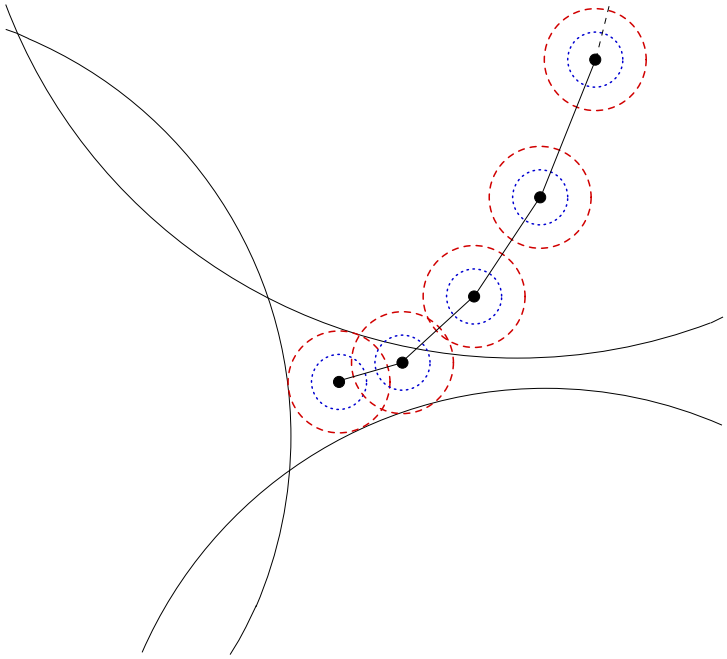
CHALMERS
UNIVERSITY OF TECHNOLOGY



Coordination of robots via a wireless network

Bachelor's thesis in Signals and Systems

OSCAR BERONIUS
EMANUEL LUNDÉN
MARCUS MALMQUIST
AGNES ROHLIN

Supervisors: MARKUS FRÖHLE, THEMISTOKLIS CHARALAMBOUS
Examiner: HENK WYMEERSCH

Department of Signals and Systems
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2016

Bachelor's thesis 2016:SSYX02-16-27

Coordination of robots via a wireless network

OSCAR BERONIUS
EMANUEL LUNDÉN
MARCUS MALMQUIST
AGNES ROHLIN

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Signals and Systems
Chalmers University of Technology
Gothenburg, Sweden 2016

Coordination of robots via a wireless network
OSCAR BERONIUS
EMANUEL LUNDÉN
MARCUS MALMQUIST
AGNES ROHLIN

Supervisor: Markus Fröhle, Department of Signals and Systems
Supervisor: Themistoklis Charalambous, Department of Signals and Systems
Examiner: Henk Wymeersch, Department of Signals and Systems

Bachelor's Thesis 2016:SSYX02-16-27
Department of Signals and Systems
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Image representation of position estimation being improved through gradient descent.

Typeset in LaTeX
Gothenburg, Sweden 2016

Acknowledgements

We would like to thank our main supervisor Markus Fröhle for guiding us, as well as Themistoklis Charalambous for his valuable input throughout the project. We would also like to thank Henk Wymeersch, our examiner.

Oscar Beronius, Emanuel Lundén, Marcus Malmquist and Agnes Rohlin, Gothenburg, May 2016.

Abstract

The purpose of this project is to build a system comprised of autonomous robots, that can be coordinated according to some objective function. It includes modeling a suitable objective function that can be used to optimize the quality of a relayed signal, if such a signal were sent between the robots in the system. Such signals would be transferred over Wi-Fi and it is used for the overall communication.

The results are two methods that minimizes the distances between a base station via the robots (of model *Pioneer 3-DX*) to an end node. Thus the robots ended up on a straight line between the base station and the end node without colliding. The main methods used to produce the results were through the use of ROS (Robot Operating System), position estimation using trilateration, more accurately with extended Kalman filters, and RCMs (Ranging and Communication Modules).

Concluding, two movement algorithms, extended Kalman filters, two coordination algorithms with the objective function of minimizing said distances and a simple collision avoidance were developed. For further development, it is recommended to improve the collision avoidance or applying a different objective function to the second coordination algorithm.

Sammandrag

Syftet med projektet är att implementera ett system bestående av autonoma robotar som kan koordineras utifrån en målfunktion. Det innefattar modelleringen av en lämplig målfukntion som kan användas för att optimera kvalitén hos en signal, om en sådan var skickad emellan robotarna i systemet. Sådana signaler skulle överföras över Wi-Fi och det är vad som används för den övergripande kommunikationen.

Resultatet är två metoder som minimerar avståndet mellan en basstation via robotarna (av modell *Pioneer 3-DX*) till en slutnod. Således hamnade robotarna på en rak linje mellan basstationen och slutnoden utan att kollidera. De främsta metoderna som användes för att producera resultaten var med hjälp av ROS (Robot Operating System), positionsuppskattning med hjälp av trilateration, noggrannare med ett ickelinjärt Kalmanfilter och användandet av RCMs (Ranging and Communication Modules).

Sammanfattningsvis, två förflyttningsalgoritmer, två ickelinjära Kalmanfilter, två koordnieringsalgoritmer med målfunktion att minimera sagda avstånd, och en enkel kollisionsundvikningalgoritm utvecklades. För framtida utveckling är det rekommenderat att förbättra kollisionsundvikningsalgoritmen eller att använda en alternativ målfunktion till den andra koordineringsalgoritmen.

Contents

Abbreviations

- Ranging and Communication Module - RCM
- Ultra-wideband - UWB
- Robot Operating System - ROS
- An object that acts as a receiver and a transmitter - Transceiver
- Advanced Robot Interface for Applications - ARIA
- Wireless Access Point - WAP
- Service Set Identifier - SSID
- Media Access Control - MAC

1

Introduction

With the technological advancements in robotics and networking, the usage of autonomous robots has seen a steady increase in many different areas, including industry, elder care and security. An autonomous robot can be defined as a robot that is able to plan and execute its own actions by adapting to its surrounding environment, without the need of human intervention. A team of autonomous robots can act as a coordinated network, where the robots cooperate in performing different tasks. In this project that concept will be called an autonomous robot system, and it includes the communication, planning and performing of actions for each robot depending both its environment as well as the actions of other robots in the system. An example of such an autonomous system can be found in the warehouse developed by Amazon Robotics LLC, where a network of robots is being used to transport items [1]. The system is autonomous in a way that the movements of all robots are executed so that no collisions will occur between them, despite the limited space in which they operate, meanwhile optimizing their routes for reduced total travel time. Another example is a mobile robot system patented by Jon Rigelsford at The University of Sheffield which, using one system navigator robot and multiple functional robot, performs predetermined tasks. In [2, 3], an application of cleaning the interior of a house was presented.

The area of application that forms the basis for this project is the possibility to extend a signal between a base station and a remotely controlled robot, called end node. If a base station cannot establish a connection to its end node, for example due to distance, obstacles or distortions, it can instead connect to a relay robot. The relay robot would then act as a relay for the end node and the base station so that information, such as movement signals, can be passed along this communication network. In other words, a relay would act as a signal extender and can be used when the connection is not strong enough. This can also work in multiple steps, so that a chain of relays can be implemented to even further extend the signal range as can be seen in Figure 1.1. A scenario where this technology could be useful is for example during a meltdown in a nuclear power plant. Such an environment is typically very dangerous for humans to be close to, and if the need of emergency reparations is essential to help prevent further disaster, robots could be deployed to handle the task instead. Because the safe operating distance could be relatively far away, the usage of relay robots to extend control signals could potentially work as a solution to the problem.

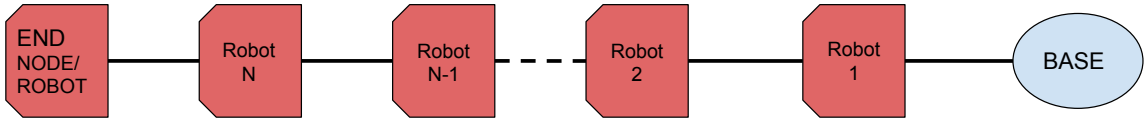


Figure 1.1: A relay network of robots where an end node sends information to each robot that only has connection with its neighbors, ending up at the base station where the robots are controlled from and the information is received.

Since the goal of the relay robots is to forward a signal with optimal quality which is directly related to the distance between TX and RX, the optimal quality can be accomplished by optimizing the positions of each robot. The optimal position of each relay depends on the positions of the base station and end node, moving the end node would require the relays to be continuously repositioned accordingly. This introduces the need for a solution comprised of coordinated robots with autonomous behavior, where each robot's position is optomized continuously as the end node's position changes. A similar research has been done by Takaaki Saitou *et al* in [4].

1.1 Purpose

The purpose of this project is to build a system comprised of coordinated autonomous robots that is able to assume a position configuration determined by minimizing a given objective function.

1.2 Assumptions, limitations, and scope

The scope for this project is to make two relay robots position themselves autonomously in regard to a base station and end node, so that a connection between each pair of relays would be optimal. The position of the end node will be simulated, and the base station should be able to be manually moved. The communication is to be held over Wi-Fi. A limitation for the project is to use a centralized network instead of a decentralized network, implying that no signal will be sent between relays. The movement of the robots should be done in such a way that they can move autonomously without the risk harming themselves.

1.3 Problem description

In regards of the scope the problems to be solved focuses mainly on the coordination of the relay robots and the necessary parts it includes. Naturally there are also a couple of prerequisites needed to be able to implement any form of coordination such as position estimation and communication tasks. The problems can be separated into the following areas.

For the communication, the following problems should be solved:

- Create a centralized network using appropriate hardware.
- Set up a communication structure suitable for the coordination.
- Find a suitable way to communicate with the robots.
- Find a suitable way to communicate with the hardware used for position estimation.

A need for position estimation introduces the following problems:

- Decide on suitable hardware to make distance measurements with.
- Implement a way to estimate a robot's position.

Finally, the coordination of the relay robots requires solving the problems that follow below:

- Implement an algorithm to move robots to a target position.
- Define an objective function that models the optimal connection between relays.
- Implement collision avoidance.
- Implement a coordination algorithm that places all of the relay robots according to a given objective function.

2

Theory

This section aims to explain the theoretical foundation of this project. To be able to understand it, some knowledge is needed in the fields of linear algebra, mathematical statistics and multivariable calculus.

2.1 Gradient descent

Consider a one-dimensional real-valued function taking an n-dimensional real-valued argument ($f\colon \mathbb{R}^n \to \mathbb{R}^1$). The gradient of a function f (denoted as ∇f) is defined as

$$\nabla f = \sum_{i=1}^{n} \frac{\partial f}{\partial x_i} \hat{x}_i \tag{2.1}$$

and is an n-dimensional vector describing the direction in which the image of the function changes the most. The norm (length) of the gradient describes how much the function value changes in the direction of the gradient. See [5] for further details on gradient.

Gradient descent (sometimes called *steepest descent*) is an iterative method that, given a function f and an initial coordinate

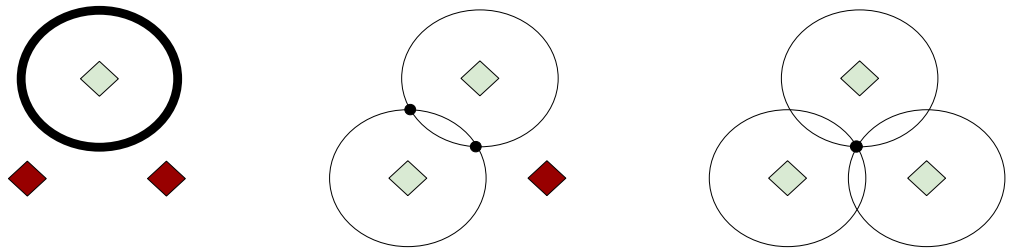
$$\vec{x}_0 = \sum_{i=1}^{n} x_{0_i} \hat{x}_i \tag{2.2}$$

in the function's domain, attempts to find a local minimum of the function where $\nabla f = \vec{0}$. The method accomplishes this by calculating the gradient at a given point, take a step ($k\nabla f(\vec{x}_0)$ for some value of $k < 0$) in the direction of the gradient and calculates the gradient in the new position and so on. Mathematically this is described by $\vec{x}_i = \vec{x}_{i-1} + k\nabla f(\vec{x}_{i-1})$. If finding a local maximum of a function f is of interest, one can use gradient descent with a positive step length ($k > 0$) or using gradient descent on a function $g = -f$. For more details, see [6].

2.2 Trilateration

Consider a two-dimensional system with some points at fixed positions in some reference coordinate system, and some objects with unknown positions in the same reference coordinate system. Assuming

it is possible for an object to measure its distance to each point, it is possible to calculate the position of the object using the distance to at least three of the points [7]. How this works is shown in Figure 2.1. Figure 2.1a shows how knowledge of the distance from an object to one point implies that all possible locations for the object are represented by a circle around the point, with a radius equal to the distance. Figure 2.1b shows all possible locations of the object if distances to two of the points are known. Since the possible locations of the object are equal to a circle around a known point, then if two points are known, the possible locations would have to be one of the two intersections of the circles. A special case of this scenario is if two points are on top of each other, in which case the possible locations of an object would be the resulting circle projection. For this project, this is assumed to not be a possibility. Figure 2.1c shows a scenario where the distance of an object and three points are known. The combined intersections of all circles result in one point, which is equal to the position of the object.



(a) The infinite possible positions when one point's position is known.

(b) Two possible positions when two points' positions are known.

(c) Only one possible position when three points' positions are known.

Figure 2.1: The three stages towards trilateration. The green diamonds indicate a known point of which the distance to the object can be measured, and a red diamond indicates an unknown point. A ring is the measured distance from the object to the point. The dots on the rings are the possible positions of the object being measured.

2.3 Extended Kalman filter

The *extended Kalman filter* [8] is a version of the *Kalman filter* [9] which linearizes the model every iteration. The filter uses information about the previous state, control input as well as measurement of the states to estimate the next state of the system. Assuming the model of the state has some uncertainty, the controls and measurements are noisy then neither of them are very likely to be accurately executed or measure the true state respectively. The filter uses the prediction from the controls as well as the measurements to estimate the next state. Estimating the new state consists of two parts: predict and update. During the prediction part, the next state is predicted using the last state and the control input. During the update part, the next state is estimated from the weighted average of the predicted state and the measured state. For further information about extended Kalman information see [8]. The extended Kalman filter will now be explained mathematically. Denotations for matrices will be similar to those used in [10].

Consider, at some discrete time instant k, a system at state $\vec{x}_k$. It is desirable to use the available information about the system (such as control input model, observation model and noise model) to calculate a weighted average of the predicted state (using the control input model and control noise model) and the observed state (using the observation model and observation noise model), which is

the most accurate estimation of the true state that is available.

The state $\vec{x}_k$ is a function of the previous state $\vec{x}_{k-1}$, control input $\vec{u}_{k-1}$ and process noise $\vec{w}_{k-1}$ (assumed to be zero-mean white Gaussian noise) which has a covariance matrix $\mathbf{Q}_{k-1}$.

$$\vec{x}_k = \vec{f}(\vec{x}_{k-1}, \vec{u}_{k-1}, \vec{w}_{k-1}) \tag{2.3}$$

Thus the prediction of the this state, $\hat{\vec{x}}_{k|k-1}$, is a function of the previous estimated state $\hat{\vec{x}}_{k-1|k-1}$ and the control input.

$$\hat{\vec{x}}_{k|k-1} = \vec{f}(\hat{\vec{x}}_{k-1|k-1}, \vec{u}_{k-1}) \tag{2.4}$$

The predicted state covariance matrix $\mathbf{P}_{k|k-1}$ is then calculated using the previous state covariance matrix $\mathbf{P}_{k-1|k-1}$, the state transition matrix

$$\mathbf{F}_{k-1} = \frac{\partial \vec{f}}{\partial \vec{x}}\Big|_{\hat{\vec{x}}_{k-1|k-1}, \vec{u}_{k-1}}, \tag{2.5}$$

the process noise covariance matrix $\mathbf{Q}_{k-1}$ and the process noise transition matrix

$$\mathbf{L}_{k-1} = \frac{\partial \vec{f}}{\partial \vec{w}}\Big|_{\hat{\vec{x}}_{k-1|k-1}, \vec{u}_{k-1}}. \tag{2.6}$$

The predicted state covariance matrix can then be calculated according to

$$\mathbf{P}_{k|k-1} = \mathbf{F}_{k-1}\mathbf{P}_{k-1|k-1}\mathbf{F}_{k-1}^T + \mathbf{L}_{k-1}\mathbf{Q}_{k-1}\mathbf{L}_{k-1}^T. \tag{2.7}$$

The observation (measurement)$\vec{z}_k$ of the state is a function of the actual state $\vec{x}_k$ and the observation noise $\vec{v}_k$ (assumed to be zero-mean white Gaussian noise) which has a covariance matrix $\mathbf{R}_k$

$$\vec{z}_k = \vec{h}(\vec{x}_k) + \vec{v}_k. \tag{2.8}$$

Comparing the observation of the true state and the observation of the predicted state results in the measurement residual $\tilde{y}_k = \vec{z}_k - \vec{h}(\hat{\vec{x}}_{k|k-1})$. The residual covariance matrix is then calculated (in a similar fashion as the predicted state covariance) using the previous state covariance matrix $\mathbf{P}_{k-1|k-1}$, the observation matrix

$$\mathbf{H}_k = \frac{\partial \vec{h}}{\partial \vec{x}}\Big|_{\hat{\vec{x}}_{k|k-1}}, \tag{2.9}$$

and the observation noise covariance matrix $\mathbf{R}_k$. The residual covariance matrix can then be calculated by

$$\mathbf{S}_k = \mathbf{H}_k\mathbf{P}_{k|k-1}\mathbf{H}_k^T + \mathbf{R}_k. \tag{2.10}$$

The Kalman gain $\mathbf{K}_k = \mathbf{P}_{k|k-1}\mathbf{H}_k^T\mathbf{S}_k^{-1}$ in then used to calculate the weighted average of the measured state and the predicted state. Finally, the estimated state and covariance matrix is then updated as

$$\hat{\vec{x}}_{k|k} = \hat{\vec{x}}_{k|k-1} + \mathbf{K}_k\tilde{y}_k \tag{2.11}$$

$$\mathbf{P}_{k|k} = (\mathbf{I} - \mathbf{K}_k\mathbf{H}_k)\mathbf{P}_{k|k-1}. \tag{2.12}$$

3

Equipment and software

This section explains the hardware and software used in this project. To be able to recreate a similar project it is not necessary to use exactly the same hardware, but it is recommended to use something similar. Since the predominately used software in the project called *ROS* requires some previous knowledge, it has its own dedicated section.

3.1 Hardware

In order to set up a network with robots that can be controlled autonomously, hardware is necessary. There is also a need for a communication structure where information can be sent and received. Lastly a system for measuring and determining the position of the robots is also necessary. For this project the following equipment was used.

