bluetooth was first used for wireless communication between telephones and other devices (e.g. headset), 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 significantly 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 and by default does not support more than one connection for a device, which is not suited for the project. 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 which does not require USB. It is therefore needed to make a circuit which has a good interface between the BeagleBone and a Bluetooth chip, and make the Bluetooth chips able to communicate with each other. Due to time constraints, it is not possible to create such a circuit.

- **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.

3.3.1 XBee



Figure 3.7: The XBee Series 2 is a module using ZigBee as a protocol for wireless communication. Image provided by sparkfun.com

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 Antenna 2 series has been tested in this project. The chip has the full ZigBee protocol

built in, with a 3v3 UART interface, which makes it easy for the BeagleBone to interface. It is cheap and easy to get a hold of. 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 network and an external computer. The XBee has a mesh network topology, but requires a master within the system. This is not desirable, but is sufficient for the purpose of this project. A very undesirable property of the broadcast function in the mesh network is that it can only send 3 messages per 4 seconds. This was not discovered until late in the project due to the initial tests only used two modules, making broadcast obsolete. This problem was fixed by creating a broadcast function on the BeagleBone. The BeagleBone keeps a list of each XBee module in the network, and when it broadcasts it sends an individual message to each of them, receiving an ACK.

3.4 Obstacle detection

For the robots to be able to manoeuvre around the environment, the robots require sensors to detect the obstacles surrounding them. There are numerous sensors for detecting obstacles in the environment, with different advantages and disadvantages. The components used for obstacle detection were used with the following list in mind:

- **Size** The sensor should be small in size. There is limited space on the robot, meaning that in order to fit all the sensors on the robots, they have to be small in size.
- **Interface** The sensor should be easy to interface. This means that the sensors should either have a serial, I²C or an analogue interface.
- **Cost** The cost of the sensor should be low, since the motivation of the project is to have low-cost robots.
- **Accuracy** The sensor should have high accuracy. Unfortunately, high accuracy often means high cost as well. This means that precision might be compromised in favour of cost.

For detecting obstacles, one or more IR-laser range sensors are to be used. Reasons for this are described in Chapter 5. The requirements to the IR-laser range sensors additional to the ones above are:

- **Range** Range is not a prioritized requirement. It is required that the range of the sensor is far enough for the robot to respond to the sensor data. A range of 50 cm or more is sufficient.
- **Energy** The energy 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. If the beam is harmful and pointed toward an eye, no blink reflex will be triggered, and the damage will not be detected before it is too late.

3.4.1 Sharp GP2Y0A21YK IR Range Sensor



Figure 3.8: The Sharp GP2Y0A21YK IR Range Sensor is the range sensor used on the robot to detect obstacles. Image provided by sparkfun.com

An IR-laser range sensor that was settled upon was the Sharp GP2Y0A21YK. It is cheap, and comes with an inbuilt processor for signal processing and filtering, so that the only interface is a analogue signal. The sensor's output is unfortunately non-linear, but this was resolved by passing the output through a look-up table with a linear interlace. The sensor's strength is a non-harmful value. It can detect ranges up to 80 cm . Ranges farther than this will only give random high values. There exists other versions of this laser with extended range.

The disadvantage with this sensor is that it converges too slow to the right measurement. This means that if the detected distance varies too frequently then the sensor will not give accurate measurements. In the initial tests in the pre-study this was thought to be an error in the other equipment in the experiment. Since this was not detected earlier, the implementation of the method was not developed for the robots. There were though, conducted experiments of the method which supported the usage of the method.

3.5 Position estimation

In order for the robots to interact with each other and the environment, the robots required a positioning system. This system was also restricted to the constraints in Section 2.1.1, which lead to alternative sensors and methods to be used. Further description can be read in Chapter 6

3.5.1 IMU

The robots require a Kalman filter to estimate their own position. An IMU (Inertia Measurement Unit) will provide sensor data which improves the accuracy of the Kalman filter. The IMU contains a combination of a gyroscope, a magnetometer, and an accelerometer. The gyroscope and accelerometer measures the velocity and acceleration of the robot, which could be integrated into finding the current position of the robot. The magnetometer measures the orientation of the robot.

Magnetometers has a general flaw because they measure magnetic fields. A large amount of metal can shift the magnetic field around the sensor. Therefore if the robot moves within a metal container, e.g. a ship, the magnetometer might have a constant error. Since this is a multi-robot system, the orientation of each robot has to be communicated within the system. Fortunately, since all the robots would move within the same metal container, they would all have the same constant error. The robots only require the relative position of the other robots, meaning this constant error will be ignored.

Following are some of the sensors that were evaluated for improving the Kalman filter. As mentioned, the positioning system was not implemented on the robots, so none of the sensors were used.

HMC5883L

The HMC5883L is a triple-axis magnetometer that was tested in the project. It has an I²C interface, and can therefore be directly connected to the Beaglebone, and can be connected to other sensors of the IMU. It has been thoroughly tested on the robots, and proven to measuring the orientation of the robot accurately. A disadvantage of the sensors is that if the robot is picked up or moved, the sensor may need to be calibrated before it is used again. This can be fixed by running a calibration routine within the initialization sequence of the robots.



Figure 3.9: HMC5883L is a triple-axis magnetometer. Image provided by sparkfun.com

IMU Sensor

NTNU had recently developed a high precision IMU and GPS circuit. The circuit had all the sensors that was required by the multi-robot system, and had an I²C interface. The IMU would give the necessary precision for the robots. Unfortunately, the sensor could not be produced in time, and the cost of it was to high. This meant that the sensor could neither be tested nor used.

3.5.2 Ultrasonic sensors



Figure 3.10: The 400SR100 and 400ST100 are respectively a ultrasonic receiver and transmitter in the 40 MHz range. Image provided by prowave.com

To measure the position of the other robots, each robot are using one ultrasonic

transmitter and three ultrasonic receivers (see Chapter 6). The ultrasonic transmitter and receiver that were tested in this project were the 400ST100 and 400SR100 respectively. These were provided by Pro-Wave. They have a signal frequency of 40 MHz . These were mainly chosen because they were cheap. Tests using a signal generator on the transmitter and an oscilloscope on the receiver proved that a signal can be detected within approximately 3 meters. By creating a signal generator circuit for the transmitter and a signal amplifier circuit for the receiver, a signal could be sent over 2 meters. This was tested and confirmed. For more detail see Chapter 6

3.5.3 External camera



Figure 3.11: The Kinect motion sensor. Image provided by pressfire.no. Some content of this image has been removed

Since the planned positioning system was not able to be fully implemented, an external visual positioning system was implemented instead. The camera used was the camera residing in the Kinect sensor, provided by Microsoft. The camera output was processed in an external computer. A description of the implementation can be read in Section 6.6.2

Chapter 4

Software systems

Many of the aspects that feature in this project were split into separate software systems. This was made possible due to the BeagleBone being able to run many applications at once.

In this chapter the main systems used on the BeagleBone are presented, where each section features one system. The systems does not entirely reflect the applications implemented on the BeagleBone, but this is due to some of the subsystems and applications have been abstracted away.

The purpose of this chapter is to give a brief overview of the structure of the software system. This chapter is therefore deliberately short.

4.1 Overview

While the Arduino Mega would only support one single-threaded system, the BeagleBone made it possible to develop different applications serving different purposes. This required the different features of the robot software to be split into separate systems. The advantage of this is that one system can be removed or altered, without it affecting the rest of the systems, and makes the code for each application more maintainable.

The systems that were used in this project can be viewed in Figure 4.1. The arrows between the systems symbolises the relationship between the different systems, and are also the internal communication between the different applications.

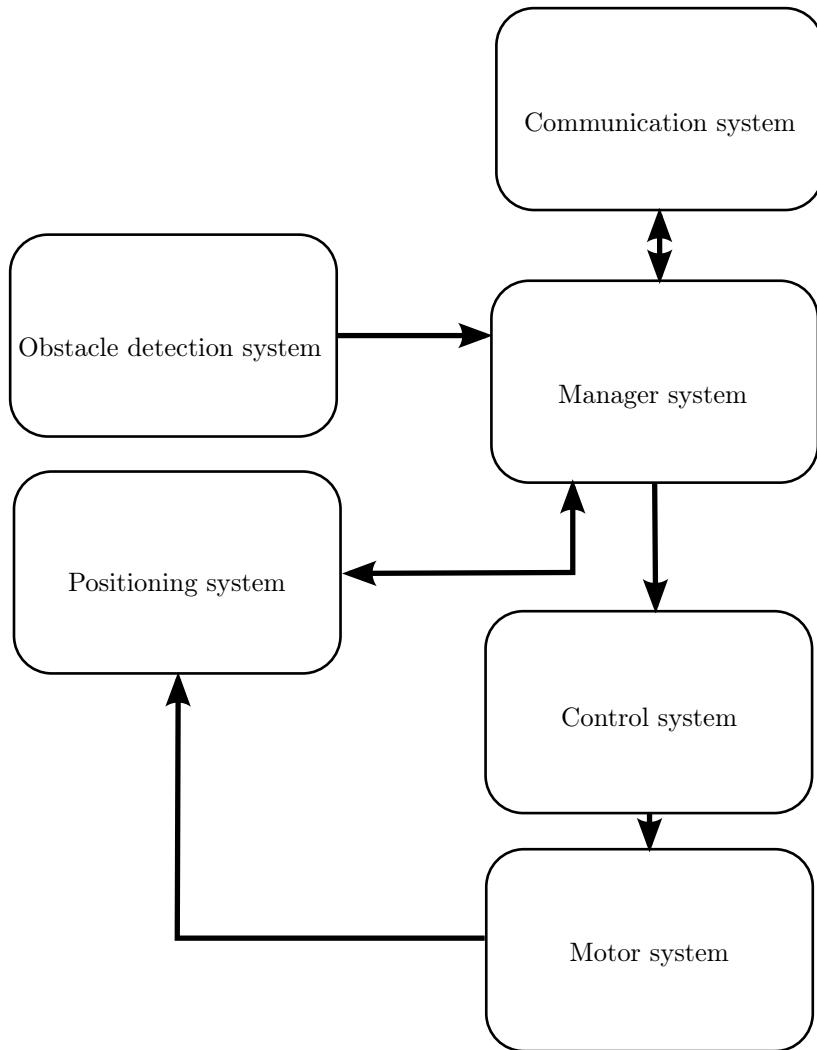


Figure 4.1: Overview of the software systems

4.2 Obstacle detection system

The obstacle detection system detects and maps the obstacles surrounding the robot. This data is then sent to the manager system for further processing.

The theory around this system is described in Chapter 5, but due to time constraints, this system was not fully implemented. The information about the surrounding obstacles is sent from the external camera vision system, which is described later.

4.3 Positioning system

The positioning system can be split into two different subsystems. The first subsystem is the relative measurement system, where the robot measures the relative positions of the other robots, and the second subsystem is the observer system, where the position of the robot is estimated based on the data from the relative measurement system and the input on the motors. The relative measurement system is described in Section 6.3, while the description of the observer system can be found in [Klausen, 2013].

As mentioned in Chapter 2, the relative measurement system was not implemented due to time constraints. This system was therefore replaced by the external camera vision system. It operated by measuring the position of all the robots, and sent the measured positions to each of the observer systems in the robots.

4.4 Motor system

The motor system is the system responsible for moving the robot. It receives the desired velocities, and sets the speed of each wheel accordingly. For further information of the internal dynamics of the movement, see [Klausen, 2013]

4.5 Communication system

The communication system is responsible for sending and receiving data from the network. The received data from the XBee is processed into JSON objects and sent to the manager system, which processes the data. Likewise, the information received from the manager system is processed from JSON objects into a more compromised form and sent to the network. The reason why the XBees do not communicate with JSON objects is that the data size is to large for it to be sent

in one message. This would lead to the robots requiring a communication protocol with message IDs, which would be time consuming to implement and give unnecessary overhead.

As mentioned in Chapter 3.3.1, the broadcast feature XBee module did not have enough data throughput to be able to meet the requirements. It was therefore decided to develop a new broadcast method. The ID of each XBee using the network is hard coded into a list within the communication system. When a message is being broadcast, the control system sends the message to each ID in the list.

The internal communication between the applications is message queues. This gave the advantage of being able to communicate without having to be dependent on each other. The messages sent through the message queues were either specific data structures if the queue only served one specific purpose, or a JSON object if the queue had a general purpose.

4.6 Control system

The control system is the system than controls what action should be taken depending on the given input. The control system is within the *reactive paradigm* of artificial intelligence[Murphy, 2000]. This means that the robots acts on the input they sense, and does not plan any further actions (Figure 4.2). For instance, the robots only move towards a target location, and avoid the obstacles that come in the way. They do not, however, plan a path towards the target where they take the obstacles into consideration. For a full description of the control system consult [Klausen, 2013]

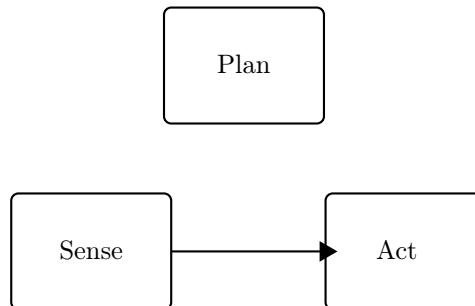


Figure 4.2: A diagram of the reactive paradigm

Since the control system is within the reactive paradigm, the objectives need to come from an external source. It is desirable to have the robots interpret a number of different type of objectives, e.g. search & rescue, transportation and mapping.

This is achieved by combining smaller tasks into task sets. Examples of such tasks are avoid obstacle A, avoid obstacle B, hold formation or move to target. If a new task set is sent to the robots, the robots will immediately obey the new task set.

This approach can be viewed as a "What you see is what you get" approach. There are both advantages and disadvantages to this. The advantage is that the robots will never diverge from the tasks, and will therefore never do less than what is demanded. The disadvantage is that the robots will not find an optimal way to perform the objectives, nor detect and avoid problems along the way. For instance if the target is in another room than the robots, the robots will move towards the wall between the target and the robots and stay there, unless they are specifically told that they must move out the door first.

4.7 Manager system

The manager system is the main system within each robot. It receives data from the obstacle detection, positioning and communication system, and processes this information so that it can be sent further to the control system in form of tasks and task sets. It also sends status messages to the communication system and further to the network.

The manager system works as the interface to the control system. One objective in this project was to create such an interface so that the control system can be switched out with other control systems.

Chapter 5

Obstacle detection system

A robot is always required to interact with obstacles in the environment around it, either if it is avoiding an obstacle or performing operations on it. It is therefore vital for a robot to have an obstacle detection system.

In this chapter the previous works on obstacle detection systems will be described, and why these are not suitable. This will be followed by a conference paper on the Inverted Particle Filter, which is a proposed method for mapping and detecting obstacles with resource constraints.

5.1 Previous work

The world is always changing and cannot always be predicted. This is why obstacle detection is required when moving through the world. There are many problems associated with detecting obstacles. Obstacles can either move or be static, they can be different sizes and shapes, and the shape and size may even vary in time. These are some of the properties which has to be detected by an obstacle detection system.