- *Ultra-wideband* (UWB) transceivers (PulsON P400 RCM) [11], were used to provide time of arrival measurements for the position estimation. They are from here on referred to as *RCMs*. It is necessary to have at least three RCMs to get a position in two-dimensions as explained in Section 2.2.

- Robots of the model *Pioneer P3-DX* were used for this project. They are compact differential-drive mobile robots equipped with motors (with two degrees of freedom), wheels and sonar [12].

- A local Wi-Fi set up with *Netgear N150 Wireless USB Adapters* [13].

- To combine the above hardware, laptop computers with Linux Ubuntu 14.04.4, *Trusty Thar* (see Section 3.2 for explanation) [14] were used.

3.2 ROS and ROSARIA

The software that was used for this project is an open source operating system for robots called *Robot Operating System* (ROS) which provides functionality for working with a variety of robots, including the robots that are used in this project [15]. It is licensed under the Creative Commons Attribute 3.0 [16].

The library that enables communication with the Pioneer robot in ROS is called *ROSARIA* [17]. It integrates *MobileRobots' Advanced Robot Interface for Applications* (ARIA) SDK which handles basic communication with a robot within ROS [18]. ROS is implemented in multiple modern programming languages such as C++ and Python. Python was used for this project. To install and run the current stable version of ROS, *Indigo Igloo*, a desktop OS of Linux Ubuntu of version 14.04.4 (called *Trusty Tahr*) is required [14, 19]. To install the ROSARIA library, see [17].

To be able to understand the way this project is implemented within ROS and ROSARIA, this section will provide an overview of how ROS works and its main functionality that is made use of in this project. This section and later sections in the report that includes ROS but does not have a reference attached is entirely based on the ROS wiki (located at `wiki.ros.org`). References to specific articles in the wiki that provide better understanding for the subject will be referred to in the Bibliography. The wiki has served as a course book to ROS and parts of the implementations it contains has been used.

3.2.1 Cores and nodes

ROS is built around what is called *cores* and *nodes*. A ROS core is the foundation for the entire operating system. Essentially it is a bundle of different nodes and programs necessary for a ROS system. The main objective of the core is to supply a way to communicate between different nodes; a core *must* be running if nodes are to be launched. ROS nodes can be viewed as programs wrapped in a ROS shells, which execute computations. A node can contain any functionality desired, but wrapped in the ROS shell that enables communication with other nodes on the same core.

3.2.2 Topics and messages

There are mainly two ways of communication between nodes, *topics* and *messages*. Topics can be seen as a bus between nodes. Messages can either be of types that ROS already acknowledges or message types can be declared within the ROS environment. These messages can be customized so that they are exactly of the type which the nodes can handle. To handle these communication types there are mainly two communication patterns used which are described in the following section.

3.2.3 Communication patterns

To handle topics, a *subscriber-publisher* pattern can be used, and to handle messages a *service-client* pattern respectively. The subscriber-publisher pattern consists of a subscriber side that listens to a topic and a publisher side that publishes to the same topic repeatedly. These sides can be written stand-alone or within nodes with other functionality. The service-client pattern consists of a client that sends a request to a specified service. Once the request is received, the service side processes this request, and returns a value if specified. Meanwhile, the client side of this pattern waits for the response message to arrive before continuing its execution. This is synonymous to a remote-procedure-call of a function.

3.2.4 The ROSARIA node

ROSARIA provides multiple implementations of basic ARIA functions such as movement signals and publishers for core information about the robots. For example, one of the topics (*cmd/vel*) that ROSARIA provides takes *twist* messages, which are movement speed instructions in two degrees of freedom. The `linear.x` *parameter* is the vertical speed in m/s and the `angular.z` *parameter* is the rotation in rad/s. See Figure 3.1 for an illustration of how the robot can be moved. The *cmd/vel* topic makes use of Aria to command a connected robot to execute movement signals specified by the given *twist* message. Moving a robot a specific length can be accomplished by sleeping the execution of a ROS node for a certain amount of time using *rospy.sleep()*, before resetting the last *twist* message.

The topic then translates the message into the ARIA SDK and commands the robot to set the velocities accordingly. To allow the robot time to execute this movement, sleeping the execution in the ROS nodes becomes a useful tool. This way one can publish a movement speed, sleep for a specified time and move a specific distance. More information on how to use the ROSARIA node can be found at [20].

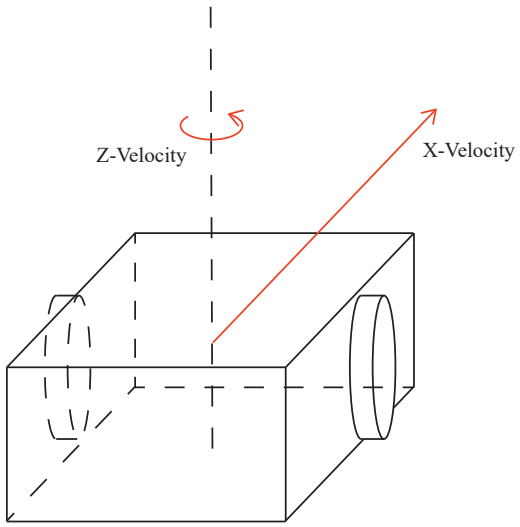


Figure 3.1: Illustration of how a robot can be moved.

4

Method

As explained in Section 1.1, the purpose of this project was to implement well-functioning coordination within an autonomous system of robots. This meant that some prerequisites were required in order to simplify the implementation of the coordination, namely two: *communication* and *positioning*. Furthermore, movement algorithms were required in order to determine the actions of each robot for every time step. Once the communication and positioning had been implemented, the work on the coordination could proceed. In the process of working towards a coordination algorithm there were two fully functioning and different versions developed which are explained in Section 4.3. For comparison on the ability and quality of the coordination algorithms see Section 5. To avoid any confusion, a computer connected to a physical robot will be called *relay computer*, the hardware used for position measurements will be referred to as *RCM*, the physical robot will be referred to as *robot*, and the whole package will be referred as a *relay*.

4.1 Prerequisites for building an autonomous robot system

The first prerequisite, communication, entails the communication between base station, relays and end node as well as the communication within a relay. To be able to send the appropriate data, establishing communication is necessary. Also, to facilitate the calculations for coordination algorithms, information about each relay's position had to be known. So the second prerequisite entailed the determining of each relay's position relative to a known reference frame.

4.1.1 Communication

To enable communication with the involved hardware components some configuration had to be done. It is not possible to remotely communicate with a robot or RCM directly and so each of the robots and RCMs had to be connected to relay computers. How the configuration of the relay was done is explained later in this section, in *Configuring a relay*.

As mentioned in Section 1.2, a centralized network was used for this project. The centralized network approach meant that, given that the relays provided some information (for example position), all calculations would be performed on a base station (a computer acting as a base station). Once the calculations had been made the appropriate actions would be sent back to each relay. How the

network structure was set-up is explained in the subsection *Configuring a centralized network*.

The actual communication had to be structured to fit the purpose of the project. The appropriate communication patterns had to be implemented in order to send the necessary data within ROS. The communication throughout ROS is explained in *Communicating through the network in ROS*.

Configuring a relay

The relay, as previously mentioned, consists of one computer connected via USB-serial to the robot and a RCM connected via Ethernet. Some configuration had to be done in order to communicate with each of these hardware components. Firstly, for the relay computer, the ROSARIA package had to be installed and a ROSARIA node running to enable any communication to the robot. As mentioned in Section 3.2.4 the ROSARIA node was then controlled with the use of a subscriber-publisher pattern in ROS where movement commands were published to the topic provided by ROSARIA. Secondly, for the RCM, the Ethernet had to be configured so that the RCM can be pinged, see Appendix C for instructions. If the RCM can be pinged, it can be communicated with via UDP and sockets. Once this configuration was made, the relay could be used. The final relay configuration can be seen in Figure 4.1.

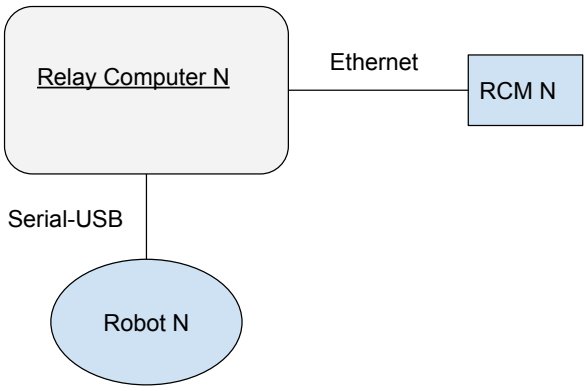


Figure 4.1: The components of a relay. It consists of a laptop connected to a robot and a RCM. These are connected via Serial-USB and Ethernet respectively.

Configuring a centralized network

To enable communication for a ROS structure there is a need for a *ROS network*. As can be read in Section 3.2.1 this requires *one* ROS core to be started and ROS nodes with different functionality can be launched upon this core and communicate with each other. This is usually done on a single computer, but since this project requires ROS nodes from different computers to communicate (base station and relay computers), the ROS network had to be introduced on a higher level. A ROS network like this requires a bi-directional Wi-Fi network and each computer needs a hostname resolvable and known by all other computers.

The network was set-up using Wi-Fi USB adapters mentioned in Section 3.1 with static IP-addresses.

Making the Wi-Fi USB adapter, connected to the base station, configured as a Wireless Access Point (WAP or simply AP) and the other Wi-Fi USB adapters, connected to the relay computers, configured as clients. The relay computers could then connect to the WAP and a bi-directional network was created. To configure the Wi-Fi USB adapters, the *hostapd* user daemon from Linux Wireless [21] (protected under the Creative Commons Attribution 4.0 International License [22]) was used with adjustments towards the network structure needed for this project. Instructions on how the configuration was done are provided in Appendix A and additional information is available at [23].

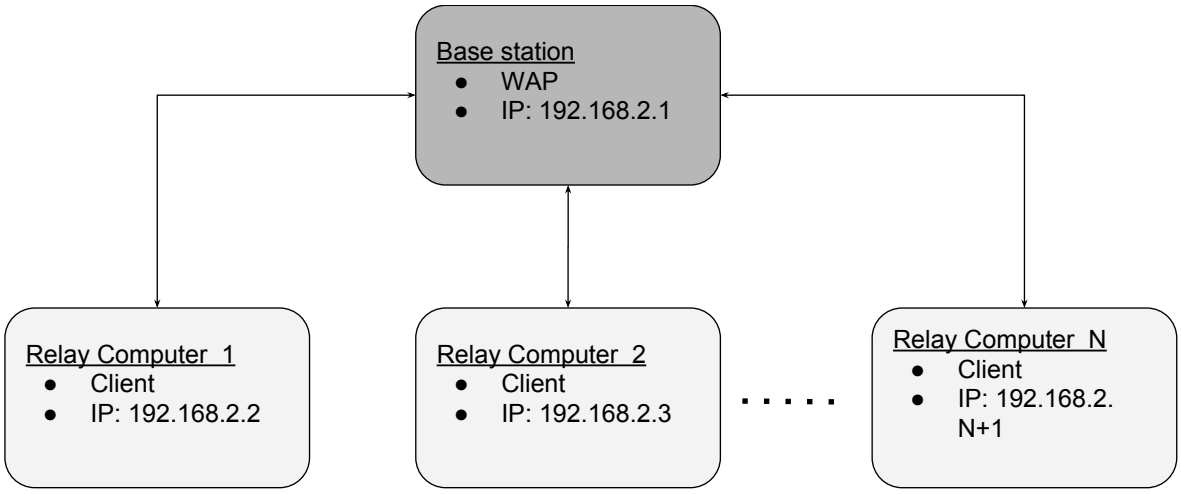


Figure 4.2: The physical network. Base station is the computer designated the role of a WAP that connects all other relay computers as clients with the given IP addresses.

The network structure (see Figure 4.2 for an overview) enables a ROS network with multiple computers to be set-up because it fulfills the two requirements. The first requirement with bi-directional communication is satisfied. The second requirement with known hostnames and corresponding IP addresses is also satisfied. This requirement is satisfied since the network was set up with static IP addresses and hostnames that were chosen and therefore known. The static IP addresses enable the configuration of the ROS environmental variables that have to be set (such as the hostname and port where the core is launched) when configuring the ROS network for multiple computers. A step by step tutorial on how to set up the multiple machine ROS Network can be found on the ROS wikipedia at [24]. Once this ROS network was up and running, communication between all computers was enabled.

Communicating through the network in ROS

The coordination algorithms requires the base to have some communication structure inside ROS in order to request and receive data from the relays. For this, the service-client structure mentioned in Section 3.2.3 was used. It allowed the base station to request data from a relay computer, wait for a response while the relay computer retrieves the data, and then use the received data to make some computations.

Different services were set-up for different message types so that the relay computer runs multiple services where each service waits for the base station to make requests. For instance the position data acquired by the RCMs was requested this way. This service-client structure was also used the other

way around. Instead of requesting data to be sent back to the base station, the service receives a command instead of a request. The command is then executed by the relay computer by publishing it to the physical robot, as explained in Section 3.2.4. Lastly, an acknowledgment message is sent back to the base station, indicating that the command was executed. This version of the service-client structure was for example used for motion commands.

With the network set-up as well as the communication structure, the final ROS network can be seen in Figure 4.3. The base station uses service requests to communicate with the relay computer and the relay computer publishes data to the physical robot. The RCMs are left out since their communication is done outside of ROS with python function calls instead.

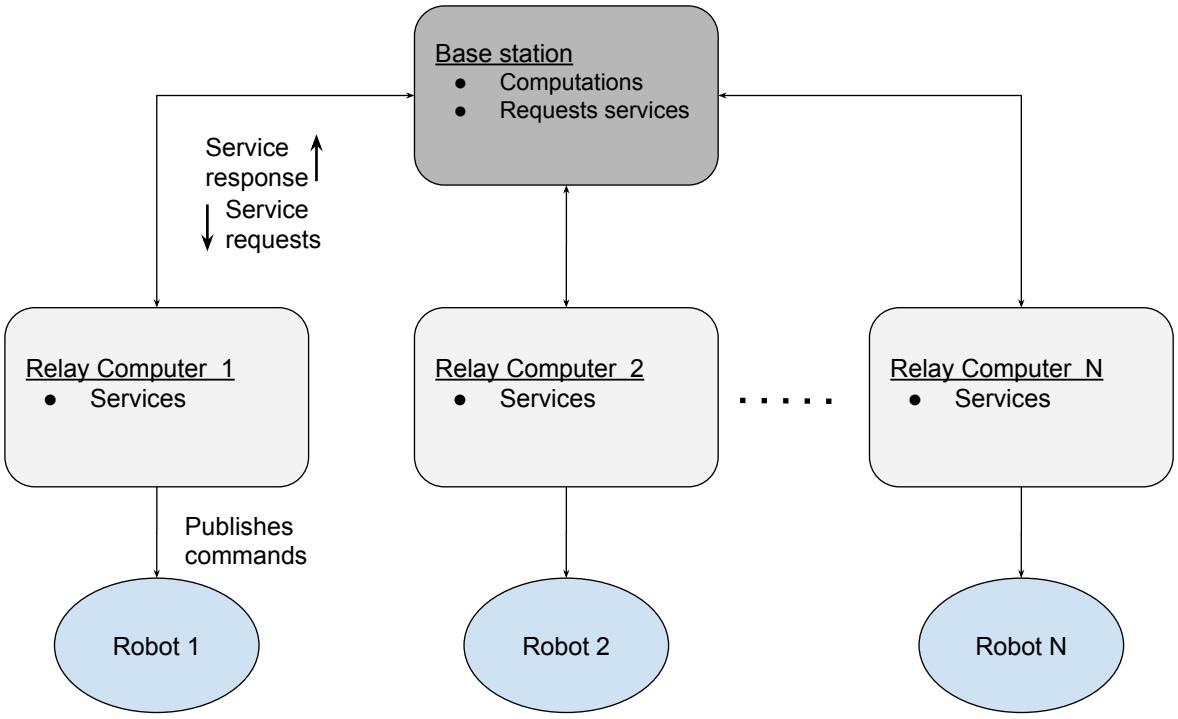


Figure 4.3: An overview of the ROS network structure. The base station performs all the computation and calls upon services located on the relay computers to execute with service requests, followed by the execution and service response being sent back. Commands may be sent to the relay robots at the execution of services.

4.1.2 Position estimation

In order to determine where to move the relays during coordination, as well as calculating movement signals denoted *twist messages*, knowledge of each relay's position is required.

To determine the position of a relay, the distance between the relay and some reference positions was used. The distance was calculated by measuring the time of arrival (TOA) using a RCM. This is explained in Section below. Because the process of measuring the TOA is a subject to noise, there will be some uncertainty in the measured position. This uncertainty can either be decreased by performing several measurements, which takes longer time, or by using a filter.

A Kalman filter was used to more accurately estimate the position of each relay using only a few measurements. As explained in Section 2.3 an extended Kalman filter uses controls and measurements to calculate a new state, for this implementation it uses the controls sent to the relay computers as well as the measured position to estimate a position. This position estimation is more accurate than either measurement and prediction alone. The use of the Kalman filter is explained in Section below.

Estimating a position using RCMs and trilateration

In order to relate the relays' positions to each other, a reference coordinate grid is needed. This was set up by placing three stationary RCMs at known locations, called *anchors*. Although only two anchors are required to build a two-dimensional reference coordinate grid, it is however required to have three anchors to determine a relay's position. The relays would measure how far away they were from each of the anchors and use this information to calculate their own position using trilateration as explained in Section 2.2.

The measuring process was done by requesting a range measurement from each of the anchors and use this information to minimize the residual in an iterative process. The residual is

$$f(\vec{x}) = \sum_{i=1}^{N} (\vec{d_i} - \vec{r_i}) \cdot (\vec{d_i} - \vec{r_i}) = \sum_{i=1}^{N} ||\vec{d_i} - \vec{r_i}||^2 \tag{4.1}$$

where $\vec{r_i}$ is the vector pointing from anchor i to the calculated position $\vec{x}$ of the relay for some iteration and

$$\vec{d_i} = \frac{\vec{r_i}}{||\vec{r_i}||} ||d_i||. \tag{4.2}$$

Note that $||d_i||$ is known and is the measured distance to anchor i but $\vec{d_i}$ is not.

The initial estimation was set to be the center of mass of the triangle formed by the three anchors. From there, the gradient descent method, explained in Section 2.1, was used to find the position that optimally minimizes Equation (4.1). This iteration process would continue until the residual was lower than some chosen threshold ($f(\vec{x}) < A$), where A is the threshold for the absolute residual, or the difference between the residuals of two subsequent iterations was lower than some other threshold ($f_{k+1}(\vec{x}) - f_k(\vec{x}) < R$), where R is the threshold for the relative residual, as this would mean that the residual does not significantly change between iterations. Figure 4.4 illustrates the use of gradient decent to find the minimum of the residual.

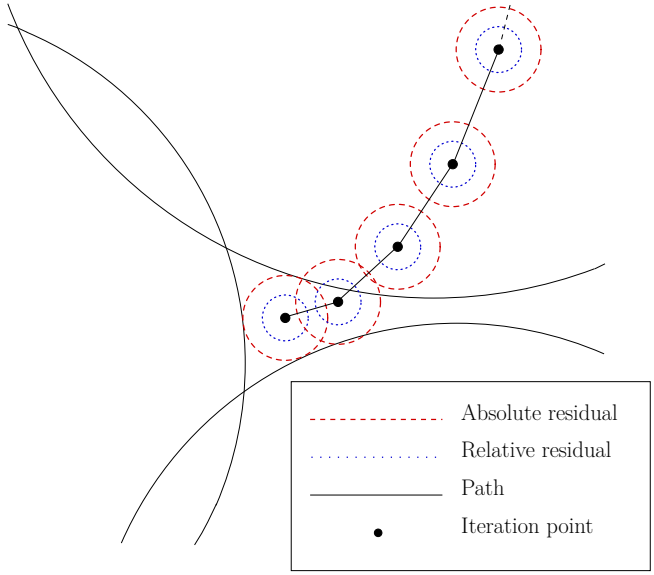


Figure 4.4: Illustration of using gradient decent to find the minimum of the residual. If all of the measured distances (black circles) are within an iteration point's red circle, the iteration process would exit since the iteration process is satisfied with the residual. If a iteration point has its precessing iteration point within the blue circle, relative residual, the iteration process would exit as well since the residual does not significantly change between iterations and it is satisfied as well.

Improving position estimation using extended Kalman filter

The extended Kalman filter used in this project is a set of equations that uses information about the previous state of the robot (position, orientation and velocities), the input controls sent to the relay computer (velocities and rotation), the measured position and the noise in all of the above to calculate an estimation of the state. For the position estimation the position and orientation part of the state is of main interest.

The state transition model depends only on the controls and not on the velocities in the previous state due to the velocities being set to what the the controls are (which are supplied to the ROSARIA node), as explained in Section 3.2.4.

Since the available motion controls were velocity, denoted X positive forward, and angular velocity, denoted Z positive counterclockwise, it was appropriate to build a state transition model explained in Section 2.3 around a circular motion, with radius $r = \frac{X}{Z}$, shown in Figure 4.5 and a lower bound for Z for which the motion would be considered linear below the lower bound. The model of the state at time instant $k-1$ is denoted $\vec{\hat{x}}_{k-1|k-1}$ and the actual state is denoted $\vec{x}_{k-1}$. The predicted new model of the state based on the state transition model is denoted $\vec{\hat{x}}_{k|k-1}$ between the time instant k and $k-1$. The corrected model of the state is then denoted $\vec{\hat{x}}_{k|k}$ after a correction of the predicted model of the state has been made. The actual new state is denoted $\vec{x}_k$.

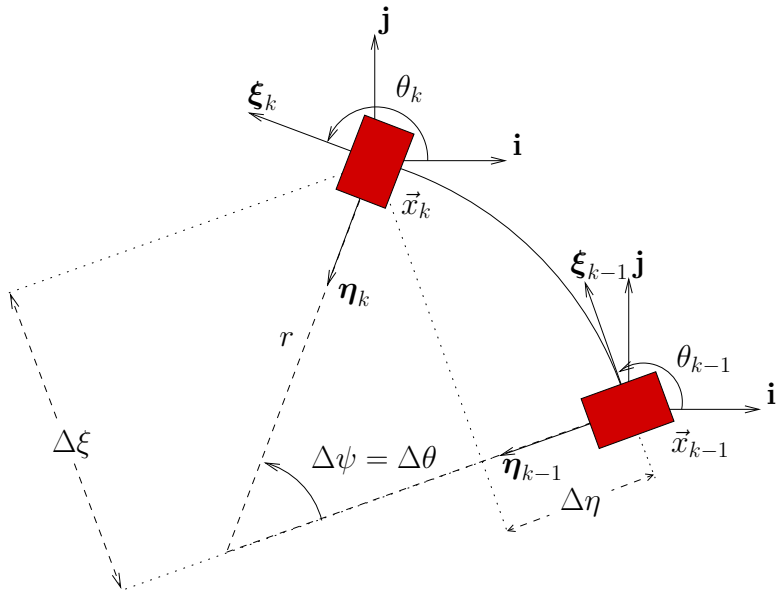


Figure 4.5: A robot executing its controls, at time instant $k-1$, causing its state to transition from $\vec{\hat{x}}_{k-1}$ to $\vec{\hat{x}}_k$. The **ij**-coordinate system has the same orientation as the reference coordinate system with its origin at the center of the robot. The $\boldsymbol{\xi\eta}$-coordinate system has the same origin as the **ij**-coordinate system but its orientation follows that of the robot.

The motion between the state at time instant $k-1$ and k for a circular motion can be seen in Figure 4.5. To simplify modeling this state transition, needed for the extended Kalman filter as described in Section 2.3, one can start by describing the state transition prediction in the relay's inertial frame of reference (the $\boldsymbol{\xi\eta}$-coordinate system) and then rotate that transition to the **ij**-coordinate system and finally translate that transition to the **xy**-coordinate system. The state transition prediction the $\boldsymbol{\xi\eta}$-coordinate system is

$$\vec{\hat{x}}^{\boldsymbol{\xi\eta}}_{k|k-1} = \begin{pmatrix} \hat{\xi}_{k|k-1} \\ \dot{\hat{\xi}}_{k|k-1} \\ \hat{\eta}_{k|k-1} \\ \dot{\hat{\eta}}_{k|k-1} \\ \hat{\psi}_{k|k-1} \\ \dot{\hat{\psi}}_{k|k-1} \end{pmatrix} = \begin{pmatrix} \frac{\hat{X}_{k-1}}{\hat{Z}_{k-1}} \sin \hat{Z}_{k-1}\Delta t \\ \hat{X}_{k-1} \\ \frac{\hat{X}_{k-1}}{\hat{Z}_{k-1}}(1 - \cos \hat{Z}_{k-1}\Delta t) \\ 0 \\ \hat{Z}_{k-1}\Delta t \\ \hat{Z}_{k-1} \end{pmatrix}. \tag{4.3}$$

The state transition prediction in the **xy**-coordinate system can be found by rotating the transition in Equation (4.3) in the $\boldsymbol{\xi\eta}$-coordinate system by an angle $-\theta_{k-1|k-1}$ (to get to the **ij**-coordinate system) and then translating the system by the position in the **xy**-coordinate system $(\hat{x}_{k-1|k-1}, \hat{y}_{k-1|k-1})^T$. This can be done using an operator $\hat{\boldsymbol{\Omega}}$ such that $\vec{\hat{x}}_{k|k-1} = \hat{\boldsymbol{\Omega}}\vec{\hat{x}}^{\boldsymbol{\xi\eta}}_{k|k-1}$. This operator is

$$\hat{\boldsymbol{\Omega}} = \begin{pmatrix} \hat{x}_{k-1|k-1} \\ 0 \\ \hat{x}_{k-1|k-1} \\ 0 \\ \theta_{k-1|k-1} \\ 0 \end{pmatrix} + \begin{pmatrix} \cos\theta_{k-1|k-1} & 0 & -\sin\theta_{k-1|k-1} & 0 & 0 & 0 \\ 0 & \cos\theta_{k-1|k-1} & 0 & -\sin\theta_{k-1|k-1} & 0 & 0 \\ \sin\theta_{k-1|k-1} & 0 & \cos\theta_{k-1|k-1} & 0 & 0 & 0 \\ 0 & \sin\theta_{k-1|k-1} & 0 & \cos\theta_{k-1|k-1} & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \tag{4.4}$$

After applying $\hat{\mathbf{\Omega}}$ to the state and using some trigonometric identities the state can be expressed as

$$\vec{\hat{x}}_{k|k-1} = \begin{pmatrix} \hat{x}_{k|k-1} \\ \hat{\dot{x}}_{k|k-1} \\ \hat{y}_{k|k-1} \\ \hat{\dot{y}}_{k|k-1} \\ \hat{\theta}_{k|k-1} \\ \hat{\dot{\theta}}_{k|k-1} \end{pmatrix} = \begin{pmatrix} \hat{x}_{k-1|k-1} + \frac{\hat{X}_{k-1}}{\hat{Z}_{k-1}}[\sin(\theta_{k-1} + \hat{Z}_{k-1}\Delta t) - \sin\theta_{k-1}] \\ \hat{X}_{k-1}\cos\theta_{k-1} \\ \hat{y}_{k-1|k-1} + \frac{\hat{X}_{k-1}}{\hat{Z}_{k-1}}[\cos\theta_{k-1} - \cos(\theta_{k-1} + \hat{Z}_{k-1}\Delta t)] \\ \hat{X}_{k-1}\sin\theta_{k-1} \\ \hat{\theta}_{k-1} + \hat{Z}_{k-1}\Delta t \\ \hat{Z}_{k-1} \end{pmatrix}. \tag{4.5}$$

For a linear motion (no rotation) thus $\hat{Z}_{k-1} = 0$ the state transition prediction is simpler and does not need a simplification by looking at it in a different coordinate-system. It can be modeled as

$$\vec{\hat{x}}_{k|k-1} = \begin{pmatrix} \hat{x}_{k|k-1} \\ \hat{\dot{x}}_{k|k-1} \\ \hat{y}_{k|k-1} \\ \hat{\dot{y}}_{k|k-1} \\ \hat{\theta}_{k|k-1} \\ \hat{\dot{\theta}}_{k|k-1} \end{pmatrix} = \begin{pmatrix} \hat{x}_{k-1|k-1} + \hat{X}_{k-1}\cos\theta_{k-1|k-1}\Delta t \\ \hat{X}_{k-1}\cos\theta_{k-1|k-1} \\ \hat{y}_{k-1|k-1} + \hat{X}_{k-1}\sin\theta_{k-1|k-1}\Delta t \\ \hat{X}_{k-1}\sin\theta_{k-1|k-1} \\ \hat{\theta}_{k} \\ 0 \end{pmatrix}. \tag{4.6}$$

The actual state transition can be found by replacing the input controls with the executed controls (in Equation (4.5) and (4.6)), which can be found by applying a noise to the input controls. The control noise $w_{k-1} = (w_{X,k-1}, w_{Z,k-1})^T$ at time instant k was assumed to vary linearly with the controls $\hat{X}$ and $\hat{Z}$ sent to the robots (input controls). The actual controls executed by the robots would then be $X_k = \hat{X}_{k-1}(1 + w_{X,k-1})$ and $Z_k = \hat{Z}_{k-1}(1 + w_{Z,k-1})$.
As explained in Section 2.3, to find $\mathbf{P}_{k-1|k-1}$ the matrices $\mathbf{F}_{k-1}$ and $\mathbf{L}_{k-1}$ are needed. For the circular motion, $\mathbf{F}_{k-1}$ was

$$\mathbf{F}_{k-1} = \begin{pmatrix} \frac{\partial \hat{x}_{k|k-1}}{\partial \hat{x}_{k-1|k-1}} & \frac{\partial \hat{x}_{k|k-1}}{\partial \hat{\dot{x}}_{k-1|k-1}} & \frac{\partial \hat{x}_{k|k-1}}{\partial \hat{y}_{k-1|k-1}} & \frac{\partial \hat{x}_{k|k-1}}{\partial \hat{\dot{y}}_{k-1|k-1}} & \frac{\partial \hat{x}_{k|k-1}}{\partial \hat{\theta}_{k-1|k-1}} & \frac{\partial \hat{x}_{k|k-1}}{\partial \hat{\dot{\theta}}_{k-1|k-1}} \\ 0 & \frac{\partial \hat{\dot{x}}_{k|k-1}}{\partial \hat{\dot{x}}_{k-1|k-1}} & 0 & 0 & \frac{\partial \hat{\dot{x}}_{k|k-1}}{\partial \hat{\theta}_{k-1|k-1}} & 0 \\ \frac{\partial \hat{y}_{k|k-1}}{\partial \hat{x}_{k-1|k-1}} & \frac{\partial \hat{y}_{k|k-1}}{\partial \hat{\dot{x}}_{k-1|k-1}} & \frac{\partial \hat{y}_{k|k-1}}{\partial \hat{y}_{k-1|k-1}} & \frac{\partial \hat{y}_{k|k-1}}{\partial \hat{\dot{y}}_{k-1|k-1}} & \frac{\partial \hat{y}_{k|k-1}}{\partial \hat{\theta}_{k-1|k-1}} & \frac{\partial \hat{y}_{k|k-1}}{\partial \hat{\dot{\theta}}_{k-1|k-1}} \\ 0 & 0 & 0 & \frac{\partial \hat{\dot{y}}_{k|k-1}}{\partial \hat{\dot{y}}_{k-1|k-1}} & \frac{\partial \hat{\dot{y}}_{k|k-1}}{\partial \hat{\theta}_{k-1|k-1}} & 0 \\ 0 & 0 & 0 & 0 & \frac{\partial \hat{\theta}_{k|k-1}}{\partial \hat{\theta}_{k-1|k-1}} & \frac{\partial \hat{\theta}_{k|k-1}}{\partial \hat{\dot{\theta}}_{k-1|k-1}} \\ 0 & 0 & 0 & 0 & 0 & \frac{\partial \hat{\dot{\theta}}_{k|k-1}}{\partial \hat{\dot{\theta}}_{k-1|k-1}} \end{pmatrix} \tag{4.7}$$

where the non-zero matrix elements are as presented in Equation (4.8)

$$
\begin{aligned}
\frac{\partial \hat{x}_{k|k-1}}{\partial \hat{x}_{k-1|k-1}} &= 1 \\
\frac{\partial \hat{x}_{k|k-1}}{\partial \hat{\dot{x}}_{k-1|k-1}} &= \frac{\sin \hat{Z}_{k-1}\Delta t}{\hat{Z}_{k-1}} \\
\frac{\partial \hat{x}_{k|k-1}}{\partial \hat{y}_{k-1|k-1}} &= 0 \\
\frac{\partial \hat{x}_{k|k-1}}{\partial \hat{\dot{y}}_{k-1|k-1}} &= -\frac{1-\cos \hat{Z}_{k-1}\Delta t}{\hat{Z}_{k-1}} \\
\frac{\partial \hat{x}_{k|k-1}}{\partial \hat{\theta}_{k-1|k-1}} &= \frac{\hat{X}_{k-1}}{\hat{Z}_{k-1}} \cos\left(\theta_{k-1|k-1} + \hat{Z}_{k-1}\Delta t\right) - \frac{\hat{X}_{k-1}}{\hat{Z}_{k-1}} \cos\theta_{k-1|k-1} \\
\frac{\partial \hat{x}_{k|k-1}}{\partial \hat{\dot{\theta}}_{k-1|k-1}} &= -\frac{\hat{X}_{k-1}}{\hat{Z}_{k}^2} \sin\left(\theta_{k-1|k-1} + \hat{Z}_{k-1}\Delta t\right) + \frac{\hat{X}_{k-1}}{\hat{Z}_{k-1}} \cos\theta_{k-1|k-1} + \hat{Z}_{k-1}\Delta t + \frac{\hat{X}_{k-1}}{\hat{Z}_{k}^2} \sin\theta_{k-1|k-1} \\
\frac{\partial \hat{\dot{x}}_{k|k-1}}{\partial \hat{\dot{x}}_{k-1|k-1}} &= 1 \\
\frac{\partial \hat{\dot{x}}_{k|k-1}}{\partial \hat{\theta}_{k-1|k-1}} &= -\hat{X}_{k-1} \sin\theta_{k-1|k-1} \\
\frac{\partial \hat{y}_{k|k-1}}{\partial \hat{x}_{k-1|k-1}} &= 0 \\
\frac{\partial \hat{y}_{k|k-1}}{\partial \hat{\dot{x}}_{k-1|k-1}} &= \frac{1-\cos \hat{Z}_{k-1}\Delta t}{\hat{Z}_{k-1}} \\
\frac{\partial \hat{y}_{k|k-1}}{\partial \hat{y}_{k-1|k-1}} &= 1 \\
\frac{\partial \hat{y}_{k|k-1}}{\partial \hat{\dot{y}}_{k-1|k-1}} &= \frac{\sin \hat{Z}_{k-1}\Delta t}{\hat{Z}_{k-1}} \\
\frac{\partial \hat{y}_{k|k-1}}{\partial \hat{\theta}_{k-1|k-1}} &= \frac{\hat{X}_{k-1}}{\hat{Z}_{k-1}} \sin\left(\theta_{k-1|k-1} + \hat{Z}_{k-1}\Delta t\right) - \frac{\hat{X}_{k-1}}{\hat{Z}_{k-1}} \sin\theta_{k-1|k-1} \\
\frac{\partial \hat{y}_{k|k-1}}{\partial \hat{\dot{\theta}}_{k-1|k-1}} &= \frac{\hat{X}_{k-1}}{\hat{Z}_{k}^2} \cos\left(\theta_{k-1|k-1} + \hat{Z}_{k-1}\Delta t\right) + \frac{\hat{X}_{k-1}}{\hat{Z}_{k-1}} \sin\left(\theta_{k-1|k-1} + \hat{Z}_{k-1}\Delta t\right) - \frac{\hat{X}_{k-1}}{\hat{Z}_{k}^2} \cos\theta_{k-1|k-1} \\
\frac{\partial \hat{\dot{y}}_{k|k-1}}{\partial \hat{\dot{y}}_{k-1|k-1}} &= 1 \\
\frac{\partial \hat{\dot{y}}_{k|k-1}}{\partial \hat{\theta}_{k-1|k-1}} &= \hat{X}_{k-1} \cos\theta_{k-1|k-1} \\
\frac{\partial \hat{\theta}_{k|k-1}}{\partial \hat{\theta}_{k-1|k-1}} &= 1 \\
\frac{\partial \hat{\theta}_{k|k-1}}{\partial \hat{\dot{\theta}}_{k-1|k-1}} &= \Delta t \\
\frac{\partial \hat{\dot{\theta}}_{k|k-1}}{\partial \hat{\theta}_{k-1|k-1}} &= 1
\end{aligned}
\tag{4.8}
$$

and for linear motion it was

$$
\mathbf{F}_{k-1} = \begin{pmatrix}
\frac{\partial \hat{x}_{k|k-1}}{\partial \hat{x}_{k-1|k-1}} & \frac{\partial \hat{x}_{k|k-1}}{\partial \hat{\dot{x}}_{k-1|k-1}} & 0 & 0 & \frac{\partial \hat{x}_{k|k-1}}{\partial \hat{\theta}_{k-1|k-1}} & 0 \\
0 & \frac{\partial \hat{\dot{x}}_{k|k-1}}{\partial \hat{\dot{x}}_{k-1|k-1}} & 0 & 0 & \frac{\partial \hat{\dot{x}}_{k|k-1}}{\partial \hat{\theta}_{k-1|k-1}} & 0 \\
0 & 0 & \frac{\partial \hat{y}_{k|k-1}}{\partial \hat{y}_{k-1|k-1}} & \frac{\partial \hat{y}_{k|k-1}}{\partial \hat{\dot{y}}_{k-1|k-1}} & \frac{\partial \hat{y}_{k|k-1}}{\partial \hat{\theta}_{k-1|k-1}} & 0 \\
0 & 0 & 0 & \frac{\partial \hat{\dot{y}}_{k|k-1}}{\partial \hat{\dot{y}}_{k-1|k-1}} & \frac{\partial \hat{\dot{y}}_{k|k-1}}{\partial \hat{\theta}_{k-1|k-1}} & 0 \\
0 & 0 & 0 & 0 & \frac{\partial \hat{\theta}_{k|k-1}}{\partial \hat{\theta}_{k-1|k-1}} & 0 \\
0 & 0 & 0 & 0 & 0 & 0
\end{pmatrix}
\tag{4.9}
$$

where the non-zero matrix elements are as presented in Equation (4.10)

$$\begin{aligned}
\frac{\partial \hat{x}_{k|k-1}}{\partial \hat{x}_{k-1|k-1}} &= 1 \\
\frac{\partial \hat{x}_{k|k-1}}{\partial \hat{\dot{x}}_{k-1|k-1}} &= \Delta t \\
\frac{\partial \hat{x}_{k|k-1}}{\partial \hat{\theta}_{k-1|k-1}} &= -\hat{X}_{k-1} \sin \theta_{k-1|k-1} \Delta t \\
\frac{\partial \hat{\dot{x}}_{k|k-1}}{\partial \hat{\dot{x}}_{k-1|k-1}} &= 1 \\
\frac{\partial \hat{\dot{x}}_{k|k-1}}{\partial \hat{\theta}_{k-1|k-1}} &= -\hat{X}_{k-1} \sin \theta_{k-1|k-1} \\
\frac{\partial \hat{y}_{k|k-1}}{\partial \hat{y}_{k-1|k-1}} &= 1 \\
\frac{\partial \hat{y}_{k|k-1}}{\partial \hat{\dot{y}}_{k-1|k-1}} &= \Delta t \\
\frac{\partial \hat{y}_{k|k-1}}{\partial \hat{\theta}_{k-1|k-1}} &= \hat{X}_{k-1} \cos \theta_{k-1|k-1} \Delta t \\
\frac{\partial \hat{\dot{y}}_{k|k-1}}{\partial \hat{\dot{y}}_{k-1|k-1}} &= 1 \\
\frac{\partial \hat{\dot{y}}_{k|k-1}}{\partial \hat{\theta}_{k-1|k-1}} &= \hat{X}_{k-1} \cos \theta_{k-1|k-1} \\
\frac{\partial \hat{\theta}_{k|k-1}}{\partial \hat{\theta}_{k-1|k-1}} &= 1.
\end{aligned} \tag{4.10}$$