There are many ways of detecting obstacles. One could use a range finder to detect how close an obstacle is, or use a stereo vision system which extracts the features of each obstacle and stores it so the data can be reused. The choice of sensors and methods depend on the requirements and constraints of the system.

Since there are many approaches to detecting obstacles, there is also a wide variety in how complex the system can be, where complexity can be defined in how many sensors and resources the system uses. The system could for instance, detect if there

is something a given distance in front of the robot. This would not be very complex, but it would also harder to manoeuvre around the obstacles in the environment. Another extreme is if the system detected all the obstacles around the robot and placed all of them in a map, creating a full overview of the environment. This would make it easier for the robot to manoeuvre around the environment, but would require both many sensors and resources. It is therefore crucial to choose a system which suits both the requirements and limitations of the robot.

In this project there is not much room for complexity. The limited processing power and memory capacity limits the number of methods that could be implemented on the system. Below are the main requirements of the obstacle detection system:

- **Small map** The main objective of the robots is not to construct a full map of the environments. The only requirement is to have a map which is large enough for the robots to be able to avoid an obstacle, and share the position of the obstacles so they do not have to be detected again by other robots.
- **Predictable resource consumption** The resource consumption of the robots in form of memory and storage is limited. The method must therefore have a predictable resource consumption and preferably for it to be constant. Another preference is that the resource consumption can be tuned to fit the limitations of the system.
- **Low-power, low-cost sensors** The main motivation of this project is to create a low-power, low-cost multi robot system. The obstacle detection system should therefore reflect this motivation. Sensors requiring large amounts of energy, or which cost much, are not well suited for the system. It is also preferable to use few sensors. The more sensors that are required, the more power is required.

The obstacle detection system should have a map which has a constant size. This would make the resource consumption more predictable. The map should also have some form of resolution, making it easier to tune the resource consumption based on the resolution of the map. Range finders using laser or ultrasound are better suited for the requirements above. Using either a LIDAR or stereo vision could cost too much, be too power consuming and/or require too much computational power.

There are many methods for detecting obstacles. Many with different advantages and disadvantages. The focus of these methods is often on constructing maps or full instances of obstacles, making them very resource consuming. To the authors knowledge, there are not many methods for detecting obstacles, which can be shared by other robots, and have a low resource consumption. It seems like this project is in a grey zone of this research field, where alternative solutions are required.

5.1.1 SLAM

SLAM, short for Simultaneous Localization And Mapping, is a fairly new approach for detecting obstacles. SLAM methods try to solve two problems at once: Where is the robot, and what does the environment look like? SLAM uses sensor data to construct a map of the environment and to estimate the position of the robot within the map simultaneously.

The sensors used in SLAM are normally movement detection sensors, e.g. GPS, IMU or odometrics, and feature detection sensors e.g. camera vision or LIDAR [Campbell and Wynne, 2011].

The movement detection sensors measure the movement of the robot. This data is then processed in either a Kalman filter or a particle filter to estimate the new position of the robot. An example of this is a robot using a rotary encoder to measure the distance travelled, and a magnetometer to measure the angle of the robot. By combining these, the trajectory of the robot's movements can be estimated.

The feature detection sensors does not directly measure features, they rather take measurements that can be processed into features, using feature extraction methods. An example of this is to use camera vision and a SIFT (Scale-invariant feature transform)[Lowe, 2004] algorithm to detect features in individual images taken from the camera. By using two cameras the features from each camera can be compared and the distance to each of the detected features can be estimated.

The main processes within the SLAM methods are sense and move. Move increases the uncertainty of both the position of the robot and the detected obstacles, while sense decreases it. When the measurements from the movement sensors are consistent with the measurements from the feature detection sensors, then the SLAM method gets more confident in the estimated movement and position of the obstacles. After the distance to each feature and the position of the robot is estimated, a SLAM method can use this data to construct a map of the environment and place the robot within the map.

The main problem with all SLAM methods is *loop closure*. This occurs when the robot moves to a location on the map which is already detected. As the robot progresses through an unknown environment, the error of the estimated data accumulates, making the robot less and less certain if the estimates are correct. The accumulated error creates an offset from the actual position of the robot. When the robot comes to a place it has already been, the measured data and the constructed map does not coincide, because of this offset. When this occurs, all the previous data has to be adjusted to align the two points together. It requires intense computational power to both detect and close these loops, when this is required at runtime. The map loops are not only a problem. It is also useful for the accuracy of the map. When a loop closure is resolved all the previous data

about the map has become more accurate and therefore more reliable.

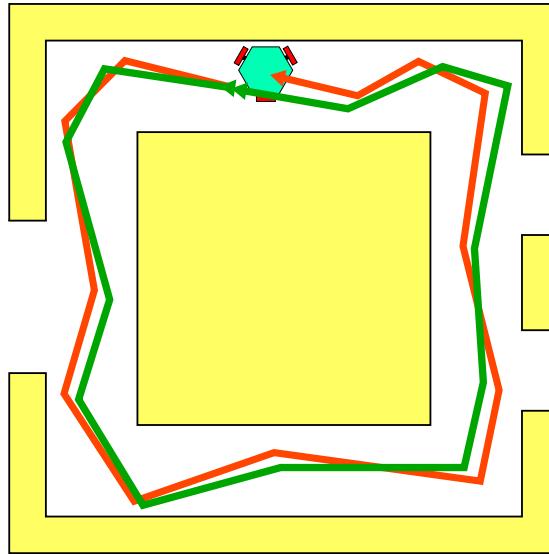


Figure 5.1: Example of loop closure. The orange arrow shows the actual path of the robot, while the green is the path measured by the robot. When the robot measures that it is back to its start location, it is not actually in the start position.

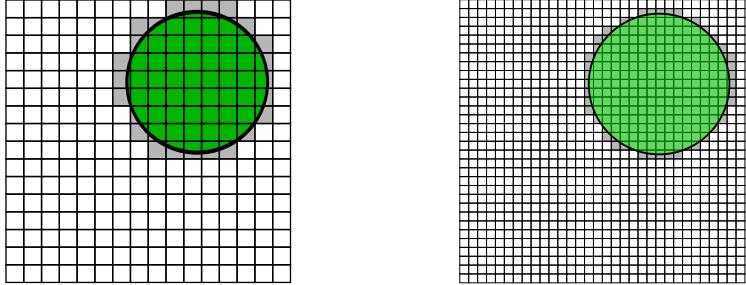
SLAM is very suitable for exploring and mapping an area, where constructing maps is the main task of the robot. Three SLAM methods are described below. The MonoSLAM method has recently been developed, while the other methods are more known and more frequently used. Note that in these methods, that the term pose is often used. A pose is defined as the position and orientation of an agent/robot at a given point in time.

GridSLAM

In GridSLAM the map is set up in a grid, two dimensional array, where each cell has a value within a unit interval, telling the likelihood of an obstacle within the cell[Chae and Yu, 2010]. The unit interval is a value between 0 and 1. If the cell has a value of 0 then the position of the cell does not contain any obstacles, while a value of 1 means the cell contains an obstacle. The grid map can be thought of as an image, where each cell is a pixel with a value ranging from black to white.

The detail of the map is dependent of the resolution. The bigger the resolution is, the more details the map has, but the more data is required to store the map.

The less resolution the map has, the less data is required to store the map, but the more discretization error occurs (Figure 5.2). Therefore, systems that has limited storage or memory must have reduced resolution.



(a) A low resolution map. The map is a grid of 16x16
 (b) A high resolution map. The map is a grid of 32x32

Figure 5.2: A low and high resolution map

The grid can either use absolute position and have a dynamic size, or have a relative position and a fixed size.

If the position of the grid is absolute, then the robot moves around on the map and each obstacle has a fixed position. The problem with this grid type is that if the robot moves outside the boundaries of the map, the map has to be expanded, hence the dynamic size. This means that the size of the grid is not predictable, leading to unpredictable usage of memory and storage.

If the grid has a relative position, the grid is centred around the robot. The robot is therefore always in the centre, while the obstacles have a position relative to the robot. Since the robot is always in the centre, the robot cannot move outside the map. Therefore the grid can have a fixed size, and the memory usage is predictable. As the robot move, the grid has to move to keep the robot in the centre of the map. This is done by moving the map to accommodate the movement of the robot. The problem with this approach is that since the grid is an array of values, the array has to be shifted whenever the robot moves. If the grid has a high resolution or a large size, then the grid array is also very large. Frequently shifting a large array of values can potentially be a problem as it requires much computing power. Figure 5.3 shows this problem.

One problem with the GridSLAM method in regards to memory and storage, is that the cells contains equal amount of information. This means that a cell with a value of 0 (no obstacle) has the same data size as a cell with a value of 1 (an obstacle). This means that when stored, empty space has the same data size as an obstacle. It would be much more beneficial to not have to store and process

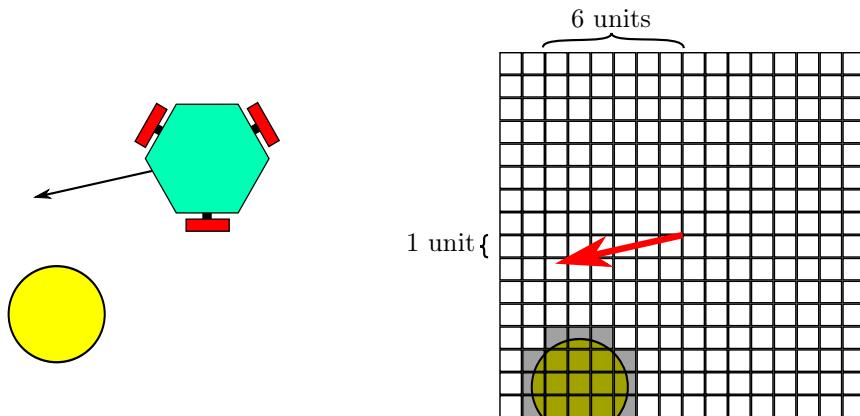


Figure 5.3: An example of a problem with shifting arrays. The robot moves, causing the array to have to shift accordingly. This example with 16x16 cells and shifting a total of 7 units results in a total of 1792 shift operations.

empty space, but rather represent empty space as nothing, and not having to store or process it.

GraphSLAM

GraphSLAM[Thrun, 2006] does not represent the map as contours, but rather have each obstacle represented as a whole in the map. GraphSLAM uses something called *relative motion constraints*, which tells the relationship between two poses, and *relative measurements constraints*, which tells the relationship between one obstacle and one pose. These constraints can be viewed as rubber bands, and when the constraints are relaxed, the resulting position of the obstacles and poses will be evened out so the tension of these rubber bands are even. GraphSLAM uses the estimated distance between each of the robot's poses and the estimated distance to an obstacle at a given pose to calculate the relative position between each pose and obstacle. This is done by applying relative motion constraints between each pose and relative measurement constraints between each pose and each obstacle. In the end the map will contain each pose the robot has had and each obstacle it has detected, and the relative position between them.

GraphSLAM uses a matrix Ω and a vector ξ to create the map. The Ω matrix is an $n \times n$ matrix and ξ is an n vector, where n is the number of poses the robot has captured plus the number of obstacles it has detected. If a new obstacle is detected or the robot moves to a new pose, n is incremented, and the new row/column represents the new obstacle/pose on the map. The element Ω_{ij} represents the

relationship between the poses/obstacles i and j , while ξ_i represents the sum of the distances between pose/obstacle i and all the other poses/obstacles.

Example

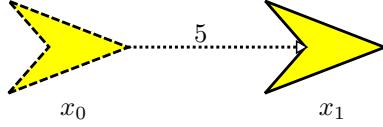


Figure 5.4: A robot moving from x_0 to x_1

A one dimensional robot starts at position x_0 and moves to position x_1 which is 5 cm away from x_0 (Figure 5.4). This means that

$$x_1 = x_0 + 5 \quad (5.1)$$

which is a relative motion constraint. The resulting Ω and ξ look like this:

$$\Omega = \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}, \xi = \begin{bmatrix} -5 \\ 5 \end{bmatrix} \quad (5.2)$$

By using Equation 5.2 and applying it to

$$\Omega x = \xi \quad (5.3)$$

you end up with equation Equation 5.1. By further moving the robot from x_1 to x_2 which is 3 cm in measured distance, then $x_2 = x_1 + 3$ (Figure 5.5), and the resulting Ω and ξ are:

$$\Omega = \begin{bmatrix} 1 & -1 & 0 \\ -1 & 1 & -1 \\ 0 & -1 & 1 \end{bmatrix}, \xi = \begin{bmatrix} -5 \\ 2 \\ 3 \end{bmatrix} \quad (5.4)$$

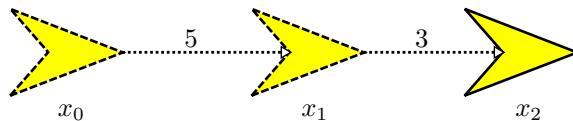


Figure 5.5: A robot moving from x_1 to x_2

If an obstacle was detected in x_0 , and again in x_1 and x_2 with the distances 10, 5 and 1 respectively (Figure 5.6), the resulting Ω and ξ are

$$\Omega = \begin{bmatrix} 1 & -1 & 0 & -1 \\ -1 & 1 & -1 & -1 \\ 0 & -1 & 1 & -1 \\ -1 & -1 & -1 & 1 \end{bmatrix}, \xi = \begin{bmatrix} -15 \\ -3 \\ 2 \\ 16 \end{bmatrix} \quad (5.5)$$

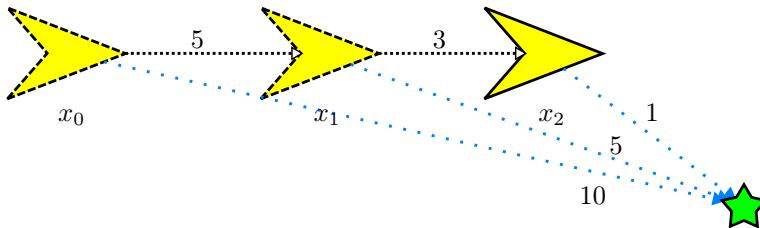


Figure 5.6: An obstacle being detected at each of the three poses. Note that in this example the obstacles and poses is on an one-dimensional line, while the image is portrayed in two dimensions to enhance clarity

The Ω matrix and ξ vector contains all the constraints for each pose and obstacle and the number of constraints increases as the robot explores new obstacles and measures new poses, and will gradually construct the map and the relationship between each object.