For the circular motion, $\mathbf{L}_{k-1}$ was (under the assumption that the noise scaled linearly with $\hat{Z}$ and $\hat{X}$)

$$\mathbf{L}_{k-1} = \begin{pmatrix}
\frac{\partial \hat{x}_{k|k-1}}{\partial w_{X,k-1}} & \frac{\partial \hat{x}_{k|k-1}}{\partial w_{Z,k-1}} & 0 & 0 & 0 & 0 \\
\frac{\partial \hat{\dot{x}}_{k|k-1}}{\partial w_{X,k-1}} & 0 & 0 & 0 & 0 & 0 \\
\frac{\partial \hat{y}_{k|k-1}}{\partial w_{X,k-1}} & \frac{\partial \hat{y}_{k|k-1}}{\partial w_{Z,k-1}} & 0 & 0 & 0 & 0 \\
\frac{\partial \hat{\dot{y}}_{k|k-1}}{\partial w_{X,k-1}} & 0 & 0 & 0 & 0 & 0 \\
0 & \frac{\partial \hat{\theta}_{k|k-1}}{\partial w_{Z,k-1}} & 0 & 0 & 0 & 0 \\
0 & \frac{\partial \hat{\dot{\theta}}_{k|k-1}}{\partial w_{Z,k-1}} & 0 & 0 & 0 & 0
\end{pmatrix} \tag{4.11}$$

where the non-zero matrix elements are as presented in Equation (4.12)

$$\begin{aligned}
\frac{\partial \hat{x}_{k|k-1}}{\partial w_{X,k-1}} &= \frac{\hat{X}_{k-1}}{\hat{Z}_{k-1}} (\sin(\theta_{k-1|k-1} + \hat{Z}_{k-1}\Delta t) - \sin \theta_{k-1|k-1}) \\
\frac{\partial \hat{x}_{k|k-1}}{\partial w_{Z,k-1}} &= \hat{X}_{k-1} (\cos(\theta_{k-1|k-1} + \hat{Z}_{k-1}\Delta t)\Delta t - \frac{\sin(\theta_{k-1|k-1} + \hat{Z}_{k-1}\Delta t) - \sin \theta_{k-1|k-1}}{\hat{Z}_{k-1}}) \\
\frac{\partial \hat{\dot{x}}_{k|k-1}}{\partial w_{X,k-1}} &= \hat{X}_{k-1} \cos \theta_{k-1|k-1} \\
\frac{\partial \hat{y}_{k|k-1}}{\partial w_{X,k-1}} &= \frac{\hat{X}_{k-1}}{\hat{Z}_{k-1}} (\cos \theta_{k-1|k-1} - \cos(\theta_{k-1|k-1} + \hat{Z}_{k-1}\Delta t)) \\
\frac{\partial \hat{y}_{k|k-1}}{\partial w_{Z,k-1}} &= \hat{X}_{k-1} (\sin(\theta_{k-1|k-1} + \hat{Z}_{k-1}\Delta t)\Delta t + \frac{\cos(\theta_{k-1|k-1} + \hat{Z}_{k-1}\Delta t) - \cos \theta_{k-1|k-1}}{\hat{Z}_{k-1}}) \\
\frac{\partial \hat{\dot{y}}_{k|k-1}}{\partial w_{X,k-1}} &= \hat{X}_{k-1} \sin \theta_{k-1|k-1} \\
\frac{\partial \hat{\theta}_{k|k-1}}{\partial w_{Z,k-1}} &= \hat{Z}_{k-1}\Delta t \\
\frac{\partial \hat{\dot{\theta}}_{k|k-1}}{\partial w_{Z,k-1}} &= \hat{Z}_{k-1}
\end{aligned} \tag{4.12}$$

and for linear motion it was

$$\mathbf{L}_{k-1} = \begin{pmatrix}
\frac{\partial \hat{x}_{k|k-1}}{\partial w_{X,k-1}} & 0 & 0 & 0 & 0 & 0 \\
\frac{\partial \hat{\dot{x}}_{k|k-1}}{\partial w_{X,k-1}} & 0 & 0 & 0 & 0 & 0 \\
\frac{\partial \hat{y}_{k|k-1}}{\partial w_{X,k-1}} & 0 & 0 & 0 & 0 & 0 \\
\frac{\partial \hat{\dot{y}}_{k|k-1}}{\partial w_{X,k-1}} & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0
\end{pmatrix} \tag{4.13}$$

where the non-zero matrix elements are as presented in Equation (4.14)

$$\begin{array}{rcl} \frac{\partial \hat{x}_{k|k-1}}{\partial w_{X,k-1}} & = & \hat{X}_{k-1} \cos\theta_{k-1|k-1}\Delta t \\ \frac{\partial \hat{x}_{k|k-1}}{\partial w_{X,k-1}} & = & \hat{X}_{k-1} \cos\theta_{k-1|k-1} \\ \frac{\partial \hat{y}_{k|k-1}}{\partial w_{X,k-1}} & = & \hat{X}_{k-1} \sin\theta_{k-1|k-1}\Delta t \\ \frac{\partial \hat{y}_{k|k-1}}{\partial w_{X,k-1}} & = & \hat{X}_{k-1} \sin\theta_{k-1|k-1}. \end{array} \tag{4.14}$$

With $\mathbf{L}_{k-1}$ as presented in Equation (4.11) and (4.13) it is necessary that the control noise covariance matrix should be

$$\mathbf{Q}_{k-1} = \begin{pmatrix} \sigma^2_{X,k-1} & 0 & 0 & 0 & 0 & 0 \\ 0 & \sigma^2_{Z,k-1} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}. \tag{4.15}$$

as explained in Section 2.3. The only observable (measurable) variables in the state was the position as explained in the Section 4.1.2. This observation was assumed to have the noise $w_{\text{meas},k}$ with standard deviation σ_{meas}.h As such the observation was

$$\mathbf{z}_k = \begin{pmatrix} x_k + w_{\text{meas},k} \\ 0 \\ y_k + w_{\text{meas},k} \\ 0 \\ 0 \\ 0 \end{pmatrix}. \tag{4.16}$$

This makes the observation matrix

$$\mathbf{H}_k = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}, \tag{4.17}$$

and the observation noise covariance matrix

$$\mathbf{R}_k = \sigma^2_{\text{meas},k}\mathbf{I}. \tag{4.18}$$

This concludes all of the necessary matrices applied to the extended Kalman filter on a state. Additionally, some standard deviation of the control noises σ_X and σ_Z was assumed as well as some standard deviation of the measurement noise σ_{meas}. To use the filter, the initial state $\vec{\hat{X}}_{0|0}$ and state covariance $\mathbf{P}_{0|0}$ had to be approximated.

4.2 Movement algorithms for robots

Using *twist* messages as basic movement operations, explained in Section 3.2.4, two kinds of movement algorithms were developed. These are explained in the following sections: 4.2.1 *Moving a relay robot to a target in sequential steps* and 4.2.2 *Moving robots to a target in a smooth motion.* In this case,

moving the relay in sequential steps means that the relay travels a fixed distance forward, stops and then rotates. The sequence is then repeated until the relay is deemed close enough to its target, at which point the algorithm stops all movement. What is meant with *smooth movement* is a relay moving forward and turning at the same time, so that no interruptions occur in the motion until target destination is reached.

4.2.1 Moving a relay robot to a target in sequential steps

In order to move a relay to a target, a sequential movement algorithm was implemented, which is called once with a parameter that specifies the target position. It works by first measuring the relay's position using the RCMs and trilateration explained in Section 4.1.2. The relay is then moved forwards 10 cm, where a new position measurement is made. Using the current, previous and target position, the direction of the relay as well as how many degrees it should turn to face the target is then calculated. It was not possible to ensure that the the direction of the relay robot was perfectly aligned to the target after rotating, due to measurement noise and process noise, whereas the direction was recalibrated every 10 cm, thus producing the sequential movement. When the distance from relay to target is less than 10 cm, the relay is moved forwards the remainder of the distance. An example of a sequential movement in four steps can be seen in Figure 4.6.

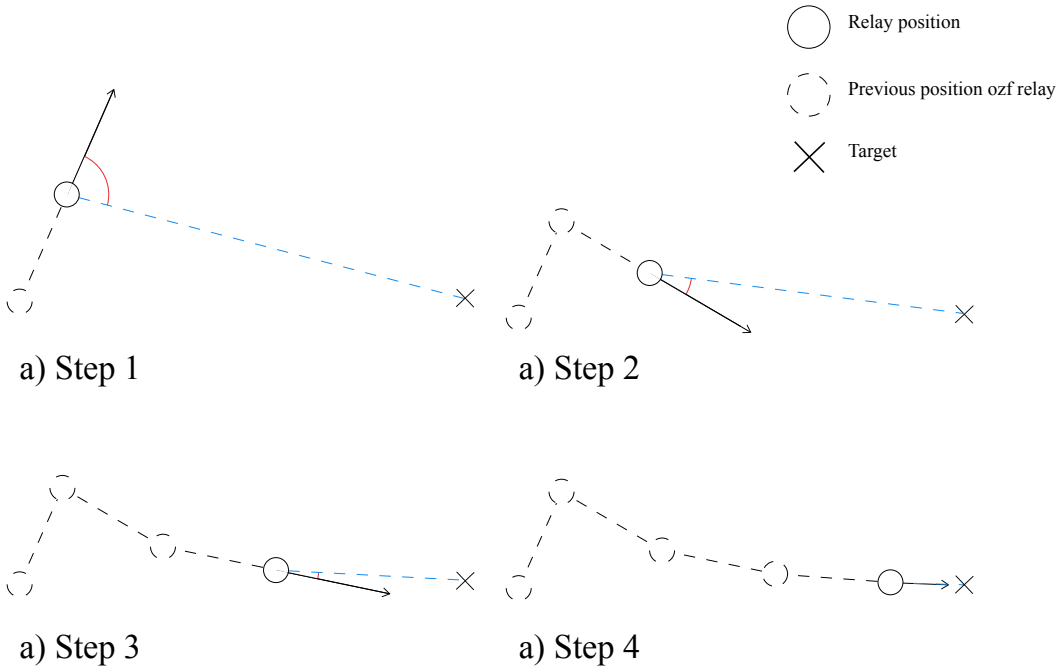


Figure 4.6: Four different steps of the sequential movement algorithm where the previous position is marked by a dashed circle, the current position by a circle and the target position by a cross. The angle depicted in red shows the angle which the relay robot has to turn in order to face the target and the shortest distance from current position to target is depicted by the dashed blue line. This is a simulated example of how a sequential movement could be executed.

The previous position of the relay $\vec{p_0} = (x_0, y_0)$, the current position $\vec{p_1} = (x_1, y_1)$ and the target $\vec{p_t} = (x_t, y_t)$. The shortest distance from current position to target and is expressed as $(||\vec{p_t} - \vec{p_1}||)$. The angle which the relay robot has to turn in order to face the target is calculated by comparing the length to target from the initial position, with the length to target from the current position resulting in the expression

$$\theta = \arccos\left(\frac{(\vec{p_t} - \vec{p_0}) \cdot (\vec{p_t} - \vec{p_1})}{||\vec{p_t} - \vec{p_0}|| \cdot ||\vec{p_t} - \vec{p_1}||}\right). \tag{4.19}$$

The rotation direction is then calculated by first checking if the initial position was within the second or the fourth quadrant in the frame of reference, then compare the gradient of the linear equation between the initial position $\vec{p_0} = (x_0, y_0)$ and the target position $\vec{p_t} = (x_t, y_t)$ (denoted k_{targ}) with the gradient of the linear equation between the initial position $\vec{p_0} = (x_0, y_0)$ and the current position $\vec{p_1} = (x_1, y_1)$ (denoted k_{move}). If k_{move} is larger or equal to k_{targ} the direction to turn is negative, otherwise positive. But if the initial position was in the first or the third quadrant (in the frame of reference) it is the opposite. The actual angle sent to the relay is θ from Equation 4.19 multiplied with the direction.

To accompany the sequential movement algorithm, two ROS services were implemented for the relay: MoveForward (Distance) and Rotate (Radians), with parameters for specifying the distance to travel or radians to rotate. As explained in Section 3.2.4, the ROSARIA node, the `linear.x` parameter of a *twist* message determines the forward movement in m/s and the `angular.z` parameter determines the rotation in rad/s. Moving forwards a set distance was accomplished by sending a *twist* message with `linear.x` set to the specified distance, divided by n. A sleep function in the rospy library was then called to sleep the execution of the service for n seconds before resetting the *twist* message, resulting in a movement of specified distance. The n parameter can be changed to produce desired results. The same technique was applied for the rotate service.

4.2.2 Moving robots to a target in a smooth motion

To move a relay to a target position without interruptions, an algorithm for smooth movement was developed. To accomplish smooth movement, the robot had to be moved forwards and turn at the same time, whereby both the `linear.x` and `angular.z` parameters of a *twist* message had to be set. An example of such a movement is shown in Figure 4.7. The smooth movement algorithm takes three parameters; current position, target and direction of relay, and works by calculating and returning a *twist* message that aligns the relay robot's trajectory towards the target position while moving forwards. The returned *twist* message is then sent to the relay robot using a service which updates the *twist* message to the ROSARIA node. Because the *twist* message is based on parameters that continuously change, it has to be called repeatedly so that the relay adjusts its path accordingly. Calling the algorithm repeatedly also means that it is possible to specify a new target for each call. How the controls were calculated is explained in *Calculating controls*.

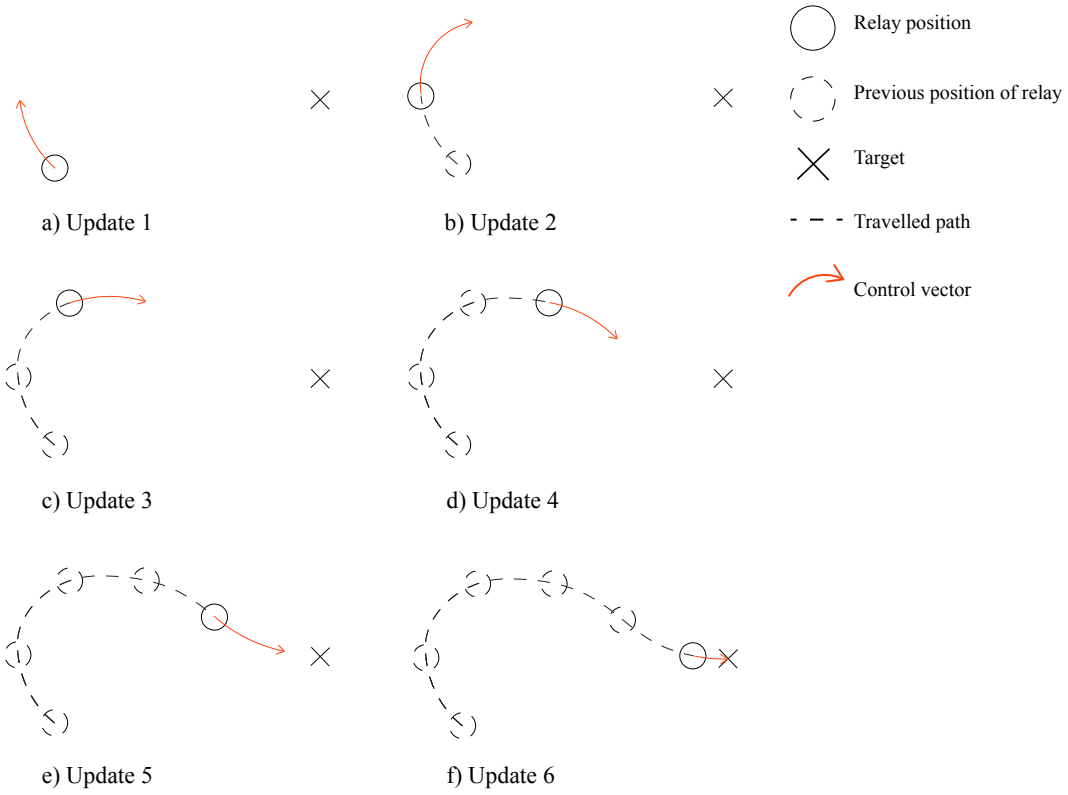


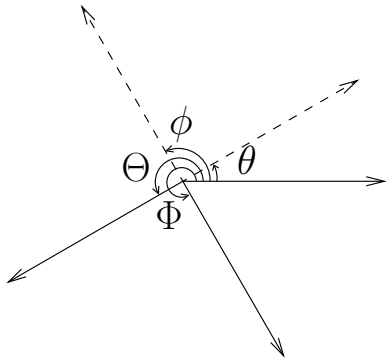
Figure 4.7: The picture shows the path of a relay robot that uses the smooth movement algorithm. The red movement vector represents the controls that the relay robot is currently running, and the previous positions of relay show points at which the controls were updated. Note that the picture does not accurately represent the frequency of the control updates, whereas in practice the updates occur around two times per second.

Calculating controls

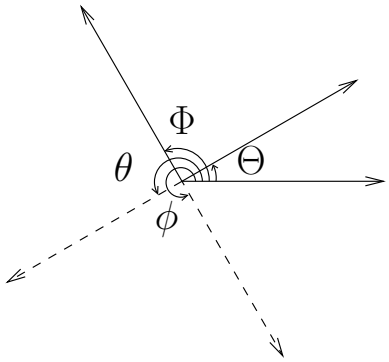
The algorithm for generating controls for the robot is comprised of three parts: calculating velocity magnitude (`linear.x`), rotation magnitude (`angular.z`), and rotation direction.

The rotation direction is computed by calculating the angle between the relay's direction to its target position and heading direction. A positive angle yields negative (clockwise) direction while negative angle yields positive (counterclockwise) direction. In order to calculate the magnitude of the velocity and rotation, the distance and angle to the robot's target position was calculated and divided by some arbitrary time (which should be larger than the time between time k and $k+1$, Δt) to get a magnitude of the velocity and orientation. To make the expressions look cleaner, the distance to the target position was denoted

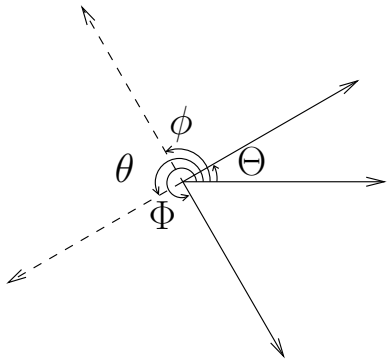
$$\vec{l} = (l_x, l_y)^T = (\vec{p}_{\text{current}} - \vec{p}_{\text{target}}) \tag{4.20}$$



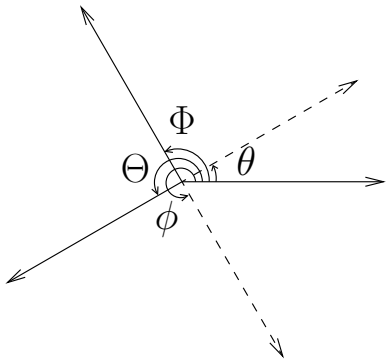
(a) CCW rotation direction. sgn(`angular.z`) = 1.



(b) CCW rotation direction. sgn(`angular.z`) = 1.



(c) CW rotation direction. sgn(`angular.z`) = −1.



(d) CW rotation direction. sgn(`angular.z`) = −1.

Figure 4.8: The different scenarios used to calculate the rotation direction (clockwise or counter-clockwise). θ is the orientation of the robot, ϕ is the angle to the target position. Θ and Φ are as seen in Equation (4.22).

The direction was determined so that the robot would always rotate in the direction that had the smallest angle to the target position. The direction was determined by Equation (4.23) using the variables from Equation (4.21) and (4.22).

$$\phi = \begin{cases} \arccos \frac{\vec{l} \cdot (1,0)^T}{||\vec{l}||} & l_y \geq 0 \\ 2 - \pi \arccos \frac{\vec{l} \cdot (1,0)^T}{||\vec{l}||} & l_y < 0 \end{cases} \tag{4.21}$$

$$\begin{aligned} \Phi &= \text{mod}(\phi + \pi, 2\pi) \\ \Theta &= \text{mod}(\theta + \pi, 2\pi) \end{aligned} \tag{4.22}$$

$$\text{sgn}(\texttt{angular.z}) = \begin{cases} 1 & 0 \leq \phi \leq \pi \wedge \phi \leq \Theta < \Phi \text{ , Figure 4.8a} \\ 1 & \pi < \phi < 2\pi \wedge \Phi \leq \theta < \phi \text{ , Figure 4.8b} \\ -1 & 0 \leq \phi \leq \pi \wedge \phi \leq \theta < \Phi \text{ , Figure 4.8c} \\ -1 & \pi < \phi < 2\pi \wedge \Phi \leq \Theta < \phi \text{ , Figure 4.8d} \end{cases} \tag{4.23}$$

Equation (4.23) can be easier to grasp in conjunction with Figure 4.8.

The magnitude of the angular velocity was determined by Equation (4.27) using the variables from

Equation (4.24), (4.25) and (4.26).

$$\begin{pmatrix}\alpha\\ \beta\end{pmatrix} = \begin{pmatrix}\cos\theta & \sin\theta\\ -\sin\theta & \cos\theta\end{pmatrix}^{-1} \cdot \vec{l} \tag{4.24}$$

$$\vec{d} = (\alpha\cos\theta, \alpha\sin\theta)^T \tag{4.25}$$

$$\phi = \begin{cases}\arccos\frac{\vec{l}\cdot\vec{d}}{||\vec{l}||\cdot||\vec{d}||} & \alpha \geq 0\\ 2\pi - \arccos\frac{\vec{l}\cdot\vec{d}}{||\vec{l}||\cdot||\vec{d}||} & \alpha < 0\end{cases} \tag{4.26}$$

$$\texttt{angular.z} = \frac{\phi}{T_Z} \tag{4.27}$$

The magnitude of the linear velocity was set to simply be the quotient of the distance to the target position divided by some arbitrary time constant T_X. The factor $\frac{\pi-\phi}{\pi}$ was included as it would counteract the purpose if the robot moves at full speed even if the robot is facing away from its target position. The expression for the linear velocity is

$$\texttt{linear.x} = \frac{\vec{l}}{T_X}\frac{\pi-\phi}{\pi}. \tag{4.28}$$

4.3 Coordinating robots

This section will describe two methods that were developed for coordinating a number of relays by minimizing some objective function to retrieve targets for the relays. The functions that are used are simplifications of a model that determines how the best signal quality can be achieved, given that a signal is transmitted between relays. These will be further explained later in this section.

The first method coordinates the relays through an iterative process that makes use of the sequential movement algorithm explained in Section 4.2.1 and position estimation using RCMs and trilateration explained in Section 4.1.2. The second method is more extensive and does this simultaneously for all relays using the smooth movement algorithm explained in Section 4.2.2, a start-up sequence, gradient descent explained in Section 2.1, a Kalman filter explained in Section 4.1.2 as well as a collision avoidance algorithm. These two coordination methods are described in the following Section 4.3.1.

4.3.1 Defining objective functions

Normally, in order to achieve maximum data transfer, the received signal should be as similar as possible to the transmitted signal (the received signal should be as little distorted as possible). Because the signal is an electromagnetic wave, the causes of these distortions are scattering, chirp, reflection and diffraction to mention a few. Another source of distortion is that the power flux density, i.e the transmitted power density, decreases with the distance from the transmitter, thus making it more difficult to distinguish the signal from background noise (such as thermal noise). All of the above mentioned distortions become more prominent as the distance to the transmitter increases and the reflection as well as the diffraction becomes a problem when obstacles are blocking the signal [25].

Given that the equipment has physical properties such as antenna transmit power and antenna gain, that can not be altered, the maximum data transfer will be achieved by minimizing the distance while

the robots have their neighboring relays in line of sight. The optimal position is calculated using gradient descent or ascent on a target function describing some parameter or parameters that should be minimized if using gradient descent or maximized if using gradient ascent. One parameter that can be used as a basis for the target function is the end-to-end bit error rate (BER). The function is then built around the probability that the information sent is altered and, in most cases, lost. The result of this for m nodes (meaning $m-2$ relaying nodes) is according to [26]. A general form of this expression is

$$f(\vec{\rho},\vec{\phi}) = \sum_{i=2}^{m} \log(1 - a\exp(-b_{i,i-1}(\rho_{i,i-1},\phi_{i,i-1})\rho_{i,i-1}^{-n})) \tag{4.29}$$

where a $(0 < a < 1)$ is a parameter that has to do with probability of a bit being flipped, $b_{i,i-1}$ is a function of system parameters for the connection between node i and $i-1$, $\rho_{i,i-1}$ is the distance between nodes i and $i-1$, $\phi_{i,i-1}$ is the difference in orientation of nodes i and $i-1$ and n is the path-loss exponent. The gradient of this function may be considerably complicated to evaluate, especially when remembering that $b_{i,i-1}$ is a function of system parameters. Since all of the Wi-Fi USB adapters used are the same for all robots, the function of the system parameters should be the same between all of the robots. Clever positioning and orientation of the Wi-Fi USB adapters can justify the approximation that the BER function is radially symmetric which removes the $\phi_{i,i-1}$ dependency. In this project, the system parameters are never measured, and are unlikely to be affected due to the limited testing area, whereas they can safely be considered to be constant. With these approximations in place, all that remains of Equation (4.29) is a function stating that the bit error rate may be minimized by minimizing the distance between the nodes. With this in mind, minimizing or maximizing a more simple function can be used to accomplish the very same goal. An example of a simple function that should be minimized is

$$f(\vec{\rho}) = \sum_{i=2}^{m} c\rho_{i,i-1}^{2} \tag{4.30}$$

where c is some constant and $\rho_{i,i-1}$ is the distance between nodes i and $i-1$.

Given the simplifications mentioned above, the gradient of Equation (4.30) is parallel with the gradient of Equation (4.29), but the magnitude is different. This means than with clever use of gradient descent, both target functions will yield very similar controls and the end result will be the same - the optimal robot configuration for maximum data transfer.

The iterative coordination method uses a simplified version of this, by only calculating the middle point between its neighbours instead of minimizing the Equation (4.30) which is done for the simultaneously coordination method. The simplified version works by generating a target for a relay, determined by the middle point of that relay's neighbors positions ($\vec{p}_{\text{left}} = (x_l, y_l)^T$, $\vec{p}_{\text{right}} = (x_r, y_r)^T$) shown in Equation (4.31). Thus using the middle point as the optimal placement instead of minimizing Equation (4.30).

$$\vec{p}_{\text{target}} = \frac{\vec{p}_{\text{left}} + \vec{p}_{\text{right}}}{2} \tag{4.31}$$

4.3.2 Coordinating relays iteratively with sequential movement

This section describes the first algorithm that was developed for coordinating the relay positions, called the iterative coordination algorithm. The relay is moved to the target position given by Equation (4.30), using the sequential movement algorithm. This loop is iterated for every other relay until the optimal placement for all of the relays has been reached. It was discovered that if more than one RCMs is used at a time, the measured values will be faulty due to interference and since the sequential movement

algorithm uses RCM-measurements several times during run time, the relay robots were optimized iteratively.

4.3.3 Coordinating relays simultaneously with smooth movement

The second coordination algorithm that was developed moves all relay robots towards a position configuration simultaneously using the smooth movement algorithm explained in Section 4.2.2. This required the possibility to determine the positions of several relays simultaneously, and as described in the previous section, only one relay could request measurements from the RCMs at a time, which was solved by measuring the coordinates of one relay and determining the coordinates of the other relays with the prediction part of an extended Kalman filter, described in Section 4.1.2. To use the Kalman filter for prediction it is necessary to know the orientation of the relays, which is calculated through a start-up sequence. The startup sequence is further described later in this section. After the position has either been predicted or corrected by actual measurements, the target position is calculated using Equation (4.31). Using the current position, direction and target of each relay robot, the smooth movement algorithm was then used to generate corresponding controls (*twist* messages) as explained in Section 4.2.2. The controls were then passed to a collision avoidance algorithm, explained in Section 4.3.3 below, which may alter the controls to ensure no collision, before sending them to the relay robot. This process was then run repeatedly, with a time constant to ensure a frequency to match the Kalman filter, each time measuring coordinates with a new relay.

Startup sequence

A startup sequence was implemented in order to determine the direction of each relay robot accurately. It works by for each robot collecting a larger sample of position estimations(50), using the RCMs as explained in Section 4.1.2, moving forwards 20 cm and then collecting another larger sample of position estimations. A vector pointing from the first position, $\vec{p}_0$, to the second position, $\vec{p}_1$, is then created. Norming this vector and taking the dot product between that vector and a normed vector parallel with the x-axis in the reference coordinate system would yield the cosine of the relays orientation. The orientation for the i:th robot would be as described in Equation (4.32) while the position would be $\vec{p}_{i,1}$.

$$\theta_i = \begin{cases} \arccos \frac{(\vec{p}_{i,1}-\vec{p}_{i,0})\cdot(1,0)^T}{||\vec{p}_{i,1}-\vec{p}_{i,0}||}, & p_{x,i,1} \geq p_{x,i,0} \\ 2\pi - \arccos \frac{(\vec{p}_{i,1}-\vec{p}_{i,0})\cdot(1,0)^T}{||\vec{p}_{i,1}-\vec{p}_{i,0}||}, & p_{x,i,1} < p_{x,i,0} \end{cases} \tag{4.32}$$

Collision Avoidance

In order to avoid damaging the relays it was necessary to implement some sort of collision avoidance. The approach was to detect a possible collision between two relays by comparing their predicted paths. If there was a risk for collision, the calculated controls would not be sent to one of the relays, and another comparison of the predicted path would be done to ensure that the collision avoidance worked. If there still was a risk for collision, none of the relays would receive the calculated controls and stop instead.

The collision detection used the prediction part of the Kalman filter and gradient descent on the

distance between the two relays as a function of time. One of the relays used the time variable t_a ($0 < t_a < \Delta t$) and the other t_b ($0 < t_a < \Delta t$) and if it existed a t_a and t_b such that the distance between the relays was less than a safe distance, the algorithm would consider this scenario as a possible collision.

4.4 Program structure

Until now necessary parts of the project have been introduced and explained, but to combine them all there has to be some surrounding code structure. There are two different packages which contains all of the code, the robotclient package and the masterclient package. The robotclient package includes everything that is needed for a robot and the masterclient package contains everything that is needed by the centralized base station. See Appendix E at page 63 for an overview of the entire code structure.

4.4.1 Masterclient package

The masterclient package was designed so that it can run both coordination algorithms. This means that the code structure includes for example initiation and class variables for both of the versions even though only one is used at a time. The package includes files for running the main program controlling the coordination algorithms, an iteration control ROS node, a relay class file, a Kalman filter class file and a controller class file.

The main program initiates as many instances of the relay class from the relay class file as there are robots currently wanted. These instances handle all of the service requests to the robots, includes several access functions, one instance of the Kalman filter class, one instance of the controller class as well as storage of relay information. Each relay needs one instance of a controller class and a Kalman filter class for the second version of the coordination algorithm. The Kalman filter class contains everything needed to predict a state for a relay while the controller class file contains everything needed to calculate new controls in order to move towards the current optimal placement.

Once the initiation is done the process waits for a service call to be made to a service *iterator*. This service call comes from the iteration control node with the name of the version to be used. Once the service is called it will pattern match on the requested version and run an iteration(after it has run the start up sequence if the simultaneous coordination algorithm is requested). As soon the first iteration has finished executing, the iteration controller node will keep requesting the service until it is terminated. For the first version of the coordination algorithm this is done as soon as the previous iteration has finished. For the second version of the coordination algorithm a delay will be added after the last iteration before the service is called again. The delay time is set to match the time unit for the Kalman filter. This has to be done so that the start up file keeps iterating in the same frequency as the time unit that supplied to the Kalman filter. When either version of the coordination algorithm deems it is satisfied with the result, the program can be terminated and the results plotted.

4.4.2 Robotclient package

The robotclient package contains everything that has to be running on a relay computer for it to be able to participate in the coordination. This includes all of the services that the masterclient package calls, code for managing the position requests to the RCMs and a ROSARIA node. Even though the package is identical on each relay computer, there are names and variables that needs alteration depending on which relay computer the package is on. This is so that the ROS Network can differentiate the relays and the relay computer services. The robotpackage is written so that the variables and names that has to be differentiated are given as a command-line parameter on start up. For example the service for acquiring the measured position for robot number one is run by

```
> rosrun robotclient get_coord_server.py
__name:=get_coord_server1 get_coord:=get_coord1
_ip_of_uwb:=101
```

where the names and IP address of the attached RCM is given. Since the code structure of the packages supports both versions of coordination algorithms, everything is preferably run always to ensure a none faulty run.

5

Results

This chapter presents the results of the positioning, movement algorithms, coordination and collision avoidance. All graphs that are shown were produced by plotting data that was logged and retrieved during testing if not otherwise stated. The results are split into the Sections 5.1 *Position estimation*, 5.2 *Iterative coordination algorithm scenarios* and 5.3 *Simultaneous coordination algorithm scenarios*. The following results used two relays constructed from Figure 4.1 at page 12 and is shown in Figure F.1 at page 64.

5.1 Position estimation

The position estimation consists of two parts. One is through trilateration using measurements, and the other is through prediction using a motion model. When combining the two estimations along with the Kalman filter, the result is a position estimation that is more accurate than the two parts alone. In this section, the results produced by simulations of trilateration, prediction and Kalman filtering will be presented. The same arbitrary initial states $\vec{x}_{1,0} = (1, 0, 4, 0, 0.4, 0)^T$ and $\vec{x}_{2,0} = (3, 0, -1, 0, 1.5, 0)^T$ were used and the variances were $\sigma_{meas} = 0.2$, $\sigma_X = 0.2$, $\sigma_Z = 0.1$. Positions of the base station and end node was $(-0.2, -4)^T$ and $(-4, 2)^T$. The initial covariance matrix used was calculated by simulating 30 iterations starting with $\mathbf{F}_0 = \mathbf{I}$ and using the controls $\hat{X} = \hat{Z} = 0$.

5.1.1 Trilateration

The results of a position estimation using distances acquired from measurements to the three anchors with known position as well as a simulation of a noisy alignment process can be seen in Figure 5.2 and Figure 5.4. It is evident that the measured distances have some uncertainty since there is no position where all of the circles intersect as can be seen in Figure 5.1. As explained in Section 4.1.2, the position would have to be estimated by minimizing the residual. It might not be easy to verify that the estimated position (yellow dot in Figure 5.1) minimizes it by simply looking at it because the axis ration is not 1:1 in Figure 5.1, but the residual was 0.025 which corresponds to an average 9 cm difference between the measured distance to an anchor and the distance from the estimated position to the same anchor ($\sqrt{\frac{0.025}{3}} = 0.09$ m).