For the map to be fully constructed each of the constraints needs to be relaxed. The measured movements and obstacles contain error, and this can be modelled by a Gaussian distribution. By applying these distributions to all the constraints, the constraints will be relaxed. This can be done by applying the equation:

$$\mu = \Omega^{-1} \xi \quad (5.6)$$

The resulting vector μ will be the estimated position of the poses and obstacles on the map. Using the example above the results are:

$$\mu = \begin{bmatrix} 0 \\ 5 \\ 3.5 \\ 9.85 \end{bmatrix} \quad (5.7)$$

Note that in the example it was said that the measured distance from x_1 to x_2 was 3 cm . The μ vector show that it is more likely that the distance was 3.5 due to

the measured distances to the obstacle. Also the position of the obstacle is 9.85 cm . This is consistent with it being measured to be 10 cm away from x_0 which is in position 0. The distance from x_1 and the obstacle was measure to be 5. Since x_1 is at position 5 the measured position of the obstacle is also 10. For x_2 however, the position of the obstacle is measured to be 9 (8 + 1). It can therefore be seen that the position of the poses and obstacles have also been altered to get the most likely position when all the measurements and poses are considered.

GraphSLAM requires a feature detection method. This means that the obstacle has to be detected and even recognized by the robot. When the obstacle is detected the distance to it has to be calculated. A robot using GraphSLAM therefore usually use stereo vision or LIDAR to detect obstacles.

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

MonoSLAM

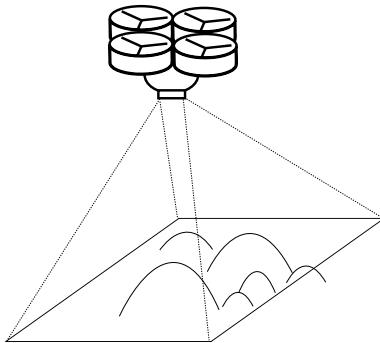


Figure 5.7: A quadcopter capturing images of a terrain. The quadcopter uses monoSLAM so each image helps create a map which also estimates the position of the quadcopter

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 captures a picture and extracts the features from it (using either SIFT, SURF, FREAK or other feature extraction 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, which is contrary to the motivation of this project. 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 surrounding them. There is no need for a complete map of the surrounding. This again leads to the conclusion that a solution like SLAM might not be necessary.

5.2 Inverted Particle Filter

To suit the requirements of the multi-robot system the Inverted Particle Filter was developed. The Inverted Particle Filter is a obstacle detection and mapping method suited for systems with limited memory capacity and processing power.

Presented below is a paper describing the Inverted Particle Filter which was submitted to SafeComp DECS'13 <http://conf.laas.fr/SAFECOMP2013/?q=node/28>:

Obstacle detection and mapping in low-cost, low-power multi-robot systems using a Inverted Particle Filter

Adam Leon Kleppe and Amund Skavhaug

Institute of Cybernetics, NTNU

Abstract. There are a vast number of methods for detecting obstacles, though not taking resource consumption into consideration. The proposed method is an obstacle detection and mapping method focused on systems with constrained memory capacity and processing power, and is called the Inverted Particle Filter. This method has small resource requirements, and can be fine tuned according to available resources. The conducted experiments show promising results.

Keywords: computational power, low-cost, low-energy, power consumption, obstacle detection, mapping

1 Introduction

The real world changes, and therefore obstacle detection is an important capability for robots manoeuvring through it. It is also a key component in creating maps of the environment, which again benefits the robots manoeuvrability. Obstacle detection has gained interest over the years and is widely used in cars, ships, planes and many other automated vehicles.

The problems in obstacle detection has been researched in many years, and there are a number of implementations and methods, each with different advantages and disadvantages. The DARPA Grand Challenge [1] invites participants to solve the general problem, and the outcome of these challenges has been used elsewhere in general. However, these solutions often focus on sensor fusion and redundancy, something which require expensive solutions with numerous sensors and computers. To the authors knowledge, a solution for low-cost, low-power applications in the challenge has not been proposed yet.

Low-power applications has gained more and more attention over the years. Applications with less power consumption, produce less heat and cost less, which are desired properties in both user interface, design and production of a device. Moore's law[2] predicts that there is exponential growth in processing power and memory capacity on a given area of integrated circuit, which again leads to more available processing power and memory capacity.

There are many benefits that come with this increased processing power and memory capacity on smaller circuits. The most common benefit is that implementations and methods can be less concerned about the resource consumption

as the circuits are getting more powerful over time. Thus, newer methods can be more complex with more focus on processing data into complete instances containing more information rather than focusing on the speed and performance of the processing, and the memory capacity required to process the data. Another way to exploit this exponential growth is that the methods that have a low resource consumption can be implemented on smaller circuits. An important benefit of this is reduced power consumption opening for new battery-powered or energy harvesting applications. To take further advantage of this it is more crucial to develop methods that require less resources such that smaller circuits in the new devices and robots, can be developed.

In the fall of 2012 a project to study the possibility of creating a low-cost, low-power multi-robot system, was commenced at the department of cybernetics at NTNU. One of the key features of this system was the obstacle detection and mapping system, which had the constraints of both low cost and low power.

In this paper, a novel particle filter method, the "Inverted Particle Filter", used for detecting obstacles and creating the map in the multi-robot system, is presented.

2 Problem definition

The main objective of the robots used in the experiment, was that they would have low-cost, low power consumption and be small in size. Another target was to make the robots independent of external reference systems.

Low cost and power consumption often requires a computer with low processing power and constrained resources. Therefore it is beneficial with an obstacle detection and mapping approach that has constant and predictable memory consumption and processing time, and that both of these are linearly dependent with the desired resolution of the map.

It is worth noting that low power consumption does not necessarily mean low processing power nor constrained resources. Nonetheless, if a method is less demanding it will in general perform better with restricted resources. Thus it is a goal to develop methods that are more resource efficient.

Since the robots were not using any external reference systems, each robot had to be equipped with a sensor to detect obstacles.

Computer vision often use either one or two cameras and a computer to process each frame captured by the camera(s). There are different algorithms to use on a frame or a series of frames to detect objects within the field of view. These algorithms often require intense computation, which is not favourable in this application.

Ultrasonic and laser range sensors are often more affordable, and require only an analogue reading to measure the distance to an object. Laser sensors are more accurate than ultrasonic sensors, since ultrasonic waves spread out and create additional echoes. The drawback with these sensors is that they only measure distances, and do not have a way of determining if there is an obstacle or not.

The two algorithms that are needed in this application is an obstacle detection algorithm and a mapping algorithm. These algorithms can easily be fused together, as once an obstacle is detected, only the position of the obstacle is required to add it to a map.

Mapping an obstacle has the benefits of reusability and that it can be distributed to other applications and even humans. However, using this approach neglects the possible movement of an obstacle. If an obstacle moves and this is not detected the map gets outdated. Another problem with the map is that the map needs to be stored, and the more obstacles detected, the more data needs to be stored in the map. This problem is an issue of decreasing concern, as the memory and storing capacity in microcontrollers grows, as mentioned earlier.

SLAM (Simultaneous location and mapping)[3] is a definition for methods that constructs a map and estimates the position of the robot on the map, based on sensor data. SLAM uses either stereo vision, LIDAR[4] or other sensors that can be used to detect obstacles. When an obstacle gets detected it is placed in the map. Current SLAM methods normally requires large amounts of memory and processing power, which is not well suited for restrictions in processing power and memory capacity. Therefore other methods had to be used in this application.

As time elapses the environment changes, and obstacles may have moved or changed shape, making the map irrelevant. Therefore it is not meaningful to create a map which is larger than a robot can maintain. In this aspect, maintainability depends on how fast the environment changes and how fast a robot can detect the changes.

It would therefore be beneficial to modify the mapping approach such that the robot only keeps a local map around the robot, where obstacles outside a defined boundary are removed from the map. Only the obstacles nearby needs to be avoided, and therefore information about the other obstacles are irrelevant. This also gains the benefit of having a more accurate map, since the obstacles in the map can be frequently updated. The other benefit of a map with boundaries is that the map has a constant size, meaning there is a finite and known number of obstacle. This makes the memory consumption predictable.

There are normally two main approaches of describing an obstacle in a map: The vector approach and the grid approach. Examples of methods for both of these approaches can be viewed in [3] and [5] respectively.

The vector approach tries to describe the obstacle with parameters and attributes. E.g. the obstacle is a circle with a radius of 1 meter in coordinates (1.2, 0.8). The obstacle use little resources when stored, and it is easy to update the obstacle since this only requires to change a few of the parameters. It is though harder to determine how to describe the obstacle, and what the parameters are. It requires much information and sensor data to make the parameters correct, which is a problem when having a limited number of resources. The resolution of the map is potentially infinite, but it is dependent on the error of the parameters.

The grid approach sets up a grid where each cell contains information about what is in the specific position of the cell. Each cell requires little information, usually only a boolean determining if there is something solid in that position or

not. The resolution and error of the map is dependent on the grid size. Therefore complex shapes will not be described well enough if the resolution is too low. There are two main problems regarding limited resources. Since it is beneficial to use a local map around the robot, then as the robot moves, the map changes. In the case of a grid map, each cell needs to use a shift operation to move to the new position. This requires much processing power, especially if the map has a high resolution. The second problem is that all the cells contain the same amount of information. This means that cells which do not contain an obstacle contains information about there not being an obstacle. This is redundant since only cells that contains an obstacle are needed.

A third approach of describing an obstacle in a map is by using particles. This is based on the vector approach, but with some properties similar to the grid approach. Each particle is part of a detected obstacle. As with the grid method, the map resolution is dependent on the number of particles, and does not have to cover the whole size and shape. Complex shapes can be described with a much higher degree than with the grid map since the position of a particle can be floating point. The particles does not have to be shifted, only change their position, making movement much easier. The particles are only contained within an obstacle, meaning that information about empty space does not need to be contained. Therefore, using particles has many benefits when having limited resources.

3 Particle filters

Particle filters[6] are recursive implementations of Monte Carlo-based statistical signal processing. In robotics they are commonly used to find the position of a robot based on measurements of the environment, and is an alternative to the model-based Kalman filter. There are advantages and disadvantages with both filters. One advantage for particle filters is that it is multimodal. If there is ambiguity in the measurements the filter can converge to multiple solutions. This is a required property when detecting multiple obstacles.

A short description of the particle filter can be that to create a finite set of particles, and give each particle in the filter the same movement characteristics as the robot using the filter. As the robot moves, each particle moves in the same manner, and as the robot takes measurements, the particle takes the same simulated measurements. The weight of a particle reflects its consistency of its measurement compared to the measurement of the robot. After a given period, the filter enters a resampling phase where particles with low weight are moved to the position of the particles with high weight with a random offset. This will eventually cause all particles to move to the same position, and thereby giving the most likely position of the robot. Figure 1 shows a particle filter using an IMU to measure the movement of the target. As the target moves, the particles that moves through the walls gets lower weights due to the target being highly unlikely to move through walls. The low-weighted particles are relocated into a more likely position, which converges to a final solution. Notice that there

are many solutions for the position of the target, but as time elapses there is less ambiguity in the measurements and the filter eventually converges to one solution. Here uses a full map of the environment to estimate the position of the target.

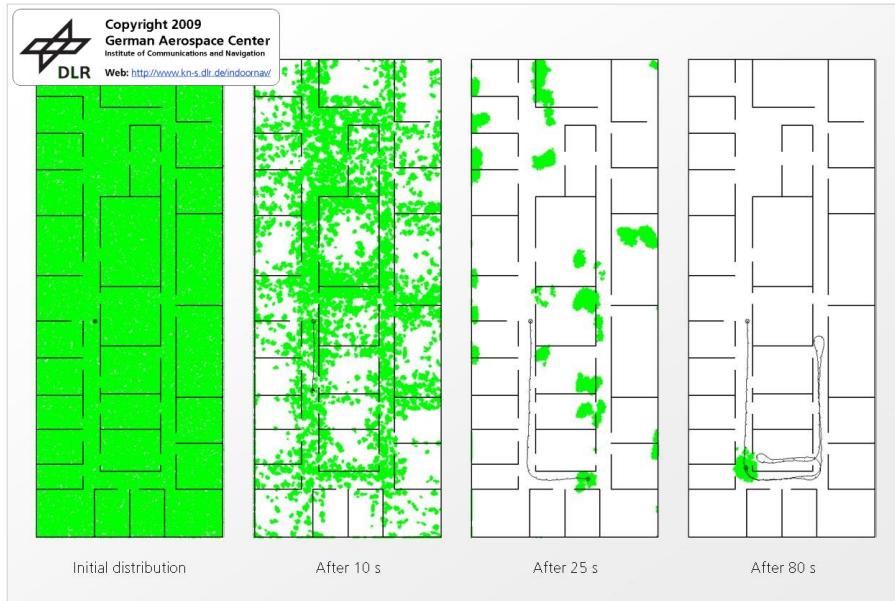


Fig. 1. A particle filter in action. Initially the particles are evenly distributed over a map, but as measurements are taken they move to a more plausible location of the target. Image provided by German Aerospace Center, Institute for Communication and Navigation[7]

The resampling phase can be designed to suit any situation. The most common approach is called the resampling wheel. This approach distributes the particles closer to the particles with high weights, increasing the number of high weighted particles. The more measurements are taken, the less ambiguous the solution gets, leading to less particles with high weights. This cycle eventually makes the particles converge to a solution.

The resampling phase can make or break the particle filter. Particle depletion is one of the major issues when resampling. Particle depletion happens when particles converge to the wrong solution. What happens is that as particles drift around according to the robots motion there may be only one or a few particles within the area with high probability of having the correct solution, and be a lot of particles in very unlikely solutions. The particles with the wrong positions is

overrepresented, and the particles with the correct answer will eventually give in and move to a different location making it very hard to converge to the correct solution unless the filter restarts.

Particle filters have some beneficial properties for low-cost, low-power systems. First of all, there are always a constant number of particles in the filter. This leads to a predictable usage of memory. Secondly, the update sequence of a particle filter is $O(n)$ which leads to a constant processing time. This means that both memory and processing time are linearly dependant on the number of particles. And thirdly, each particle can be designed to have whatever state is needed, making the particle filter a versatile approach suited for many needs.

3.1 Inverted Particle Filter

Particle filters are often used to find the position of a robot based on its measurements of the environment. The *Inverted Particle Filter* described below inverts the problem: It finds the position of the environment based on its measurements of the position.

The output from the Inverted Particle Filter is a map of particles, with the robot always in the center. The map is therefore relative to the robot. In this application the map is circular. The weight of each particle represents the likelihood of the particle being a part of an obstacle. The weights are values between 0 and 1. The weight of 0 represents the particle not being part of an obstacle, 1 represents the particle being part of an obstacle, and when the weight is 0.5 it cannot be determined. We define low-weighted, mid-weighted and high-weighted particles to have weights close to 0, 0.5 and 1 respectively.

As the robot moves, each particle on the map will move in the opposite direction. This reflects the position of the particle relative to the new position of the robot, as shown in Figure 2.

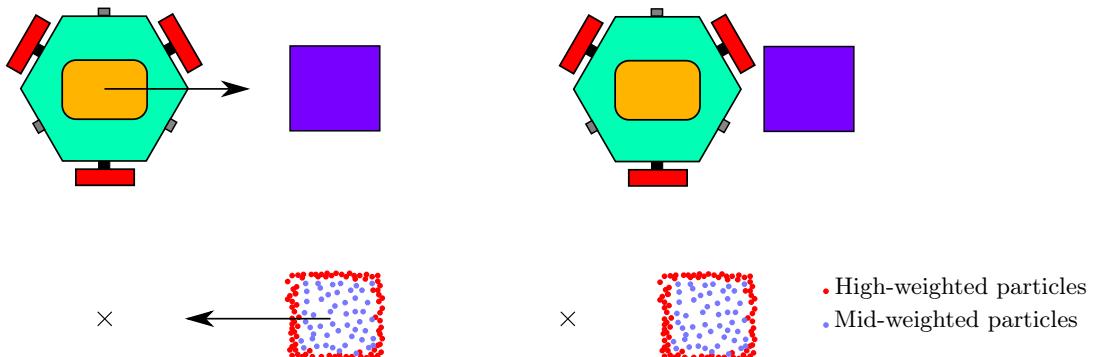


Fig. 2. A movement of the robot is represented as all the particles moving in the opposite direction in the particle filter

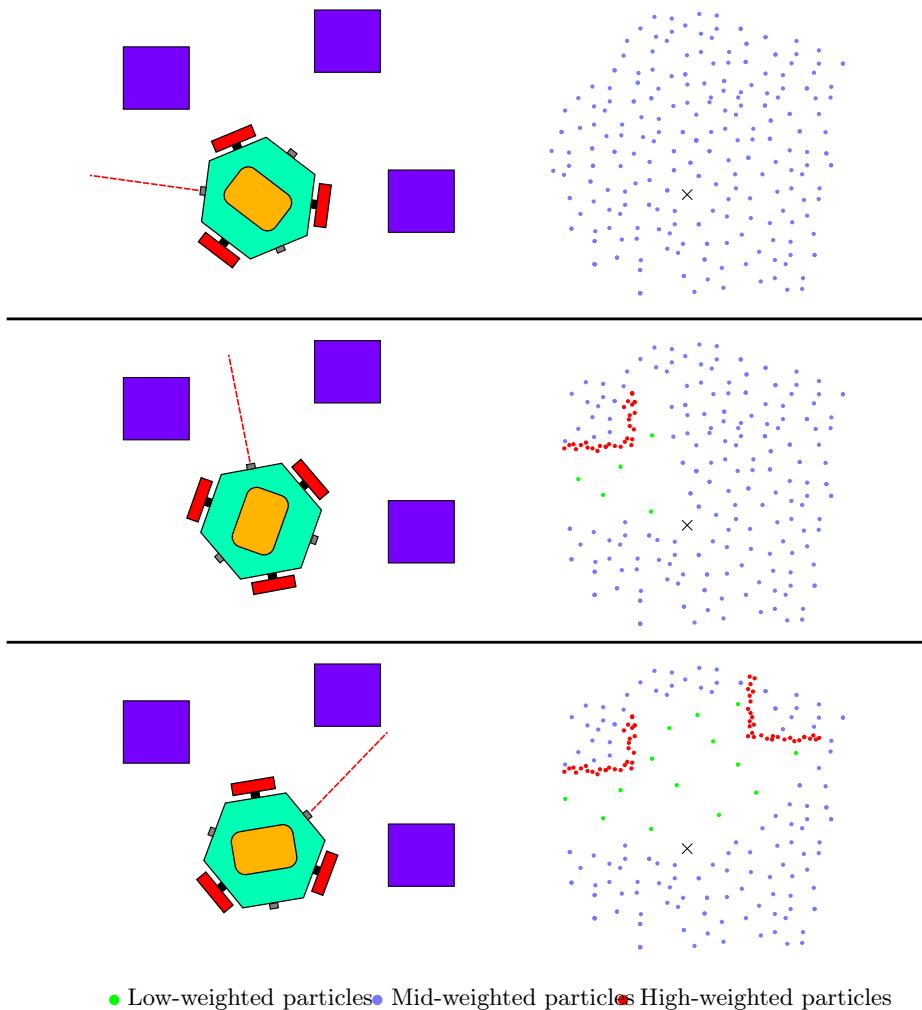


Fig. 3. On the left: A robot rotating while taking laser measurements in an environment with three obstacles. On the right: The particles in the Inverse Particle Filter as the filter is updated.

In our application a laser measurement is used to detect the distance to an obstacle, but any sensor that measures the distance to an obstacle can be used. Whenever a measurement is taken, the filter is updated. The measurement together with the orientation of the robot creates a polar coordinate. A line is then constructed from the center of the map to the measurement coordinate. The line represents, in this application, the actual laser beam. All of the particles close to the line are given a low weight, i.e. unlikely being part of an obstacle. The particles close to the measurement coordinate are given a high weight, i.e. likely being part of an obstacle. If the line runs through a particle there is nothing there, but particles where the line stops are most likely part of an obstacle. Figure 3 shows the particle filter in action. It can be seen here that at first the particles are evenly distributed, but as more measurements are taken, more of the particles move towards the position of the obstacles.

There are many benefits of using the Inverted Particle Filter. It has a predictable and constant memory usage and processing time, and depending on the number of particles in the filter. The detail of the map is also dependent on the number of particles; the more particles the more detail and obstacles can be detected. The filter is also versatile. As long as the weights are set according to the measurements, the filter will construct the map. If there are many robots using Inverted Particle Filters then the particles can be distributed among the robots. As mentioned, our application uses circular maps. This means that the robots can easily detect when the maps overlap. The particles within the overlap can be shared making obstacles detected by one robot available to another. The particles can also be sent and stored in an external computer creating a larger more connected map.

However, there are numerous of potential problems with the resampling phase in the Inverted Particle Filter. Because of this there are some guidelines that should be followed:

1. Mid-weighted particles should be untouched when resampled. This is because the particles in the mid range tells the filter that it is not known what is in the particles location. Moving such a particle will cause the filter to assume that the location is empty, potentially hiding an obstacle.
2. If the obstacles can move, there should always be particles in the empty space around the obstacle. This is to avoid particle depletion. If the obstacles move to a location where there are no particles, the obstacle will not be detected.
3. If particles move outside the boundaries of the map, then they must be relocated to another place on the map. This is because particles outside the map are unused.
4. If the robot moves, then particles with weights of 0.5 should enter from the outer boundaries, representing unknown terrain. This combined with point 3 means that particles outside the boundaries should reenter on the other side of the map. An unfortunate part of this is that there will be an uneven stream of particles reentering the filter due to the particles clustering together when they detect an obstacle. To get an even stream, it is therefore wise not only

to use the particles outside the map, but also the particles inside the map. Figure 4 shows an example of this.

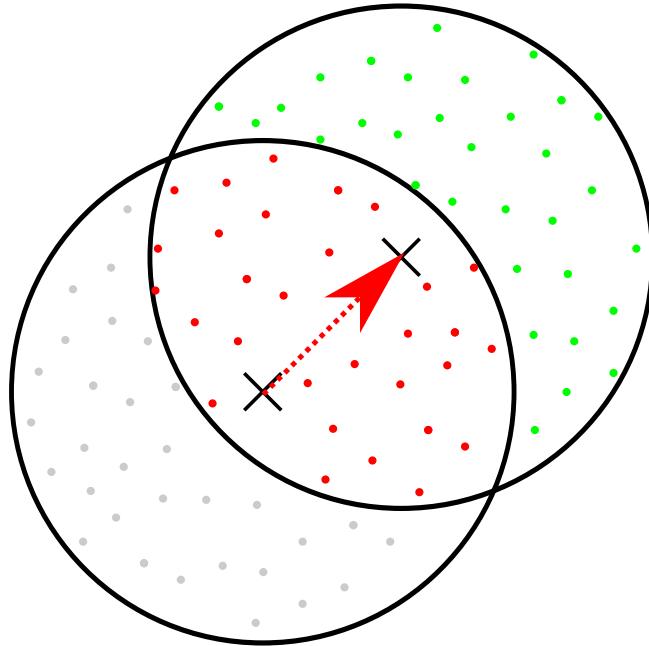


Fig. 4. The particle filter when the robot moves. The particles with grey colour are outside the map and reenteres as the particles shown in green

Using the variables in Table 1 the initialization and update algorithms for the Inverted Particle Filter can be viewed in Algorithm 1 and Algorithm 2 respectively. In Algorithm 1 it is seen that the particles are spread out evenly in a circle around origo (which is the robot). It can also be seen that the particles are rotated in the opposite direction of the robots rotation. Algorithm 2 can be split into three sections. First the weight of each particle moves toward to 0.5. This is due to the increase in uncertainty of the weight over time. After this the algorithm uses the most current measurement and constructs a line from the center of the map to the coordinate of the measurement as described earlier, and sets the weights of the particles according to their distance to the line. The last section moves the particles according to the robots movement, and if the particles move outside the boundaries of the map, the particle is removed and a new particle is created on the other side. The algorithm for the resampling phase is not presented. The resampling method used in the experiment is a resampling wheel method with a slight modification. The method is only efficient if both the robot and the obstacles are stationary, which is sufficient for this experiment.

Algorithm 1 Pseudo code for initializing the Inverted Particle Filter algorithm

Require: p, θ, ω
Ensure: $p.length <= r$
for all Particles i **do**
 // set the position and angle of the particle
 $p^i = \text{rand}(x, y)$; where $x^2 + y^2 <= r$
 $\theta^i = \phi + \pi$;
 $\omega^i = 0.5$
end for

Algorithm 2 Pseudo code for updating the particle filter the Inverted Particle Filter algorithm

Require: $p, \theta, \omega, y, m, e_y, e_m$
Line $l = \text{Line}((0, 0), y)$
for all Particles i **do**
 // reduce the information in the filter
 if $\omega^i < 0.5$ **then**
 $\omega_t^i = \omega_{t-1}^i \cdot (1 + e_\omega)$
 else
 $\omega_t^i = \omega_{t-1}^i \cdot (1 - e_\omega)$
 end if

 // update regarding measurement
 $d_y = \text{closest distance from } p^i \text{ to } y$
 $d_l = \text{closest distance from } p^i \text{ to } l$

 if $d_y \leq l_y$ **then**
 $\omega^i = 1 - \frac{d_y}{l_y} \cdot e_y$
 else if $d_l \leq l_y$ **then**
 $\omega^i = \frac{d_l}{l_y} \cdot e_y$
 end if

 // update regarding movement
 $p_t^i = p_{t-1}^i - m$
 if $\text{length}(p^i) > r$ **then**
 remove particle
 create new particle on the other side
 end if
 $\theta^i = \phi + \pi$
end for

Table 1. Variables used in the Inverted Particle Filter

p	The position of the particle
θ	The angle of the particle
ω	The weight of the particle
y	The measurement of the robot
m	The movement of the robot
ϕ	The rotation of the robot
e_y	The probability of error in the measurement
e_m	The probability of error in the movement
e_ω	The probability of error in the weight
r	The radius of the Inverted Particle Filter map
l_y	The length of the beam of the measurement

4 Experiment

The experiment was set up with a laser¹ placed upon a servo². The laser and servo were controlled by an application which ran on a Beaglebone, which is a single creditcard-sized board computer provided by Beagleboard.org/bone with a Linux operating system. The application was implemented in C++. The servo was set up to move from 0° to 90° with the step of 1° after each time the laser took a measurement. This happened at an interval of 500 ms. Each measurement outputted an analogue signal ranging from 370 to 3560 with a resolution of 16-bit. The signal was transformed in a lookup table to output the measurements in centimetres. The application treated every measurement over 80 cm as 80 cm, since these measurements were more unpredictable. Each measurement was complimented with the (known) angle of the servo, resulting in a polar coordinate relative to the laser. A various number of obstacles were placed in the field of view of the laser as shown in Figure 5(c).

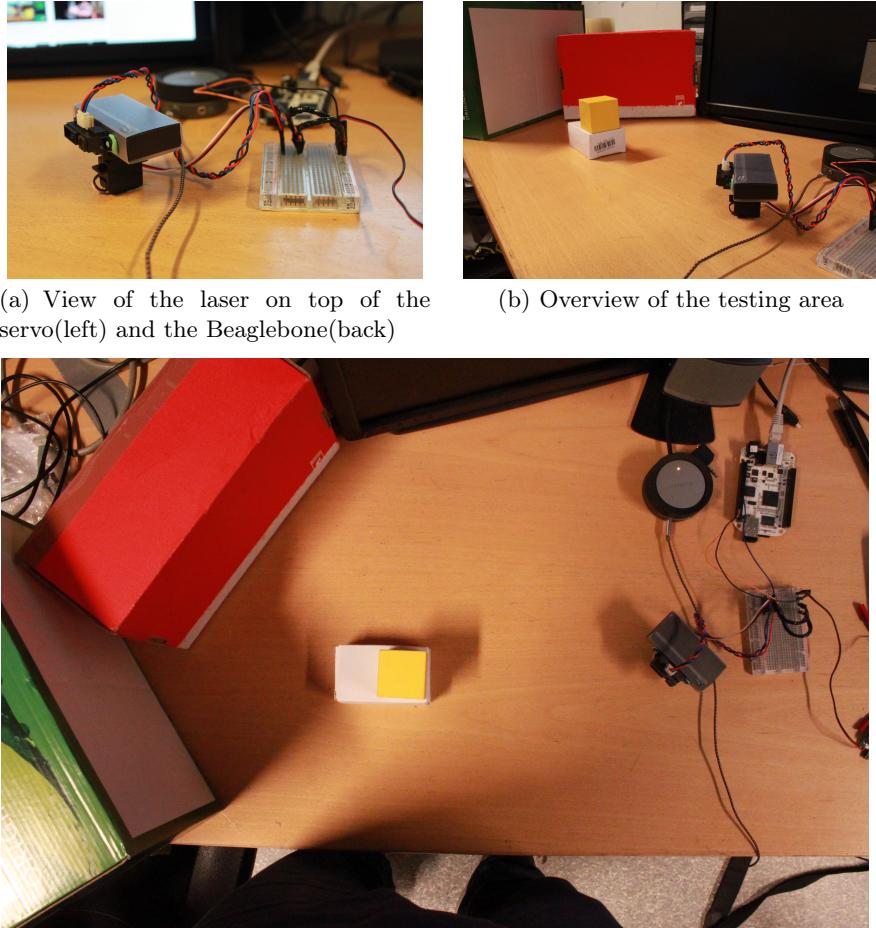
After the program started, the servo moved from 0° to 90° then back to 0° , while the laser measurements were taken. The filter was updated after each measurement. When the servo reached 0° , each particle was saved in a CSV-file and transmitted to an external computer. Images of the setup of the experiment can be viewed in Figure 5.