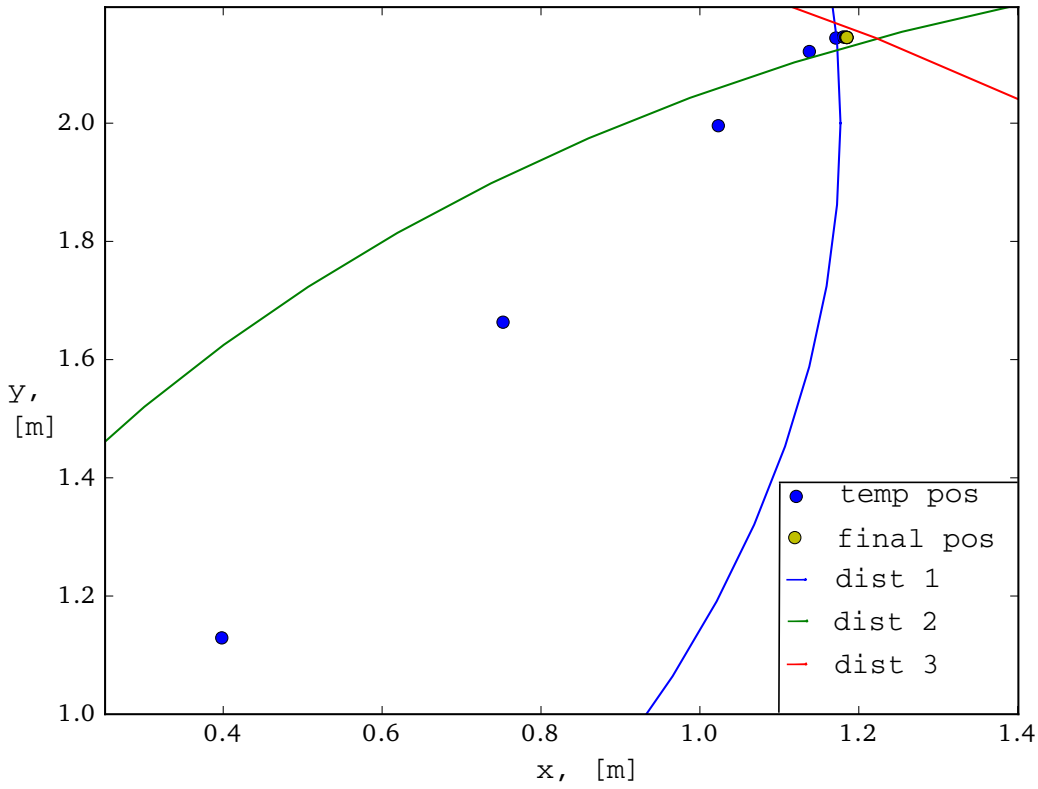


Figure 5.1: An example of the result of position estimation using measured distances to the anchors and gradient descent to find the position. The arcs are parts of circles centered at the anchors and with a radius corresponding to the measured distance to that anchor. The blue dots are positions estimated during the iteration and the yellow dot is the final estimated position that minimizes the residual.

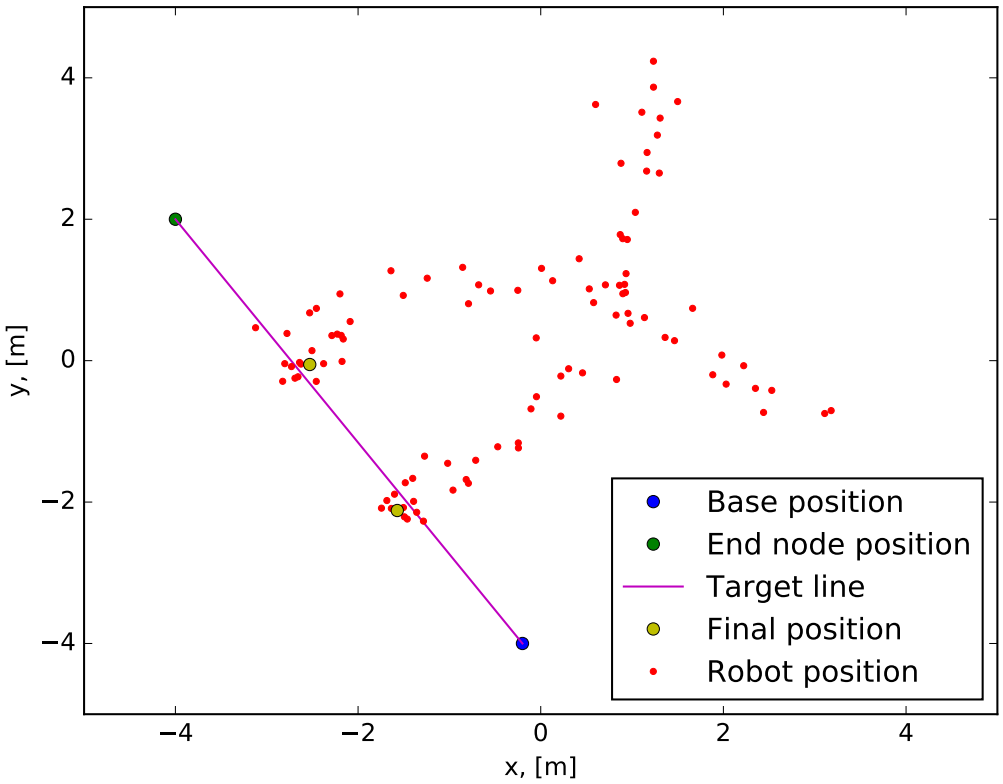


Figure 5.2: This image shows a simulation of the simultaneous coordination algorithm with two relays estimating their position using noisy measurements. The simulation ran for 100 time-steps which translates into a 50 seconds long alignment process.

5.1.2 Prediction

In order to get an understanding of how accurate the motion model of the Kalman filter is, a simulation was run using the initial configuration mentioned in Section 5.1 and the prediction part of the Kalman filter alone. From Figure 5.3 it is evident that the position estimated with the prediction part of the Kalman filter accumulates errors as time goes on. This is why the final position is not on the line between the base and end node.

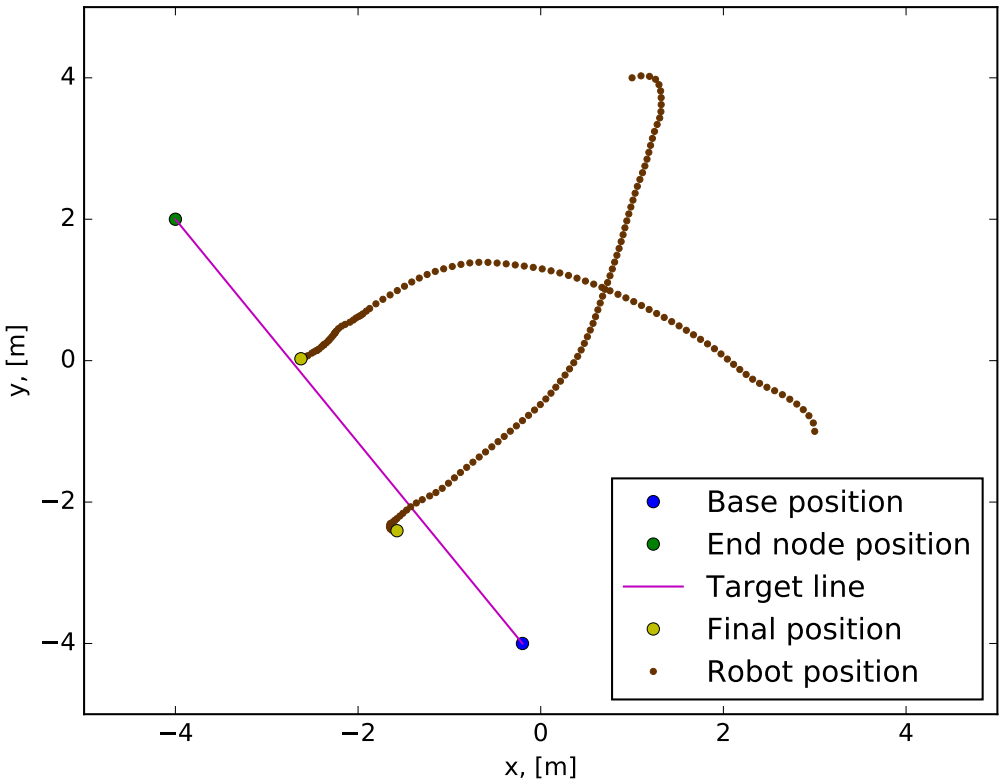


Figure 5.3: This image shows a simulation of the simultaneous coordination algorithm with two relays estimating their position using noisy input controls. The simulation ran for 100 time-steps which translates into a 50 seconds long alignment process.

5.1.3 Extended Kalman filter

Combining the prediction and result from trilateration along with the Kalman filter resulted in a a more accurate position estimation than the prediction and trilateration alone. It can be seen that the final position in Figure 5.4 is close to the target line and nearly equally spaced between the base and end node.

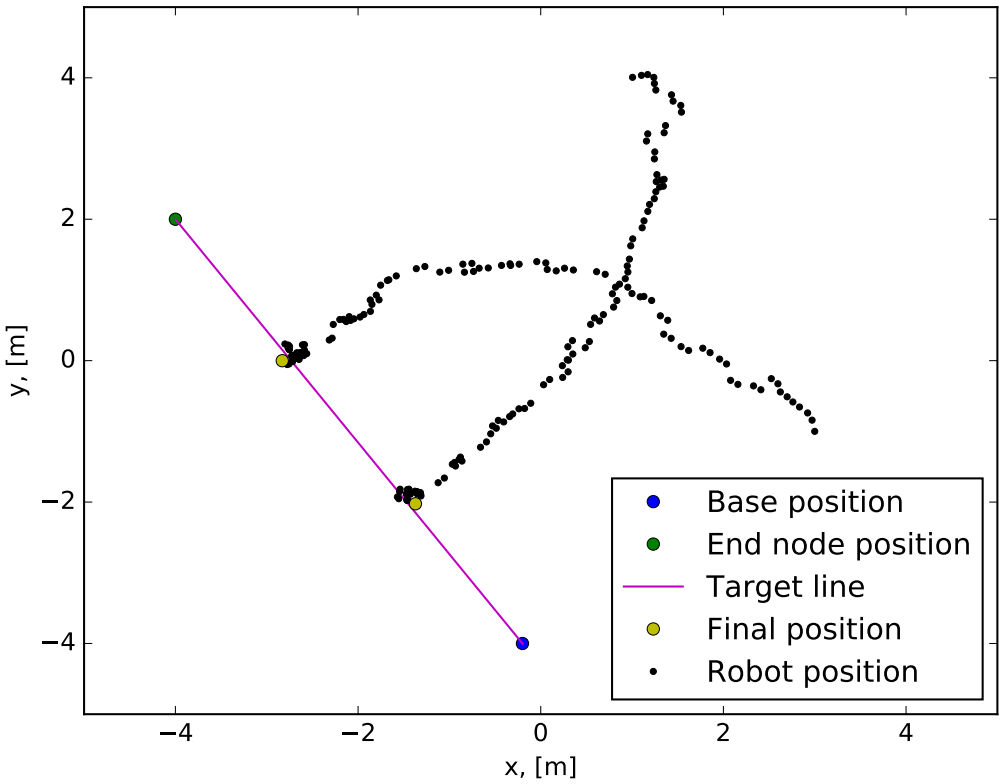


Figure 5.4: This image shows a simulation of the simultaneous coordination algorithm with two relays using an extended Kalman filter to estimate their positions when controls and measurements were noisy. The simulation ran for 100 time-steps which translates into a 50 seconds long alignment process.

5.2 Iterative coordination algorithm scenarios

This section will provide the results of the iterative coordination algorithm explained in Section 4.3. As previously explained, the iterative coordination algorithm uses the sequential movement algorithm in Section 4.2.1 and the trilateration from Section 4.1.2 to estimate the relays positions. In the figures that follow, the positions that are presented as black dots are the measured positions acquired from using trilateration and RCMs, the red crosses are the positions of the anchors, the red line displays the relays paths and the green circles shows the target positions at each time instant. The base station's and the fixed end node's positions are plotted as light blue dots and purple dots respectively. The base station's position is continuously updated using an RCM while the end node is fixed. It is also noteworthy that the position dots do not depict the actual size of a relay, which would roughly be four times larger than the dots, if shown in the figure.

For all of the the following scenarios, Table 5.1 shows the values of constants (settings) needed for producing the results. Naturally some initial state is also necessary and will be declared for each

scenario separately.

Linear movement distance	0.2 m
Sleep time linear	1 s
Angular speed	0.5 rad/s
Range measurements per Anchor	1
Ok distance to target	0.1 m
Anchor no. 1 position	$(-1, 2)^T$
Anchor no. 2 position	$(2, 0)^T$
Anchor no. 3 position	$(-1, -2)^T$

Table 5.1: Constants used for all of the scenarios when running the iterative coordination algorithm.

5.2.1 Relays starting in the same area

For the first scenario, the relays start in the upper right corner to simulate them originating from the same area by the base station. Table 5.2 shows the scenario's initial state and the settings from Table 5.1 are used to produce the result in Figure 5.5.

Base station initial position	$(1.1, 2.0)^T$
End node position	$(0, -2)^T$
Relay no. 1 initial position	$(1.9, 1.4)^T$
Relay no. 2 initial position	$(1.6, 0.6)^T$
Initial orientation for relay no. 1	180°
Initial orientation for relay no. 2	225°

Table 5.2: Additional constants used apart from the ones displayed in Table 5.1 used for the scenario *relays starting in the same area* when running the iterative coordination algorithm.

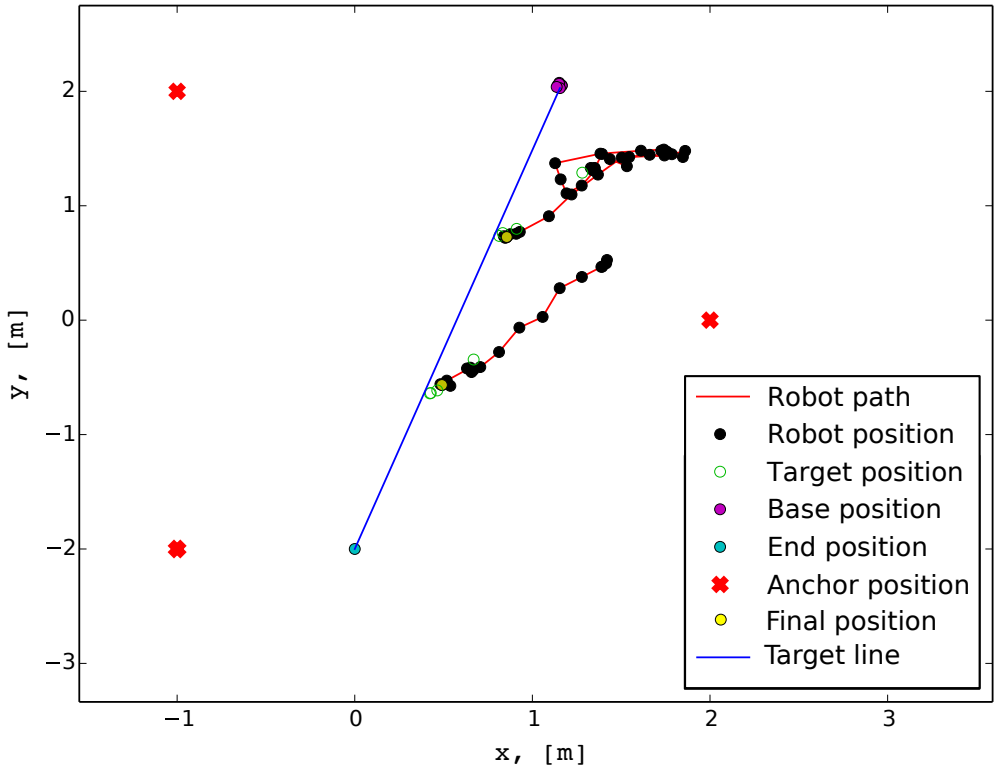


Figure 5.5: A complete run of the iterative coordination algorithm. Here, the relays start in the upper right corner of the plot, take turns in moving to the current optimal placement, until they eventually stop equally spaced on the line between the base station and the end node.

As seen in Figure 5.5 the relays approach the final optimal placements by iterative movement to the current optimal placement (target position). The total execution time as well as the number of iterations of the iterative coordination algorithm can be seen in Table 5.3. Note that one iteration of the iterative coordination algorithm constitutes of one calculation of a target position and then sequentially moving until that target position is reached.

Number of iterations	3
Execution time	89 s

Table 5.3: Number of iterations and execution time for the scenario *relays starting in the same area* when running the iterative coordination algorithm.

5.2.2 Relays approaching from opposite directions

For the second scenario, the relays start in opposite corners and then proceeds to approach their optimal placements. Table 5.4 shows the scenario's initial state. The settings shown in Table 5.1 were used to produce the results in Figure 5.6.

Base station initial position	$(1.1, 2)^T$
End node position	$(0, -2)^T$
Relay no. 1 initial position	$(-1.1, 1.3)^T$
Relay no. 2 initial position	$(1.1, -1.2)^T$
Initial orientation for relay no. 1	270°
Initial orientation for relay no. 2	90°

Table 5.4: Additional constants used apart from the ones displayed in Table 5.1 used for the scenario *relays approaching from opposite directions* when running the iterative coordination algorithm.

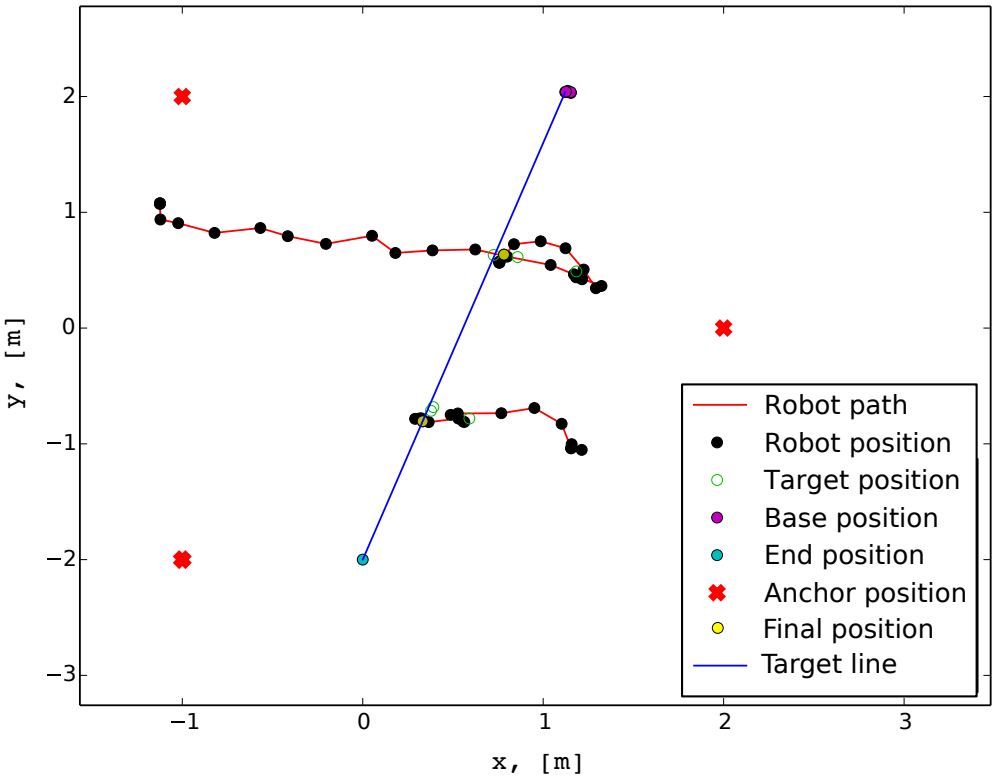


Figure 5.6: The second completion of an optimization scenario using the iterative coordination algorithm. One relay starts up to the left and down to the right.