5 Results and discussion

In Figure 6(a) the contours in the distribution of the particles represents the obstacles in the environment. There is only one quadrant in the plot that has any valuable information, Figure 6(b) is a plot of this, the upper left quadrant. There were no measurements taken in the direction of the other three quadrants. and therefore particles within those do not contain any information

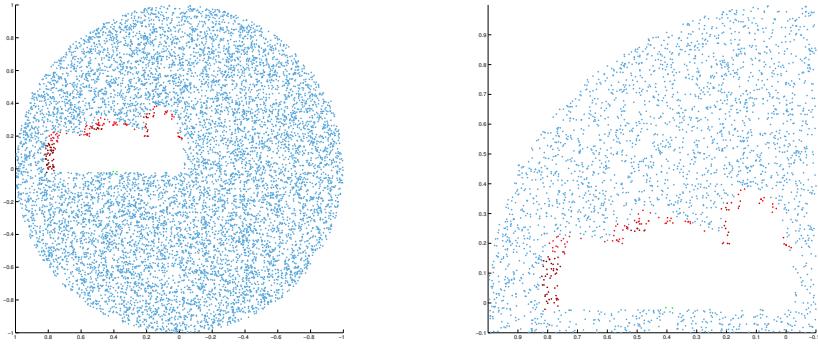
¹ The laser was a Sharp GP2Y0A21YK IR Range Sensor acquired from Robonor.no.

² The servo was a 9g Small Servo acquired from Sparkfun.com.



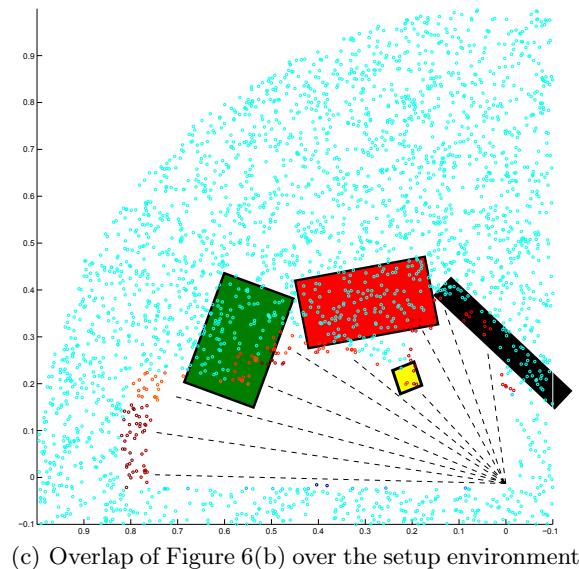
(c) Top view of the testing area. The laser can be viewed in the centre bottom of the image, while the obstacle terrain can be viewed in the top half of the image

Fig. 5. Images of the set-up and testing area



(a) Plot showing all the particles after the experiment was terminated. Red colour is a high-weighted particle, blue is a low-weighted particle and turquoise is a mid-weighted particle

(b) Zoomed in plot of Figure 6(a).



(c) Overlay of Figure 6(b) over the setup environment

Fig. 6. Plots of the particles in the Inverted Particle Filter after the experiment terminated. The colour of a particle is based on the weight ranging from green(low) to blue(mid) to red(high).

The contours of the obstacles that can be seen in Figure 5(c) are detected and plotted in Figure 6(b). An overlap of the plot in Figure 6(b) and the setup of the obstacles is found in Figure 6(c). The particles printed with red color have a high weight and represents the contours of an obstacle. Note that the particles in red with a distance of more than 0.8 m from the center represents as the maximum range of the sensor used is 80 cm.

It can be seen in Figure 6(b) that there are no particles in the area where there are no obstacles. This is, as mentioned earlier, problematic if either the robot or obstacles move, but in this experiment it makes the figure more clear.

It is worth noting that there has been conducted experiments with the robot moving while taking measurements. The experiments were successful, but the reaction time of the laser is too slow. The laser's measurements does not give the exact distance of the laser beam instantly. If the distance of the laser beam changes, the measurements gradually converges to the distance. Therefore the laser measurements cannot keep up with the robot's movements if it moves to quickly.

Each particle in the implementation only used 5 single-float values, meaning a filter of 10 000 particles only require 50 kB of memory on a 32-bit processor. An approximate measurement of the memory usage of the filter was measured to 1.5 MB on the Beaglebone. The measurement is a rough estimate since there were no simle way of measuring memory usage on the Beaglebone with our selected tools.

This experiment was also done when the Beaglebone was replaced with an Arduino Mega, a single board microcontroller using an 8-bit ATmega1280. The experiment was successful, implying that the filter in general can be implemented even on smaller microcontrollers. Note that the number of particles was reduced to 1000 particles.

An objection that might be raised here in these experiments is that the particles do not represents avoidable obstacles. These could be found by creating lines between neighbouring high-weighted particles which then again could be combined to form obstacles, or other feature extraction methods used in visual processing. In contrast to the motivation many of these methods are computationally intensive. However, if the system is too constrained to perform such a processing, the filter could easily just return the nearest particle to be avoided.

6 Conclusion

The Inverted Particle Filter is a tool for detecting and mapping obstacles, with a focus on limited resources. The main advantages of the filter is that it requires small amounts of resources both for executable code and memory while running. These resource requirements for the filter are predictable and, the filter can be tuned according to the limitations of the system.

References

1. Thrun, S., Montemerlo, M., Dahlkamp, H., Stavens, D., Aron, A., Diebel, J., Fong, P., Gale, J., Halpenny, M., Hoffmann, G., Lau, K., Oakley, C., Palatucci, M., Pratt, V., Stang, P., Strohband, S., Dupont, C., Jendrossek, L.e., Koelen, C., Markey, C., Rummel, C., Niekerk, J.V., Jensen, E., Alessandrinini, P., Bradski, G., Davies, B., Ettinger, S., Kaehler, A., Nefian, A., Mahoney, P.: Stanley : The Robot that Won the DARPA Grand Challenge. *Journal of Field Robotics* **23**(April) (2006) 661–692
2. Moore, G.E.: Cramming more components onto integrated circuits. *38*(8) (1975)
3. Thrun, S.: The Graph SLAM Algorithm with Applications to Large-Scale Mapping of Urban Structures. *The International Journal of Robotics Research* **25**(5-6) (May 2006) 403–429
4. Campbell, J., Wynne, R.: Introduction to Remote Sensing. (2011)
5. Chae, H., Yu, W.: Artificial landmark map building method based on grid SLAM in large scale indoor environment. *2010 IEEE International Conference on Systems, Man and Cybernetics* (October 2010) 4251–4256
6. Gustafsson, F., Gunnarsson, F., Bergman, N., Forssell, U., Jansson, J., Karlsson, R., Nordlund, P.j.: Particle Filters for Positioning , Navigation and Tracking. *Signal Processing, IEETTransactions* **50**(2) (2002) 1–13
7. Robertson, P.: <http://www.kn-s.dlr.de/indoornav/>

Chapter 6

Positioning system

One of the main requirements in an autonomous cooperative system is a positioning system. In order to make robots able to cooperate on different tasks, they all need to know what position their at an where to go next.

This chapter first presents the previous work done with positioning systems. The chapter then proceeds with a description of what is required by the system and why a relative positioning system was chosen, and the different methods of measuring relative positions. After this, the relative positioning system proposed for the robots is presented in detail, followed by the implementations done of this system.

6.1 Previous work

A reference system is required in order to describe the position of a robot or vehicle. The most common reference system is the world, where a position in the world is described by using latitude and longitude. All reference systems require a reference point, a point to compare the measured position against. *Absolute positioning* is when the reference point is pre-defined, known in advanced, and is the same for everybody using the reference system. Common methods for measuring absolute position is by using triangulation and trilateration methods, such as GPS. Absolute positioning is often used in ships, planes, cars and many other applications.

Relative positioning, on the other hand, does not have a pre-defined reference point. The position of an object is relative to the entity which measured the position of the object. Relative positioning systems are often used in navigation, localization and robotics among others.

Absolute positioning is normally most used in autonomous robotics and navigation, while relative positioning is more used in the industry. This due to GPS and other systems being able to give a very accurate measurement of the absolute position of a robot. By using GPS, odometry and an IMU, and use the sensor data in a Kalman filter or other types of position estimators, the absolute position of the robot or vehicle can be accurately estimated. The estimated position of the robot or vehicle can be distributed to other internal and external systems requiring this information, which would make the robots/vehicles able to cooperate on tasks like moving to a target or holding a formation. Often these robots and vehicles also are large in size and move slowly. This gives an estimator time to create accurate estimations, and the robots/vehicles time to handle this information.

When the robot/vehicle is situated indoors, the GPS will not get a good enough signal to function properly, and is therefore unable to provide reliable sensor data. This can be solved by placing reference points in the environment which the robot/vehicle can measure. If the measurements provide a distance from the reference points to the robot, then the position can be found with trilateration (see Section 6.3.1). There are many ways of implementing reference point systems. Two examples of such implementations is differential GPS[Vik, 2012] and beacon towers[Sperre et al., 2012].

Differential GPS (DGPS) is mostly used to enhance GPS accuracy and remove common errors. Relative positioning with DGPS works by having two or more GPS senders stationary on ground, on a known position, and having the robot/vehicle receive the deviation of the GPS signal from each of these. The GPS senders that are stationary get their position from the GPS system as they would normally do, while the robot receives its position relative to each sender. This is also called local GPS, or LPS.

Beacon towers use towers or reference points set up in the environment. There are two types of beacon towers: The towers that sends a signal which the robot listens to, and the towers that listens to a signal sent from the robot. An implementation of the latter approach can be read in [Sperre et al., 2012]. The system described there has two infrared lasers spinning on top of the robot. Three stationary towers are placed on known locations, which detects when a laser beam passes through. The two lasers on the robot has a fixed distance between each other and rotate with a fixed and known angular speed. As the lasers spins, a tower will detect the two laser beams with a time difference. By using this time difference and the known angular speed, the distance from the tower to the robot can be calculated. Using three or more towers and a trilateration method, will make the robot able to estimate its position.

Another common method to estimate the position of a robot is to use camera vision. Camera vision requires one or more cameras set up around the environment, preferably with a clear view of the robots. Each robot are also required to be

distinguishable from the other robots, e.g. 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 an algorithm for detecting symbols or colors. The data is then distributed to each of the robots. This method was implemented on the robot system since the proposed positioning system was not finished in time. This is further described in Section 6.6.2

The methods described above would be hard to implement on the robots with the given constraints. The GPS is not accurate enough for this application. Each robot is approximately 15 cm in diameter. The GPS has a accuracy of approximately 5 m , and would therefore not fit the requirements. The GPS is also not suited for indoor usage. Systems with approaches like the DGPS or the beacon tower would break with the constraint of having no external reference systems. As mentioned, external reference systems are a potential weakness when exploring hazardous and unknown terrain, and in our case would require too much set-up time in demonstrations. The camera vision method would require an external computer as well, and would not be very scalable. This method would also make the robots rely only on the camera vision system, which is not a desired property when it comes to cooperation. DGPS, beacon towers and camera vision would also create a boundary which the robots must be within in order to be detected.

6.2 Relative positioning

Making each robot able to detect only 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 of neighbours to deal with. This enables the possibility of creating a fully routed mesh network between the robots, with the robots able to communicate through routes in their neighbourhood. The system is therefore very scalable. The problem in these mesh networks is that information has to go through many nodes to go from one end of the network to the other, thus being very slow. This is not a concern as a robot does not need much information about all the robots in the network (only its neighbours), and it 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 partitioned for some reason, there will be elected one master for each network, and a merge between networks will need another

re-election. Every change in the network will therefore require a check of who is the master in the system. This also means that some of the slaves in the network has to be a backup if the master is lost from the network. The master is also required to communicate with the whole network, inhibiting scalability.

- **No reference points** As explained earlier, using external reference points to find the position of the robots is not favourable. A fault in a reference point, or a lack of reference points may potentially immobilize the robots. They also require deployment, which cannot be done in all environments, and restricts the area the robots can operate. When the robots are able to detect their position based on their neighbours, they will not rely on any external reference systems.
- **Environment** Being able to behave consistently in different environments is desirable. To have one robot able to handle all sorts of environments without needing to switch sensors or other hardware can be cost efficient. If the robot does not require other reference systems than itself to detect other robots it cooperates with, it can do so in many 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 each other within the swarm. It would be hard for a locust to have a master or an external reference point. The only requirement of a cricket for the swarm to work, is to measure the position of its neighbours. The Boids algorithm[Reynolds, 1987] is an algorithm able to create swarm behaviour by only using relative position and velocity. This algorithm however, 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 by these examples that measuring relative position is a desirable properties when developing robots with cooperative behaviour.

6.2.1 Measuring relative position

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

Camera vision is getting increasingly popular to use. This is both due to newer computers have increased processing power, and is therefore able to process images at greater speeds, and that using cameras can be used for obstacle detection,

mapping, as well as measuring position. Following here are some advantages and disadvantages to using camera vision:

Advantages

- **Detects more** Camera vision methods detects features. This means it can detect all the objects in the field of view. Therefore it does not only detect the neighbouring robots, but also obstacles in the environment. The opportunity to merge the positioning system and obstacle detection system would be beneficial, since the robots would only require one sensor.
- **Identification** It is easy to place symbols, lights or colours on each robot, giving them a unique identification. This can then be detected by the feature extraction algorithm, making it able to identify which robots are neighbours.

Disadvantages

- **Heavy computations** Image processing often requires much processing and memory resources to work. This is due to amount of sensor data which is required to analyse in order to get the desired information. This is not suited for this project, because of the constraints in processing and memory capacity (see Section 2.1.1). This might also make the computational unit unable to handle the other deadlines that it might have.
- **Field of view** Cameras have a restricted 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 robots, 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 a robot has had between these two measurement is unknown. Another method of fixing this problem is to use many cameras, which again increases the number of the sensors, the amount of sensor data and the necessary power consumption of the robot.
- **Learning algorithm** Image processing requires some sort of learning algorithm to recognize the extracted features. The problem with using a learning algorithm for recognizing objects is that the algorithm might not converge fast enough or in worst case, not at all. This would lead to the robot not recognizing objects properly and would be a risk to the system. It is more reliable and desirable to use methods that have consistency.

It can be seen here that camera vision is not suited for solving the relative position problem for this project. It is more used in robots or computers capable of handling computational demanding processes, and commonly used by single robots.

The less common sensor used for measuring relative positions of other robots, is by using light. Sensing distances with light implies adding two or more light sources (preferably LEDs) and photodetectors on each robot. Each robot then uses the photodetectors to detect the luminescence of nearby light sources.

Advantages

- **Continuous detection** Since light travels at the speed of light, there is no measurable delay from the light being emitted to it being detected. This means that each light source can be detected continuously without delay.
- **Detecting angles** It is easy to detect the angle of the light source by using three or more photodetectors. If the distance between the measuring robot's photodetectors are known, the direction of the light source can be calculated using trilateration.

Disadvantages

- **Identification** The only ways to distinguish one light source from another is by using different frequencies or by using modulation. 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 difficult using light. To detect the distance of an object emitting light, it requires the object to have more than one light source, and a known distance between them. When viewing two light sources on a robot from the side, the distance between the sources vary if viewed from different angles, see Figure 6.1. 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. Therefore it would be difficult to detect frequencies in the mid range. Higher frequencies like X-rays and gamma rays would be difficult to produce, and potentially dangerous. The sensors are therefore limited to only using frequencies lying in the microwave

and radio wave region, which does not have many options when it comes to photodetectors.