As can be seen in Figure 5.6 the relays approach their targets positions until the final optimal placements are reached. Note that the first target position of the relay with the initial position at $(-1.1, 1.3)^T$ overshoots the final placement line since the second relay is still at its initial position of $(1.1, -1.2)^T$. The total execution time as well as the number of iterations of the iterative coordination algorithm can be seen in Table 5.5.

Number of iterations	2
Execution time	56 s

Table 5.5: Number of iterations and execution time for the scenario *relays approaching from opposite directions* when running the iterative coordination algorithm.

5.2.3 Base station moving during execution time

For the last scenario the iterative coordination algorithm is tested as the base station is moved during execution time. Table 5.6 shows the scenario's initial state. The initial state in combination with the settings from Table 5.1 was used to produce Figure 5.7.

Base station initial position	$(1.2, 1.7)^T$
End node position	$(0, 2)^T$
Relay no. 1 initial position	$(1.1, 0.7)^T$
Relay no. 2 initial position	$(0.7, -0.8)^T$
Initial orientation for relay no. 1	$170°$
Initial orientation for relay no. 2	$160°$

Table 5.6: Additional constants used apart from the ones displayed in Table 5.1 for the scenario *base station moving during execution time* when running the iterative coordination algorithm.

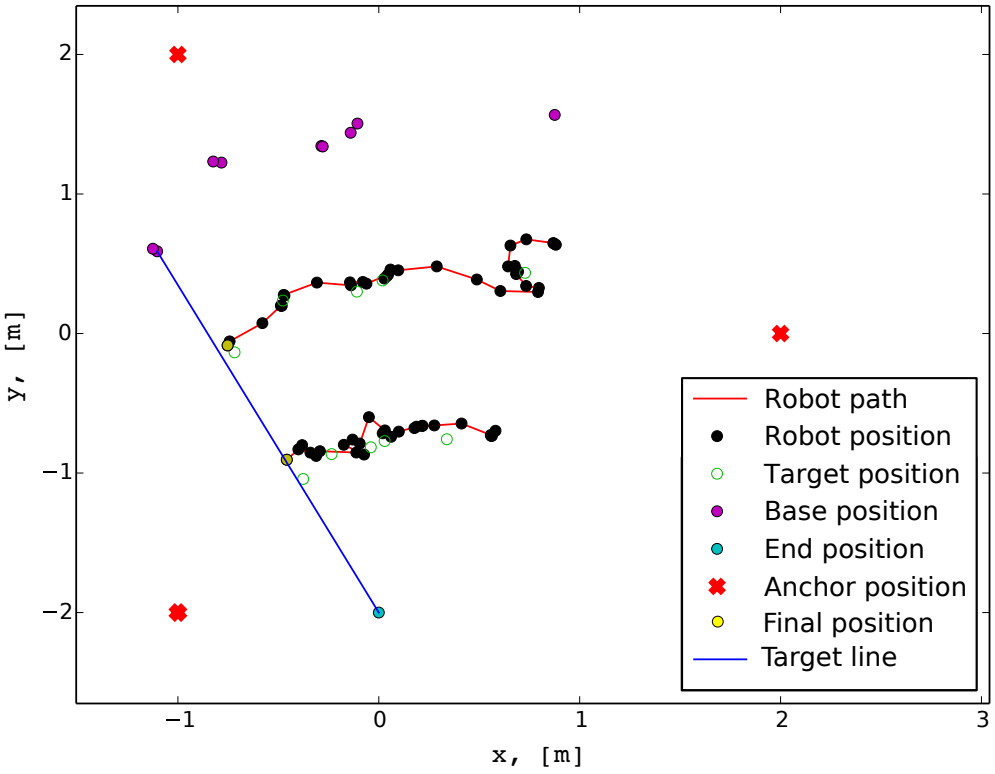


Figure 5.7: Demonstration of the continued preservation of the optimal positions for the relays as the base station is moved in an arc for the iterative coordination algorithm.

As seen in Figure 5.7 the relays follow the base station as its position is moved. Note though that the base station position is fetched as soon as it is needed for calculating the next target position and is therefore only updated as many times as the target position is. The total execution time as well as the number of iterations of the iterative coordination algorithm can be seen in Table 5.7.

Number of iterations	4
Execution time	53 s

Table 5.7: Number of iterations and execution time for the scenario *base station moving during execution time* when running the iterative coordination algorithm.

5.3 Simultaneous coordination algorithm scenarios

This section will provide the results of the simultaneous coordination algorithm explained in Section 4.3.3. As previously explained, the simultaneous coordination algorithm uses the smooth movement algorithm presented in Section 4.2.2 and the Kalman filter from Section 2.3 to estimate the relays' positions. The figures basically present the same information as for the iterative coordination

algorithm explained in Section 5.2. Note that the position of the robot is the position corrected by the Kalman filter.

For the following scenarios where the simultaneous coordination algorithm is used, Table 5.8 shows the values of constants (settings) needed for producing the results. Note that this does not include the simulated perfected scenario in 5.3.1. The initial states will be declared for each scenario separately.

Linear speed maximum:	0.2 m/s
Linear speed minimum	0 m/s
Angular speed maximum	1 rad/s
Angular speed minimum	0 rad/s
Standard deviation in velocity	0.05
Standard deviation in angular velocity	0.025
Standard deviation in measurement	0.05
Gradient Decent constant	0.3
Desired Iteration time	0.25 s
Velocity time constant	1 s
Angular velocity time constant	1 s
Range measurements per Anchor	1
OK distance to target	0.1 m
Anchor no. 1 position	$(-1, 2)^T$
Anchor no. 2 position	$(2, 0)^T$
Anchor no. 3 position	$(-1, -2)^T$
Safe distance for Collision Avoidance	0.7 m

Table 5.8: Constants used for all of the scenarios when running the simultaneous coordination algorithm.

5.3.1 Perfect scenario

To show how the simultaneous coordination algorithm performed in a perfect scenario (meaning $\sigma_{meas} = \sigma_X = \sigma_Z = 0$), a simulation was made using the same initial configuration as in Figure 5.4 and can be seen in Table 5.9. The results of this simulation can be seen in Figure 5.8.

Base station initial position	$(-0.2, -4)^T$
End node position	$(-4, 2)^T$
Relay no. 1 initial state	$(1, 0, 4, 0, 0.4, 0)^T$
Relay no. 2 initial state	$(3, 0, -1, 0, 1.5, 0)^T$

Table 5.9: Additional constants used apart from the ones displayed in Table 5.8 for the *perfect scenario* when running the simultaneous coordination algorithm.

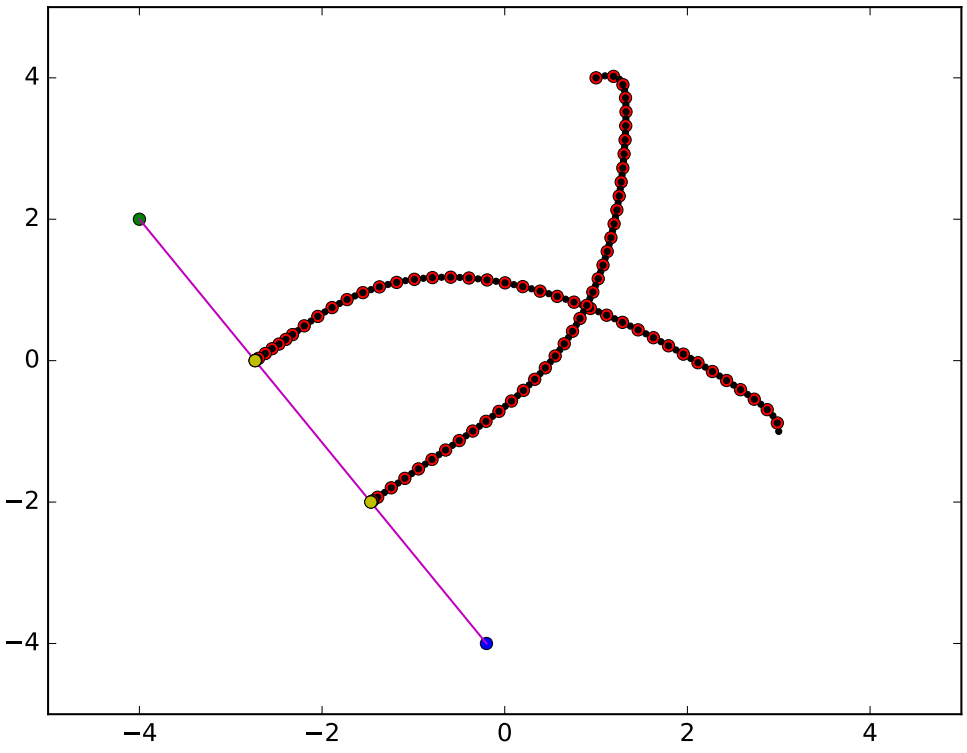


Figure 5.8: This image shows a simulation with two relays using an extended Kalman filter to estimate their positions when no control noise or measurement noise was present. The red dots are measurements of the relays' positions and the smaller black dots are the position estimated by the filter. The final positions of the nodes are the yellow dots approximately equally spaced on a line between the base station and end node (blue and green dots). The simulation ran for 80 time-steps which translates into a 40 seconds long alignment

5.3.2 Relays starting in the same area

For the first scenario the relays start in the upper right corner in order to simulate them originating from the same area by the base station. Table 5.10 shows the scenario's initial state. The original state in combination with the settings shown in Table 5.8 resulted in the Figure 5.9.

Base station initial position	$(1.1, 2.1)^T$
End node position	$(0, 2)^T$
Relay no. 1 initial position	$(1.8, 1.4)^T$
Relay no. 2 initial position	$(1.6, 0.5)^T$
Initial orientation for relay no. 1	183°
Initial orientation for relay no. 2	231°

Table 5.10: Additional constants used apart from the ones displayed in Table 5.8 for the scenario *relays starting in the same area* when running the simultaneous coordination algorithm.

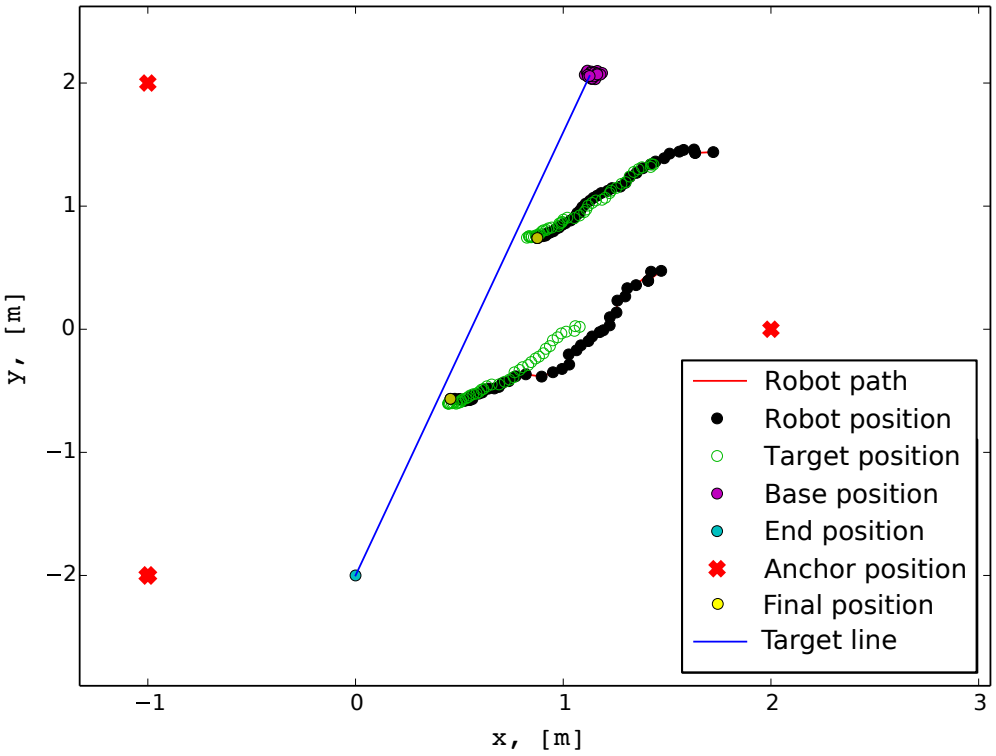


Figure 5.9: The full execution of the simultaneous coordination algorithm. The relays start in the upper right corner of the plot, then simultaneously move to the optimal position in between their neighbors, and eventually stopping equally spaced on the line between the base station and the end node.

As can be seen in Figure 5.9, the relays move towards the optimal placement in small steps by continuously calculating a target position that lies on the path towards the current optimal placement. The total execution time as well as the number of iterations of the simultaneous coordination algorithm can be seen in Table 5.11. Note that one iteration of the simultaneous coordination algorithm consists of a continuous recalculation of a target position and updated controls. The controls as a function of time can be seen in Figure 5.10.

Number of iterations	43
Execution time	12 s

Table 5.11: Number of iterations and execution time for the scenario *relays starting in the same area* when running the simultaneous coordination algorithm.

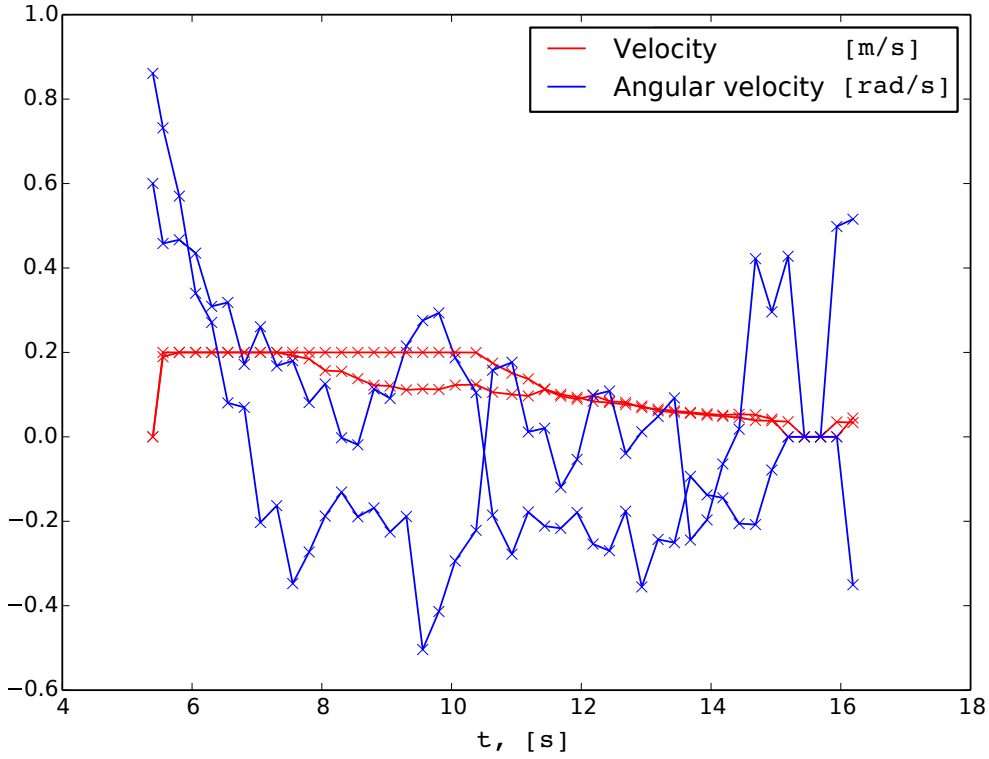


Figure 5.10: Plot for each relay's velocity (m/s) and angular velocity (rad/s) displayed on the y-axis, over time (s) on the x-axis, for the scenario *relays starting in the same area* when running the simultaneous coordination algorithm. Note that the final rotation speed is relatively large but the linear is low. This is likely due to fluctuations in the position due to measurements causing the target position to be slightly further away than the predefined *OK distance to target* from Table 5.8. The angle between the relay's heading direction and the target position can vary greatly which can yield a wide range of rotation controls.

5.3.3 Relays approaching from opposite directions

In this scenario, the relays start in opposite corners and then proceeds to approach their optimal placements. Table 5.12 shows the scenario's initial state for this scenario. The settings shown in Table 5.8 are used to produce the results that are shown in Figure 5.11.

Base station initial position	$(1.2, 2.1)^T$
End node position	$(0, 2)^T$
Relay no. 1 initial position	$(-1.1, 1.3)^T$
Relay no. 2 initial position	$(1.1, -1.2)^T$
Initial orientation for relay no. 1	$292°$
Initial orientation for relay no. 2	$80°$

Table 5.12: Additional constants used apart from the ones displayed in Table 5.8 for the scenario *relays approaching from opposite directions* when running the simultaneous coordination algorithm.

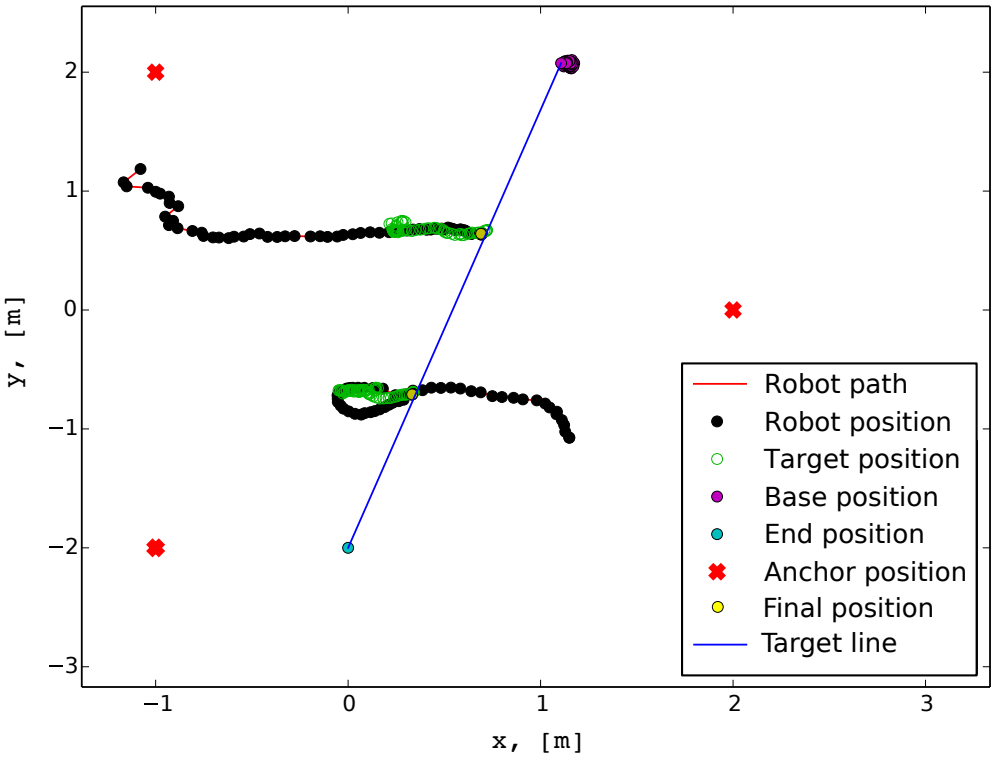


Figure 5.11: Plot of the simultaneous coordination algorithm that moves the relays where relay 1 starts up to the left and relay 2 starts down to the right. Notice how the target position for relay 1 is always *in the direction* of the area in between its neighbors, and vice versa for relay 2, until the target positions end up at the end positions for the relays upon which the optimization is completed.