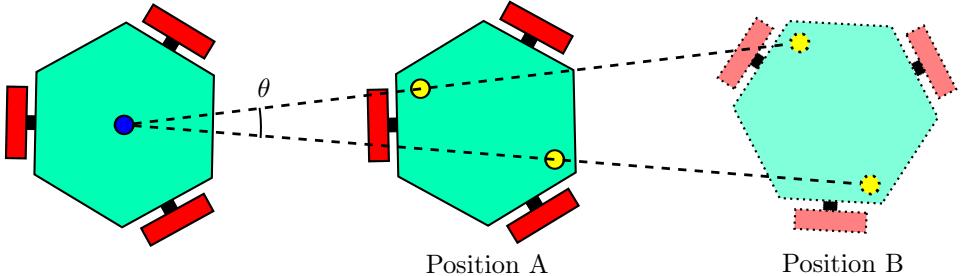


Figure 6.1: An angle θ between two light sources on robot B are measured by robot A. If the angle of robot B is not known, robot A cannot determine if robot B is in position A or B

Measuring positions using light is difficult without knowing the robots' angle, and even then it would have many issues. It can be argued that it only would require to detect the strength of the light source to detect the distance, and having many photodetectors to detect the angle. This is a possible solution, but this still has a problem with identifying each robot, and the potentially many sources of measurement noise.

Another method for measuring the relative position while using light, is to modify the beacon tower method. By making one spinning laser tower on top of each robot and three reference towers on each robot, the relative position could be calculated. This solution, however, requires many more sensors than the other methods, and would still have many unresolved issues.

Sound, on the other hand, has properties that make it easy to use for measuring relative positions. By placing one sound transmitter and three or more sound receiver, with known distances between each of them, on each robot, the relative distance and angle can be calculated. This will be explained in Section 6.3. Here are some advantages and disadvantages by using sound for detecting relative positions:

Advantages

- **Speed of sound** The speed of sound is $340.29 \frac{m}{s}$ (in air at 1 ATM). When two microphones are placed beside each other and a sound wave traverses through them, the time difference between each detection is measurable with a high enough sampling rate. Therefore by having three or more microphones

with a known distance from each other, the position can be measured using trilateration.

- **Identification** Unlike light, the energy of a sound wave does not vary much with the frequency. This means that one sound wave at 2 Hz and one at 20 kHz have approximately the same energy given the same amplitude. If each robot has a unique frequency, the robots can be distinguished from one another. Another way for identifying each robot is presented in Section 6.4.
- **Disturbance** Above 20 kHz there are not many sounds in everyday life. Sounds higher than 20 kHz are normally created by motors and electrical appliances, but with very low amplitudes. This means that there is normally not much disturbance when using high frequencies.

Disadvantages

- **Detecting distances** When detecting distances using sound, the time between emittance and detection 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.

6.3 Relative positioning system

As mentioned in Section 6.2.1, sound has properties that are favourable for the relative positioning system. The transmitters and receivers come in various shapes and sizes, can have a wide range of frequencies and can be very cheap. It is favourable to use ultrasonic sound. This both because there is not much interfering noise in the ultrasonic frequency range, and it is not heard by humans.

The relative positioning system in this project, uses one ultrasonic transmitter and three ultrasonic receivers, placed as shown in Figure 6.2. In order to measure the relative position between each robot, each robot has to produce circular sound waves which spread out on the horizontal plane, and have a way to trilaterate the signals that are produced. This can be accomplished by placing the transmitter and receivers such that they face upwards, and with a cone on top of each of them, as shown in Figure 6.3.

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 receive the sound wave, and record when each of their receivers senses the sound

wave, as shown in Figure 6.4. The distance from the emitting transmitter to each of the receivers can then be calculated using the time difference.

The sampling rate of the positioning system is important for the accuracy. Since a robot must measure the time of impact of a sonic pulse, the sampling rate will influence the time of measurement. Speed of sound is as mentioned, $340.29 \frac{m}{s}$, meaning that it moves 1 cm per $3 \cdot 10^{-5}$ s. Therefore, if the accuracy of the distance measurement is ± 1 cm the sampling rate has to be 34 kHz or greater. The BeagleBone has an inbuilt ADC converter with a 100 kHz sampling rate, which potentially gives the accuracy of ± 3 mm.

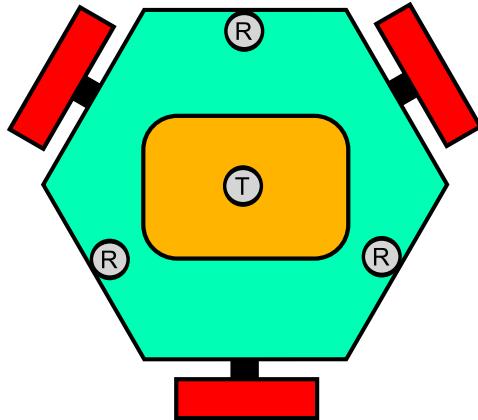
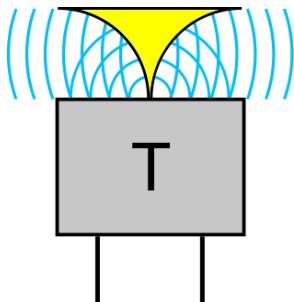
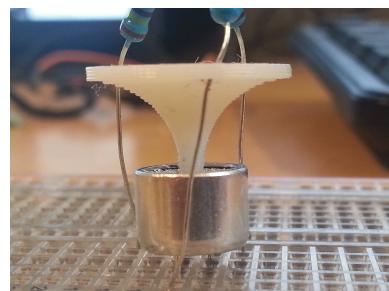


Figure 6.2: Robot viewed from above. T represents a ultrasonic transmitter, and R represents a ultrasonic receiver



(a) Transmitter viewed from the side. A sound wave emitted is spread out into a circular wave.



(b) Transmitter and cone in actual size

Figure 6.3: The set-up of the transmitter

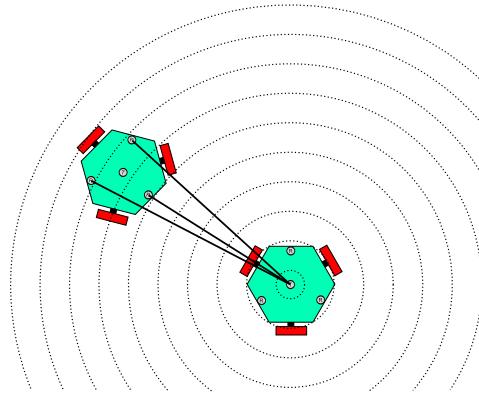


Figure 6.4: A sound wave is pulsed from one robot. The three receivers on another robot measures the pulse

6.3.1 Trilateration

Trilateration is a method dating back to ancient Greece[Laertius, 1853], and has been widely used for navigation, surveying and metrology among others.

Trilateration is used to calculate a coordinate based on properties of a triangle. An unknown coordinate can be found by knowing three coordinates and the length from them to the unknown coordinate, this opposed to triangulation which uses the angle between them to find the unknown coordinate.

The unknown coordinate can be found by measuring the distances from the three known coordinates to the unknown coordinate. By then constructing a circle around each of the known coordinates with these respective distances, the unknown coordinate can be found by looking at the intersection between these three circles (see Figure 6.5).

In the positioning system, the position of the three ultrasonic receivers are known (relative to the centre of the robot). The position of an ultrasonic transmitter generating a pulse, can be calculated by measuring the distance from each of the receivers to the transmitter. The distance can be measured by measuring the time-of-flight of the pulse from the transmitter to the receiver, and multiplying it with the velocity of the pulse.

By first assuming that the ultrasonic receivers have an equal distance apart from each other and equal distance from the centre of the robot, the equation for the

position of the transmitter is:

$$\begin{aligned} r_1^2 &= (x - x_1)^2 + (y - y_1)^2 \\ r_2^2 &= (x - x_2)^2 + (y - y_2)^2 \\ r_3^2 &= (x - x_3)^2 + (y - y_3)^2 \end{aligned} \quad (6.1)$$

Where r_i is the measured distance from the transmitter to receiver i , $\langle x_i, y_i \rangle$ is the position of receiver i , and $\langle x, y \rangle$ is the position of the transmitter. Since they have equal distance from the centre Equation 6.1 can be written:

$$\begin{aligned} r_1^2 &= x^2 + (y - d)^2 \\ r_2^2 &= (x - \sqrt{3}d)^2 + (y + \frac{1}{2}d)^2 \\ r_3^2 &= (x + \sqrt{3}d)^2 + (y + \frac{1}{2}d)^2 \end{aligned} \quad (6.2)$$

Where d is the distance from the centre of the robot to a receiver. By subtracting r_2^2 from r_3^2 and r_2^2 from r_1^2 we get:

$$\begin{aligned} x &= \frac{r_3^2 - r_2^2}{2\sqrt{3}d} \\ y &= \frac{r_2^2 + r_3^2 - 2r_1^2}{2d} \end{aligned} \quad (6.3)$$

Figure 6.5 shows a visual representation of the equations above. In order for these equations to be true, it is assumed that the distances r_1, r_2 and r_3 are so that the intersection between the circles in Equation 6.1 is exactly one point. The measured distances from the transmitter to the receivers cannot fulfil this assumption, which means that the measurements need to be processed in order to find the most likely distances. This can be done by using multidimensional scaling.

6.3.2 Multidimensional scaling

A problem that arises when measuring distances is that the measurements differ from each of the robots (Figure 6.6). 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 required to

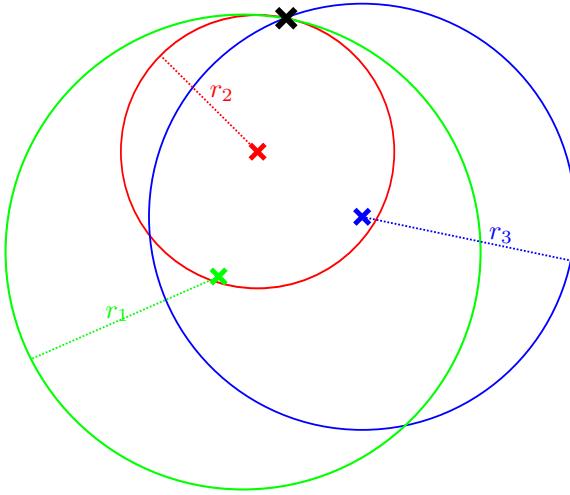


Figure 6.5: An example of trilateration. The coordinate of the black cross is the intersection of the three circles created by the distances measured from each of the known coordinates (red, green, blue) to the black cross

calculate the most plausible distances between them. The problem is formulated as

$$\delta_{i,j} = \text{robot } i\text{'s measured distance to robot } j \quad (6.4)$$

This can be put all the measurements in a *dissimilarity matrix*

$$\Delta = \begin{bmatrix} \delta_{1,1} & \delta_{1,2} & \delta_{1,3} & \cdots & \delta_{1,n} \\ \delta_{2,1} & \delta_{2,2} & \delta_{2,3} & \cdots & \delta_{2,n} \\ \delta_{3,1} & \delta_{3,2} & \delta_{3,3} & \cdots & \delta_{3,n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \delta_{n,1} & \delta_{n,2} & \delta_{n,3} & \cdots & \delta_{n,n} \end{bmatrix} \quad (6.5)$$

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

$$\delta_{i,i} = 0, \quad 1 \leq i \leq n \quad (6.6)$$

While the triangular inequality can be formulated as:

$$\delta_{i,j} + \delta_{j,k} \leq \delta_{i,k} \quad (6.7)$$

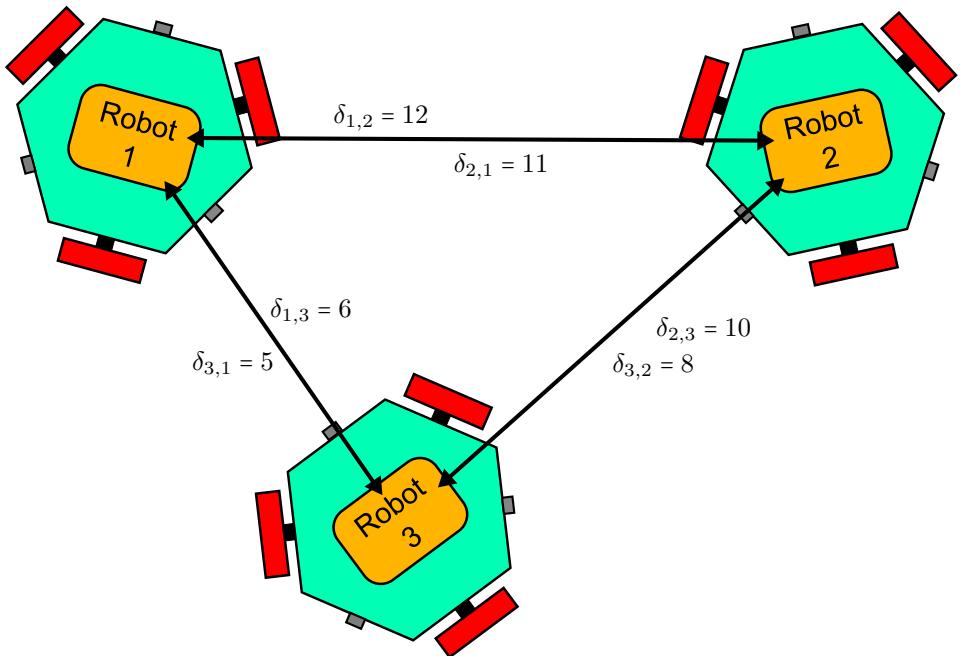


Figure 6.6: Three robots with their distances measured between each other. Equation 6.4 describes the distance δ . The distance measured from robot 1 to 2 is not the same as measured from 2 to 1

Since the entries on the matrix are distances, this can easily be proven.

An unfortunate property in this matrix is:

$$\delta_{i,j} \neq \delta_{j,i} \quad (6.8)$$

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

$$\min_{x_1, \dots, x_n} \sum_{i < j} (\|x_i - x_j\| - \delta_{i,j})^2 \quad (6.9)$$

This problem occurs because the measurements of the distances are dependent on time. Synchronized clocks will lower the time difference between the robots, which also lowers the impact of this problem.

6.4 Identification

In order for the relative positioning system to work the robots are required to differentiate each other from another. When using sound as a signal it is easy to differentiate each robot by using different frequencies. The problem with this method is that it often requires complex hardware circuits and/or fast Fourier transform to filter the different frequencies out from a sound wave. The method also requires the receivers and transmitters to be able to respectively sense and produce different frequencies.

Another solution for identifying each robot is by using *time slots*. Each robot is assigned a time slot. The time slots are fixed time intervals which are assigned to a periodic schedule. When a time slot is entered¹, the assigned robot emits a sound pulse, after some time (still within the time slot) the pulse can be measured by the surrounding robots. Since the robots know which time slot the schedule is currently in and which robot is assigned to the current time slot, the robots can identify which robot generated the pulse.

As seen in Figure 6.7, there are 4 robots interacting with each other. When the schedule enters the first time slot, the first robot sends an ultrasonic pulse. This is received at different times by each robot². When the robots enters the second time slot, the second robot sends a signal, and so forth. Since each robot has been

¹Entering a time slot is here meant as when the time within a schedule is such that the time slot goes from inactive to active, this is the opposite when regarding exit

²Note that each robot actually receives three signals, one from each ultrasonic receiver it has. These signals are then used to trilaterate the position. This has been abstracted away in the figure.

	Time slot 1	Time slot 2	Time slot 3	Time slot 4
Robot 1	Red	Green	Green	Green
Robot 2	Green	Red	Green	Green
Robot 3		Green	Green	Red
Robot 4		Green	Green	Red

Figure 6.7: Each robot transmits a signal at the beginning of their time slot (Red square). The other robots then receive the signal (Green square). Since each robot has a given time slot the other robots know which robot sent the signal.

given a unique time slot, the other robots can identify which robot sent the signal depending on which time slot the schedule is in.

Another example of this system could be: 6 robots have ultrasonic sensors as described above. In this example the time slots have a fixed 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 other robots receive and calculate the distance to it. Since the time interval is $\frac{1}{6}$ seconds, the sound emitted travels at a maximum of 56.72 meters before the pulse "enters" the next time slot. If a robot is more than the maximum distance away from another robot, this could lead to the other robot identifying the wrong robot and calculate the wrong distance to it.

The questions that could be raised in the above example is what happens if one robot moves too far from the other robots so they receive the sound wave outside the time slot? What happens when there are too many robots for the time slots to be efficient? And how to prevent or avoid these problems?

For addressing the distance question first: As recalled, the maximum distance a sound wave can travel within the time slot is 56.72 m. The sound wave will therefore probably diminish into a small wave unable to be detected by the other robots before it has travelled this distance, preventing the issue of occurring. If this is not the case, then the robot being too far away from the other robots, is

at risk of having its signal received too late (i.e. some one else's time slot). This will cripple the system, and the other robots will not be able to identify the right robot. Assuming that the robots start close enough together for this not to happen at initialization, the only reason for a robot to move 50 m away from the other robots, is if it is intentionally moving away, or it is falling behind the rest which are moving. This means that the robots can detect that one robot is moving farther away, making them able to avoid the potentially problem.

The problem can be avoided by extending the time interval, making sure that a pulse is not detected within the wrong time slot. The problem with this solution is that the sampling rate decreases, making the position of the robots refreshed less frequent. This could potentially lead to collision between the robots.

Another way to avoid the problem is by creating additional schedules with different time slot intervals. Each robot is again assigned to one time slot within each schedule, and when the time slot is entered, the given robot generates a pulse. The frequency of the pulse depends on which schedule the time slot belong to. The robots farther away than one schedule "allows", can simply ignore the frequency bound to that schedule. Using the example shown in Figure 6.7, but with have moved to far from each other. A small delay in transmission could potentially make a pulse be detected in the wrong time slot. To avoid this the robots created an additional schedule, with a time slot interval of 0.5 s (see Figure 6.7). The first schedule is then used the robots within 50 m from each other while the second is used to detect the robots within 170 m . When robot 1 enters its time slot in the first schedule it generates a pulse with a frequency of 40 kHz . As seen in the figure only robot 2 and 4 measures the signal, while robot 3 ignores it. This is because it knows that it would potentially be in the wrong time slot. In the second schedule robot 1 generates a pulse with a frequency of 41 kHz . This it repeated with robot 2, 3 and 4. Notice that neither robot 1 nor 3 listens to the pulses sent with 40 kHz . This is because they know they are at risk of receiving a signal in the wrong time slot.

Detecting robots very far away from each other is neither necessary nor interesting. Robots farther away from each other than 50 m might be out doing other objectives (making their position not interesting for the other robots), and there is no risk in colliding with them either (making detecting their position not necessary).

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 cycle of the schedule still were to be 1 second, the time slot interval would have to be $\frac{1}{100}$ 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 before it would be detected in the wrong time slot. If the time slots were to have a period of $\frac{1}{10}$ s , the robots would be able to hold the deadlines, but the complete cycle would be 10 seconds. This means that the update rate of the positions would

Schedule 1 40 Hz	Time slot 1	Time slot 2	Time slot 3	Time slot 4
Robot 1	█			█
Robot 2	█	█		█
Robot 3		█	█	█
Robot 4		█		█
Schedule 2 41 Hz	Time slot 1			Time slot 2...
Robot 1	█			
Robot 4		█		█

Figure 6.8: Example of identifying robots when the robots are to far away from each other. Robot 1 and 3 do not listen to each other in Schedule 1, but do in Schedule 2. Robot 2 and 4 do not listen to Schedule 2

be 10 seconds, which is very inefficient and could lead to potential collisions.

The solution is again to use different schedules bound to different frequencies. Having 10 schedules with frequency range from 40 kHz to 50 kHz with a step of 500 Hz , and a period of $\frac{1}{5}$ s , each robot would be able to detect each other within a range of 68 m . This means that 10 robots transmits a signal within the same time slot, but at 10 different frequencies. When a robot detects a sound, it can find out which frequency the sound has, and since it knows the current time slot it can identify which robot generated the pulse. 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.

6.5 Synchronized clocks

Clock drift is a common problem amongst all clocks. This is an almost unavoidable problem, hence a method for synchronizing the clocks in each robot is required.

Clock drift leads to *clock skew*[Krishna and Shin, 1997]. Clock skew is the difference in time between two clocks. It is safe to assume that the robots will at one time have a noticeable difference in time, meaning the clock skew between each robot is so large that measurements of time will be noticeably erroneous.

An example of a problem occurring with a large clock skew is given in Figure 6.9. Here two robots, A and B, have a clock skew of 1 s . Clock skew is relative, meaning that $T_{skew(A,B)} = T_A - T_B = 1\text{s}$ and $T_{skew(B,A)} = T_B - T_A = -1\text{s}$. Robot A starts by transmitting 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 calculated distance between them would therefore have an error of over 340 m . When robot B transmits a signal, the calculated difference would be $\Delta_t + 1$, which is an error of 340 m in the opposite direction, giving a total error of over 680 m .

A time difference of 1 second is not often achieved if the robots have the same clock architecture and the clocks are initialized at the same time, but it is still very common in computer networks. If the clock skew of the example above the error in measured distance of over 30 cm . It is therefore necessary to have a clock skew lower than 50 μs .

A clock skew will always persist and will eventually increase over time as all clocks have a clock drift, and will therefore drift apart. Hence, it is necessary to implement

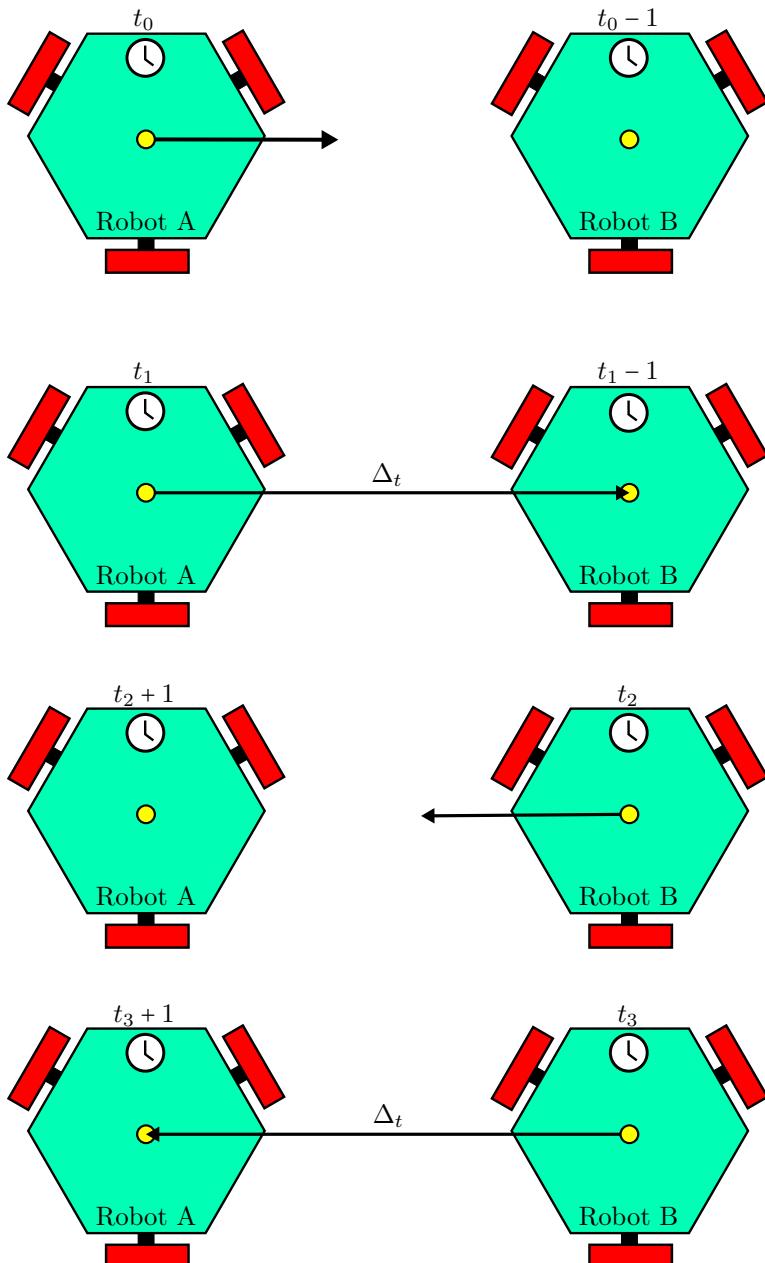


Figure 6.9: Problems with non-synchronized clocks. Robot A's clock is one second faster than Robot B's. When Robot A transmits and Robot B receives, the real time between them is Δ_t , but the calculated difference is $\Delta_t - 1$. When Robot B transmits the calculated time difference is $\Delta_t + 1$

a synchronized clock algorithm to adjust the clocks of each robot so that the clock skew is held under the required level.

The easiest method is for one robot to send its current time to another robot, and make the other robot adjust its current time to the received time. This method has an accuracy of a few milliseconds, since the transmission time of sending the data is not taken into consideration. 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 system dictating the time to the slave systems. Algorithms that use this are for instance Christian's algorithm and Berkeley algorithm.
- **Distributed systems** are systems where there are no master nor slaves, 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 actual time is not significant, only that the clock skew of the robot network is very small. Therefore neither centralized nor distributed algorithms are 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] has been looked at because of its sub-microsecond precision 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 is often 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 clock skew. 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 highly scalable, by electing many clocks to be server clocks. This also makes the protocol fault-tolerant.

Figure 6.10 shows a synchronization action with the Precision Time Protocol. First the time server sends a message with the T_1 timestamp. The client records the time T'_1 when the message is received and sends a message at T_2 with the current timestamp. This is again recorded at T'_2 . The timestamp T'_2 is sent back at T_3 . The client can then calculate the time offset \tilde{o} between it and the time server by using the equation:

$$\tilde{o} = \frac{T'_1 - T_1 - T'_2 + T_2}{2} \quad (6.10)$$

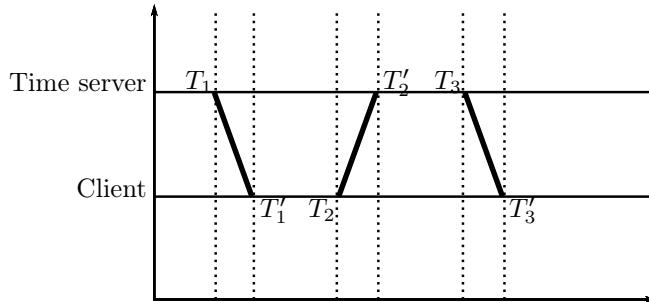


Figure 6.10: A synchronization action with the Precision Time Protocol

6.6 Implementation

6.6.1 Distance measurement

The positioning system required one transmitter and one receiver circuit.

The transmitter circuit was designed such that a digital input signal turn on or off a signal generator. The output of this generator was connected to the ultrasonic transmitter (400ST100) making the transmitter generate a ultrasonic signal. This meant that a digital pulse on the input generated an ultrasonic pulse.

The receiver circuit was to be designed so that even small signals received by the ultrasonic receiver (400SR100) would be detected and amplified. Preferably a second circuit would be constructed, gathering the received signal from each of the three receiver circuits and converting them into digital pulses which could triggering an interrupt.

The problems regarding the implementation resided in the construction of the receiver circuit. An experiment was conducted with the transmitter connected to a signal generator and the receiver connected to an oscilloscope. The distance between them was measure to 2 m , the signal generator was switched on, and the measurements of the oscilloscope was recorded. When an ultrasonic pulse was detected by the ultrasonic receiver, the measured output signal was a sinus wave with an amplitude of approximately 50 mV . The challenge was to amplify this signal. There were numerous circuit designs for this amplifier, but none of these worked properly. When a circuit design that work was found, there was no time to construct and test this properly, nor to implement the rest of the system.

Despite this, the prototypes of both an ultrasonic receiver and a transmitter circuits were constructed, and an experiment on distance detection between them has been conducted.

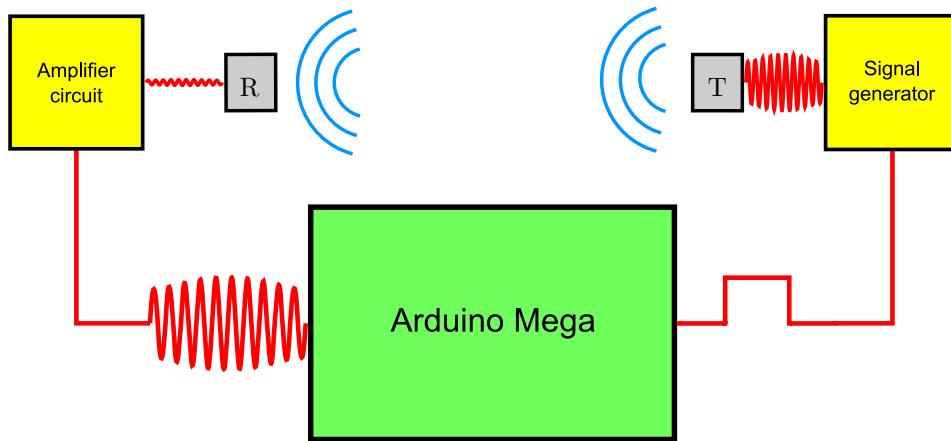


Figure 6.11: The experiment conducted of measuring distances. The Arduino Mega sends a pulse to the signal generator circuit, which generates an ultrasonic pulse. This pulse is received by the ultrasonic receiver and amplified by the amplifier circuit. The amplified signal is measured by the Arduino Mega. The time between when the pulse is sent and received is measured.

In the conducted experiment the ultrasonic transmitter and the ultrasonic receiver were placed with a distance of 50 cm apart. The transmitter generated a pulse every second and the receiver listened for the pulse. The transmitter and receiver were both controlled by the same Arduino Mega, meaning the problem with clock skew was not an issue. The time difference between the transmission and reception was recorded. The recorded time was then processed to give a measured distance between them. Some of the results can be viewed in Table 6.1.

It can be seen in Table 6.1 that the measured distances are not the actual distance between the transmitter and receiver. It can be seen that the error is approximately constant. A reason for this error may be the delay in the transmitter and receiver circuit. There is a time difference between the time the Arduino Mega sets a digital output until the time the ultrasonic pulse is generated, and between the time the ultrasonic receiver detects a signal and the time the Arduino Mega receives the signal. This constant delay can be measured and subtracted from each measurement. In this experiment the measured constant delay is approximately 2.17 ms . There are also some of the measurements with a larger error than the constant error. This could be a coincidence, or a problem in the code implementation. The code running on the Arduino Mega was designed in a big while loop. This meant that the code first generated a signal, and then enters a while loop where the Arduino Mega reads the analogue signal and the sleeps for a period of time. A time delay

Measured time [ms]	Calculated distance [m]
3.65	1.24
3.63	1.24
3.71	1.26
4.00	1.36
3.62	1.23
3.64	1.24
3.65	1.24
3.65	1.24
7.02	2.39
3.54	1.20

Table 6.1: Segment of measurements from the conducted experiment

will occur if the signal is received while the Arduino is sleeping. This error will be less of a problem if the code was running on the BeagleBone since the sampling rate is higher, and the code can be split into different applications.

6.6.2 Visual system

A visual system was implemented as an alternative to the positioning system, since the proposed system could not be implemented in time. This visual system consisted of a Kinect camera and an external computer running OpenCV [OpenCV, 1999], a framework for developing camera applications in C++.

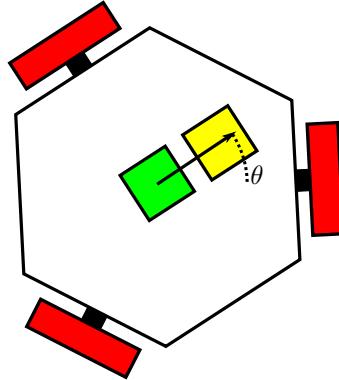


Figure 6.12: The orientation of the robots are found by calculating the angle between the centre of the green and yellow colour

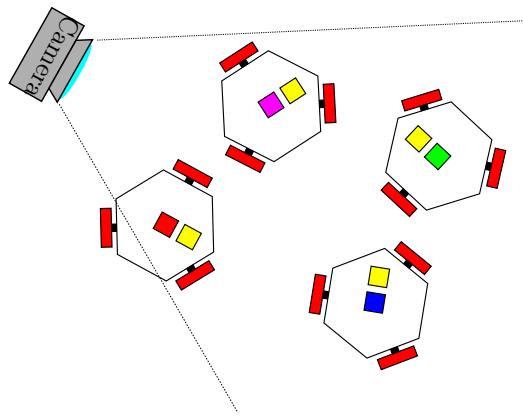


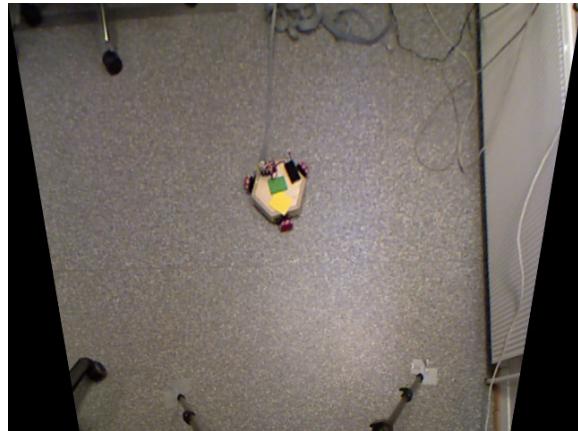
Figure 6.13: The camera views the different robots, and identifies each robot with their colour, and use the yellow colours to measure the orientation

The visual system used predefined coloured paper to identify each robot, see Figure 6.13. Each robot had one coloured paper which gave them a position and a distinct identity (either green, red, blue or magenta), and one yellow coloured paper to denote the angle of the robot, see Figure 6.12.

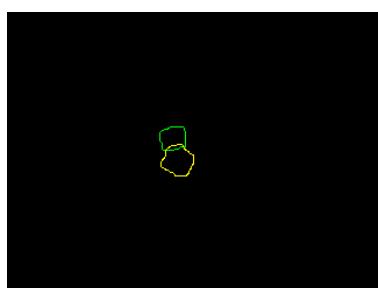
First the camera input was processed so that only the different colours (red, green, blue, magenta and yellow) were left, and the rest of the image was filtered out. This was done by selecting only the saturated (i.e. most pure) colours in the image. After this a Canny Algorithm [Canny, 1986] was used to detect the contours of the colour shapes. The algorithm was a part of the OpenCV framework, and will not be explained further. When the contours of all the colours are found, the centre of these were set as the position of the colour square, i.e. the position and angle of the robots. Some example images can be viewed in Figure 6.14.

The camera captured and processed the sensor data. Using color extraction methods each robot and their position and angle was identified, and this data was distributed to the respective robot. The complete application had a varying sampling rate of 3-4 Hz .

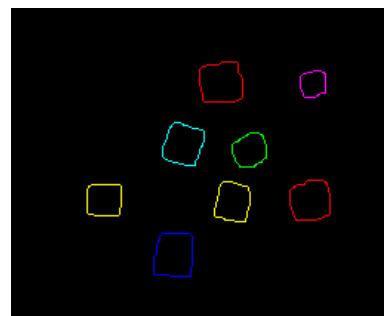
Since the implementation was done on an external computer, which was much more powerful than the BeagleBone, and the application had a sampling rate of 3-4 Hz , this shows that such an application cannot work as intended on a BeagleBone. Therefore, a camera vision as an relative positioning method is not recommended for this project, since this would require to much time to optimize the application for the BeagleBone.



(a) A camera image of a robot



(b) The contours of the coloured papers on the robot



(c) Contours of multiple coloured paper

Figure 6.14: Images of the contours of the coloured paper

Chapter 7

Discussion

This chapter will provide some hindsight of this project. The deviation from the proposed solution and the implemented solution will be discussed in depth. Each section below will discuss hindsight and experiences from each chapter. After this an overall discussion of the project is presented.

7.1 Overview

The proposed solution from the pre-study was not fully implemented. In retrospect, the proposed systems and approaches still seem to be implementable. The implemented solution has given a lot of experience and knowledge about the used components, and what issues that needs to be resolved in order for the proposed solution to function properly.

The issues that needs to be resolved on the implemented solution are few, but some of them are vital to their associated system, which prohibited the development of the multi-robot system.

Nonetheless, the multi-robot system is functional, but some aspects of the proposed solution might need to be altered in order for the multi-robot system to be able to perform as a demonstration tool.

7.2 Hardware components

Each of the components in the proposed solution were researched in the pre-study. Due to time constraints and limited budget however, not all of the components could be tested. Therefore, some of the components that were assumed to work in the intended manner, had either unexpected limitations or behaviour.

The implemented solution did not fully reflect the pre-study. One of the reasons is because in the pre-study the option of the BeagleBone was not considered. When the best candidate in the pre-study was the Arduino Mega, many of the choices were influenced by the limited resources in the Arduino Mega. When the BeagleBone was used instead, many of the components that were suited for the Arduino Mega, could have been replaced. The choices of the other components should have been questioned when the Arduino Mega was replaced. Had this been done, the outcome of the implemented solutions would have been different.

The XBee module is an example of this. The best option for wireless communication provided for the Arduino Mega is the XBee module. On the BeagleBone however, there are many more alternatives in both WiFi and Bluetooth, that are better suited for wireless communication. In hindsight of this, the main reason for choosing the XBee was because it was first chosen for the Arduino Mega. The limitation in bandwidth of broadcast feature was not discovered before late in the project. If the XBee was more thoroughly tested, this limitation would have been detected, and an alternative communication method might

The obstacle detection method used only an IR-laser range sensor. This is sufficient, but the IR-laser range sensor that was chosen was not suited for this project. Again, one of the reasons for this sensor being chosen was because it had a suitable interface to the Arduino Mega. Once the BeagleBone was used, it would have not been a problem of using either a IR-laser range sensor with higher precision, or a LIDAR.

The IMU that was supposed to be used was not obtained in time. The problems with uncertainty in delivery and cost were some of the reasons for this. The IMU was not required in the implementation, since the positioning system was purely based on camera vision.

7.3 Software systems

It can be discussed whether the control system used in this project is the right approach. As mentioned, the control system is within the reactive paradigm, which does not include any planning aspects. It would be beneficial to have a planning aspect within the control system, so that the robots would be able to perform

more complicated objectives and utilizing the potential of the multi-robot system. This discussion is outside the scope of this thesis, and may be found in [Klausen, 2013].

Fortunately, as mentioned, the multi-robot system was designed so that the control system can be replaced with other control systems, which was one of the objectives of this project.

7.4 Obstacle detection system

The Inverted Particle Filter method could also have been replaced. Since the main reason for developing this method was because of the resource constraints on the Arduino Mega, a more resource consuming method could be used on the BeagleBone. However as mentioned, there are not many obstacle detection and mapping methods which are suited for computers like the BeagleBone.

The Inverted Particle Filter is currently still the best alternative on the Beagle-Bone when the constraints are considered. This method also focuses on resource constrained systems, which is an area which requires attention.

There are as mentioned, some issues with the Inverted Particle Filter that has to be sorted in order for it to be fully functional, but these are only issues within the implementation and not within the method itself.

7.5 Positioning system

The positioning system was not implemented at all. It works in theory, and the conducted experiments supports the use of this system, but there are still issues that need to be resolved before the implementation is finished.

In the pre-study the system was almost purely theoretical. There were no experiments or implementations of the system. The previous work supported the use of this system, and it is currently still reckoned as the best alternative for this multi-robot system.

The visual system that was implemented served two purposes. It firstly made it possible for the robots to estimate their position, which made the multi-robot system functional. This made it possible for the robots to perform objectives and tasks cooperatively, which was one of the main motivations of the project.

The second purpose was that it made it possible to exclude the use of camera vision on the robots. When the visual system ran on an external computer, the system could estimate the position of the robots at a rate of 3-4 Hz . This meant

that if such a system were to be implemented on a BeagleBone, this rate would be much lower. This would not fit the requirements of the multi-robot system. It is possible to implement a camera vision system on the BeagleBone by utilizing all the potential of it, but this would steer the focus away from the main goal of the project, and potentially be too time consuming.

7.6 Final notes

The main goal of this project was to create a multi-robot system. This goal is accomplished, and the research done on multi-robot systems and cooperative behaviour has given insight into the current condition and future potential of this paradigm within robotics.

However, the time required to implement the multi-robot system and the choice of components, made the system unable to contain some of the desired features. The secondary goal of making the robots able to perform as a demonstration tool, was therefore not reached.

In retrospect of this project, it was a correct decision to compromise some of the constraints and aspects of the proposed solution, in order to accomplish the main goal of developing a multi-robot system. The resulting implementation show the advantages of cooperating robot systems, adding knowledge to the comprehensive field of multi-robot systems.

Chapter 8

Conclusion

The project for developing an autonomous cooperative multi-robot system, that started fall 2012 has had a great progress in the bestowed time.

The focus of this project has been to develop a multi-robot system which is low-cost and small in size, providing a software and hardware base which can grow with added knowledge and technology.

The proposed solution that was presented in the pre-study was implemented. The implemented solution did not meet some of the requirements of the proposed solution. Despite this, the implemented solution is still a platform able to interface different cooperative behavioural control systems.

The hardware and software platform that were developed has the potential to be a framework for multi-robot systems. The main motivation of this platform was to create a multi-robot system with an interface to a cooperative behavioural control system. The implemented platform does this, though it has been changed from the original proposed solution.

The developed Inverted Particle Filter method for obstacle detection and mapping, provides a new direction in obstacle detection and mapping. The method focuses on accuracy and flexibility in resource constrained systems. The implementation of this method used an IR-laser range sensor to measure the distance to the surrounding obstacles.

A system for position estimation has been proposed. The proposed system uses ultrasonic sensors to measure the relative distance between each of the robots. The system focuses on scalability and flexibility, proposing approaches for making the system applicable on larger multi-robot systems. The proposed system was not fully implemented, but the conducted experiments show promising results.

The resulting multi-robot system in this project has the potential of becoming a fully functional multi-robot system meeting all the requirements and constraints set up for the project. This would be a contribution to the field of robotics, which lowers the cost threshold, opening up for others to research multi-robot systems performing cooperatively at lower cost.

Bibliography

- [BeagleBoard.org, 2011] BeagleBoard.org (2011). BeagleBone. *beagleboard.org/bone*.
- [Burgard and Moors, 2000] Burgard, W. and Moors, M. (2000). Collaborative multi-robot exploration. *... and Automation, 2000* . . . , (April).
- [Campbell and Wynne, 2011] Campbell, J. and Wynne, R. (2011). *Introduction to Remote Sensing*.
- [Canny, 1986] Canny, J. (1986). A computational approach to edge detection. *IEEE transactions on pattern analysis and machine intelligence*, 8(6):679–98.
- [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.
- [Klausen, 2013] Klausen, K. (2013). *Cooperative Behavioural Control for Omni-Wheeled Robots*. Student project, NTNU.
- [Kleppe, 2012] Kleppe, A. L. (2012). Hardware platform for a Multi-Robot System.
- [Krishna and Shin, 1997] Krishna, C. M. and Shin, K. G. (1997). *Real-Time Systems*.
- [Laertius, 1853] Laertius, D. (1853). *The Lives and Opinions of Eminent Philosophers*.
- [Lowe, 2004] Lowe, D. G. (2004). Distinctive Image Features from Scale-Invariant Keypoints. *International Journal of Computer Vision*, 60(2):91–110.

- [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.
- [Murphy, 2000] Murphy, R. (2000). AI Robotics.
- [NASA, 2012] NASA (2012). Curiosity: NASA’s Next Mars Rover. http://www.nasa.gov/mission_pages/msl/index.html.
- [OpenCV, 1999] OpenCV (1999). Homepage of the OpenCV project. opencv.org.
- [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.
- [Rivard et al., 2008] Rivard, F., Bisson, J., Michaud, F., and Letourneau, D. (2008). Ultrasonic relative positioning for multi-robot systems. *2008 IEEE International Conference on Robotics and Automation*, pages 323–328.
- [Sperre et al., 2012] Sperre, A. H., Halvorsen, A., and Myren, S. R. k. (2012). *Eurobot NTNU 2012*. Master thesis, NTNU.
- [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.