Similarly to the first scenario presented in Section 5.3.2, the relays move towards the final optimal placement in small continuously updated target position. It should be noted that relay no. 2 with initial position $(1.1, -1.2)^T$ is closer to the optimal position and seems to get better position estimations in the beginning resulting in a simpler and faster path to the optimal placement. Thus, the relay managed to reach the optimal placement before the other relay did and in the end it had to retreat for the final placement. The total execution time as well as the number of iterations of the simultaneous coordination algorithm can be seen in Table 5.13. The controls as a function of time can be seen in

Figure 5.12.

Number of iterations	63
Execution time	17 s

Table 5.13: Number of iterations and execution time for the scenario *relays approaching from opposite directions* when running the simultaneous coordination algorithm.

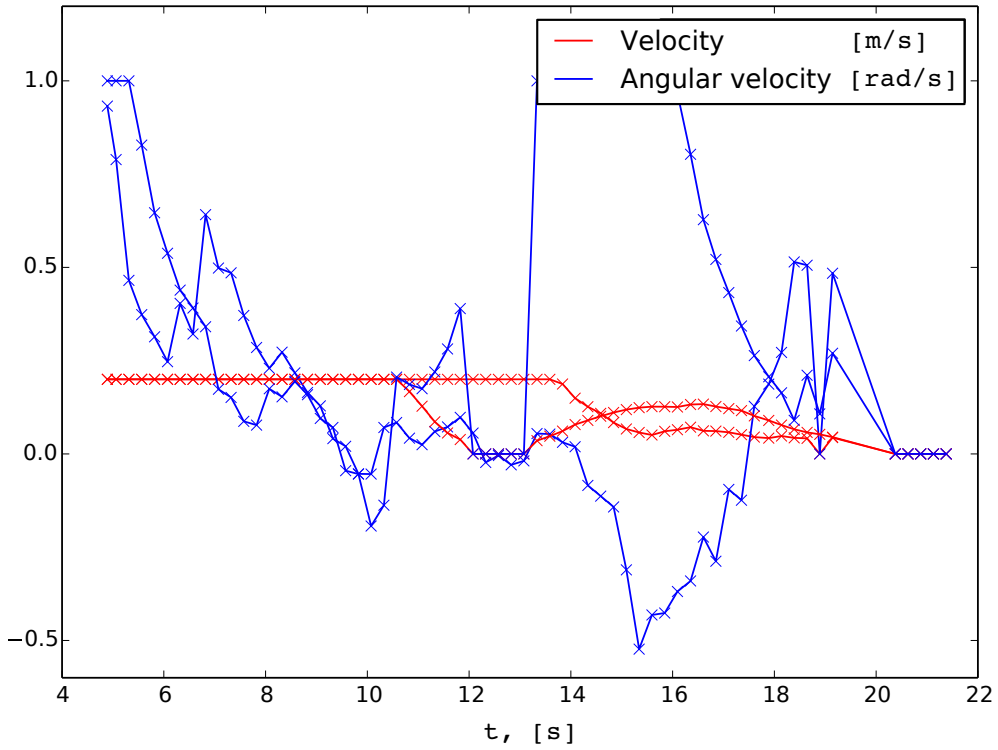


Figure 5.12: Plot for each relay's velocity (m/s) and angular velocity (rad/s) displayed on the y-axis, over time (s) on the x-axis, for the scenario *relays approaching from opposite directions* when running the simultaneous coordination algorithm.

5.3.4 Triggering collision avoidance

For this scenario, the collision avoidance for the simultaneous coordination algorithm was tested. Table 5.14 shows the scenario's initial state and the settings from Table 5.8 was used to produce the results in Figure 5.13. Note that the safe distance for before the collision avoidance signals the stopping of the relays is 0.7 m.

Base station initial position	$(1.1, 2.1)^T$
End node position	$(0, 2)^T$
Relay no. 1 initial position	$(0.9, -1.0)^T$
Relay no. 2 initial position	$(0.4, 1.0)^T$
Initial orientation for relay no. 1	$82°$
Initial orientation for relay no. 2	$292°$

Table 5.14: Additional constants used apart from the ones displayed in Table 5.8 for the scenario *triggering collision avoidance* when running the simultaneous coordination algorithm.

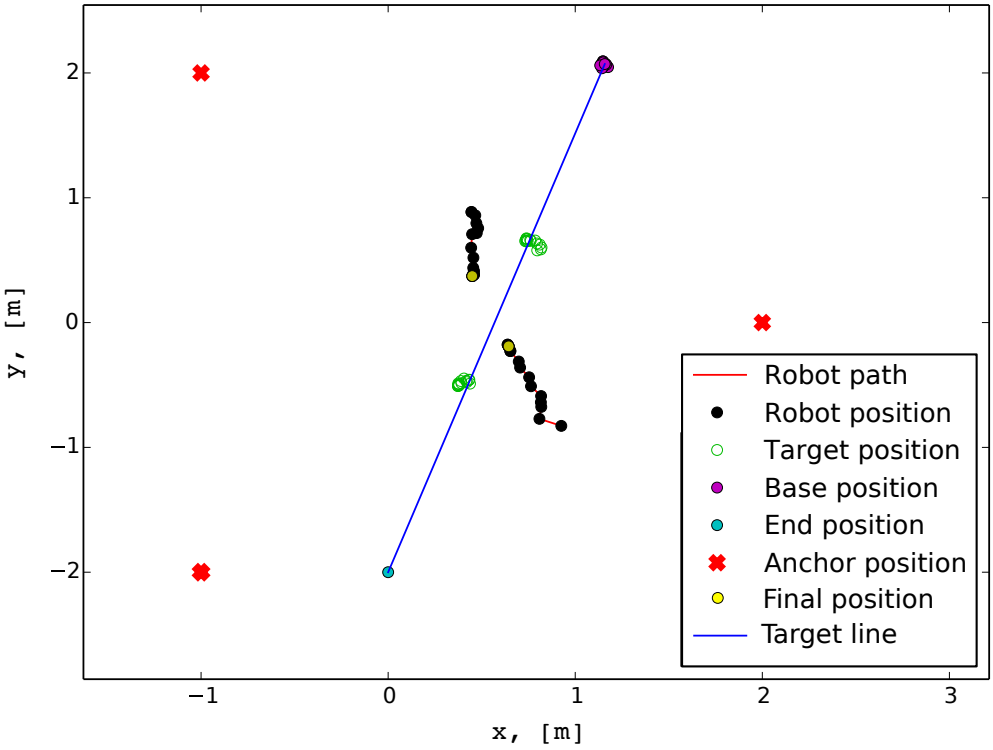


Figure 5.13: Triggering collision avoidance by starting the relays in a pair of positions so that their paths cross at the same time for the simultaneous coordination algorithm. The upper relay facing down toward the area where its optimal position is, and vice verse for the lower relay. Note that collision is avoided since the relays stop in front of each other.

In Figure 5.13, the collision avoidance for the simultaneous coordination algorithm is shown. Observe that the target positions for the relays never coincide with the actual positions (and that the relays never manages to reach the optimal placement line). This is because the collision avoidance algorithm described in Section 4.3.3 predicts a collision and returns a stop message to the main controller. The total execution time as well as the number of iterations of the simultaneous coordination algorithm can be seen in Table 5.15. The controls as a function of time can be seen in Figure 5.14.

Number of iterations	15
Execution time	6 s

Table 5.15: Number of iterations and execution time for the scenario *triggering collision avoidance* when running the simultaneous coordination algorithm.

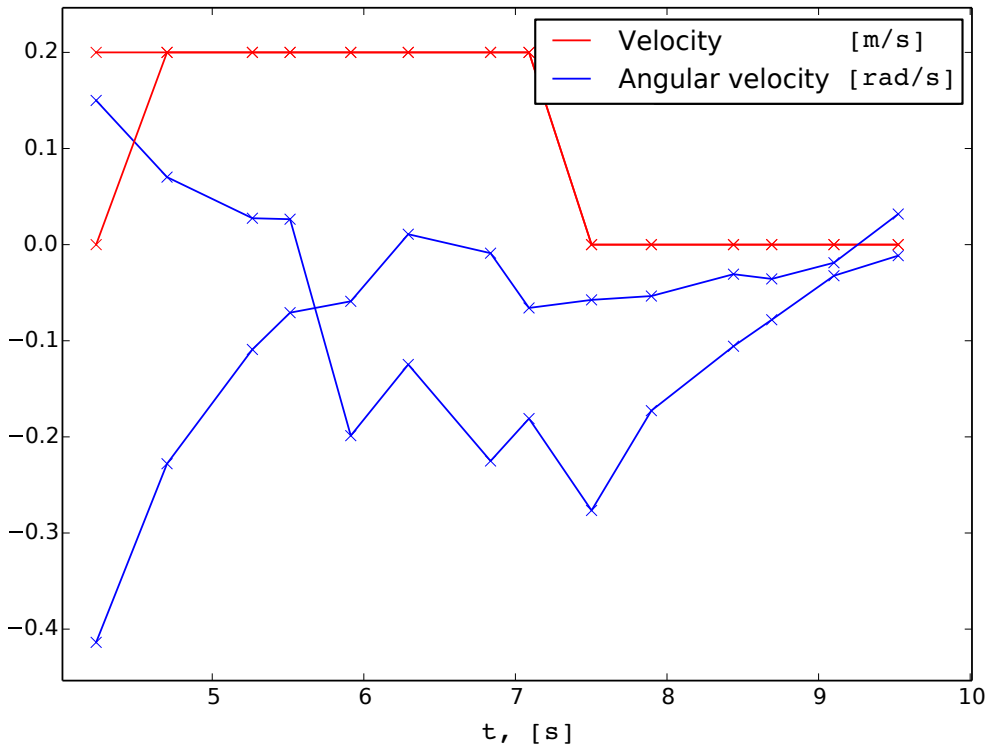


Figure 5.14: Plot for each relay's velocity (m/s) and angular velocity (rad/s) displayed on the y-axis, over time (s) on the x-axis, for the scenario *triggering collision avoidance* when running the simultaneous coordination algorithm.

5.3.5 Base station moving during execution time

The last scenario shows how the simultaneous coordination algorithms updates the optimal placements as the base station is moved during execution time. Table 5.16 shows the scenario's initial state and the settings from Table 5.8 are being used to produce the Figure 5.15.

Base station initial position	$(1.1, 2)^T$
End node position	$(0, 2)^T$
Relay no. 1 initial position	$(1.2, 0.9)^T$
Relay no. 2 initial position	$(0.8, -0.8)^T$
Initial orientation for relay no. 1	$170°$
Initial orientation for relay no. 2	$225°$

Table 5.16: Additional constants used apart from the ones displayed in Table 5.8 for the scenario *base station moving during execution time* when running the simultaneous coordination algorithm.

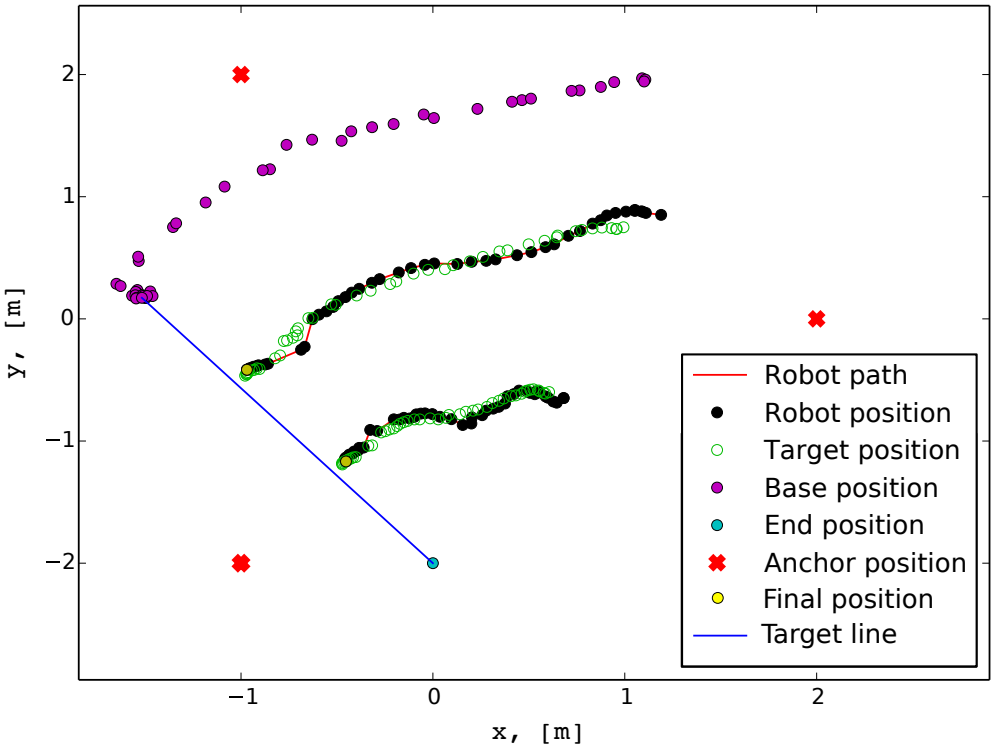


Figure 5.15: Altering the position of the base station in both directions in an arc shows that moving the base station results in maintained optimal positions for the relays using the simultaneous coordination algorithm.

Figure 5.15 show how updating the base stations position consciously affects the optimal placement of the relays. Note that the base station is moving even as the coordination is finished, which explains why the final positions of the relays don't entirely reach the final optimal placement. The total execution time as well as the number of iterations of the simultaneous coordination algorithm can be seen in Table 5.17. The controls as a function of time can be seen in Figure 5.16.

Number of iterations	43
Execution time	16 s

Table 5.17: Number of iterations and execution time for the scenario *base station moving during execution time* when running the simultaneous coordination algorithm.

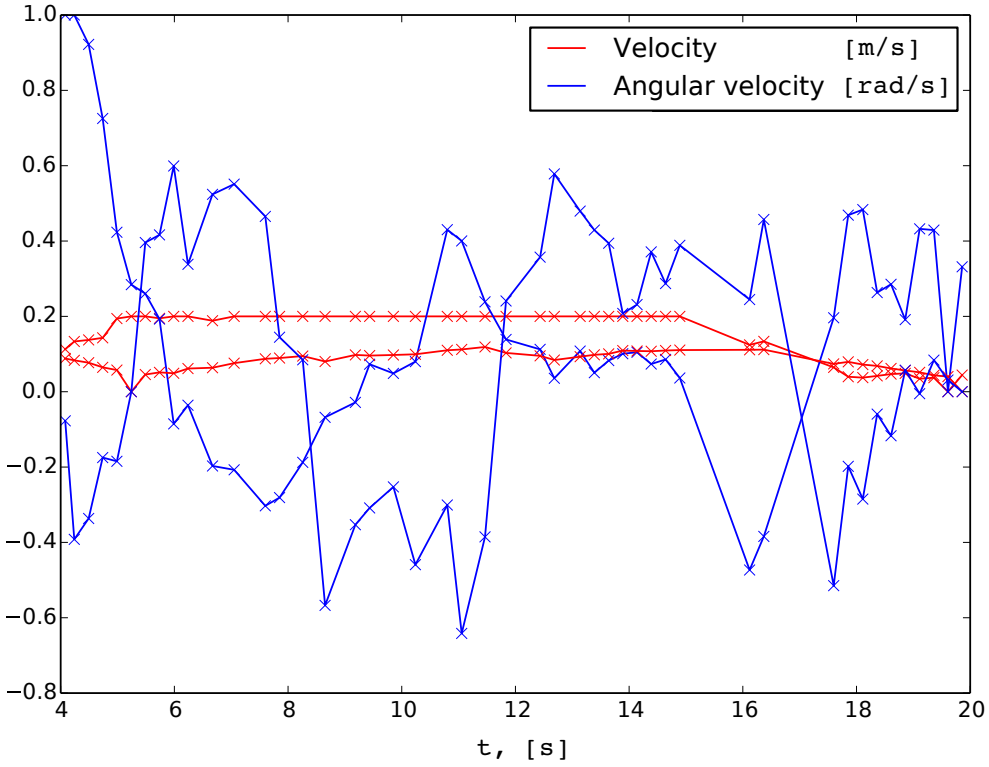


Figure 5.16: Plot for each relay's velocity (m/s) and angular velocity (rad/s) displayed on the y-axis, over time (s) on the x-axis, for the scenario *base station moving during execution time* when running the simultaneous coordination algorithm.

6

Discussion

This chapter will discuss the result of the project and whether the purpose was fulfilled or not. The graphs from Chapter 5 will be analyzed in order to get an understanding of how well the developed positioning, Kalman filters, movement algorithms as well as coordination algorithms work.

6.1 Justification of selected approach

Remember that in Sections 1.1 and 1.2 it was established that the relays should optimize connection quality through coordinated autonomous control. As mentioned in Section 4.3 two objective functions, that roughly model the signal quality, were used and minimized in order to generate targets for the relay robots to move to. These objective functions were however based on the distance between relay robots instead of other parameters that more accurately model the signal quality, for example the ones used in Mostofi's work [26]. In other words, the two more simplified objective functions do not take obstacles or disturbances into account.

There were two reasons for using simplified objective functions. Firstly, the testing area for the positioning was relatively small, which meant no relevant differences in signal quality could have been measured, rendering a signal quality based objective function ineffective. Secondly, the operating system for the robots that was used, ROS (see Section 3.2), does not natively support communication between roscores, meaning that it is not possible to extend a signal through multiple networks in ROS. Such a signal extension is required in order to measure necessary parameters for an objective function that is based on signal quality *between* the relays.

The rest of the purpose was thereby chosen to be focused on, namely developing coordination and movement algorithms for the relays. Due to this, a centralized network configuration was used instead of a decentralized configuration. Although a solution for a decentralized network was developed during the later stages of the project, it was not used because it was deemed irrelevant given our focus. The solution is comprised of a communication bridge between roscores on one machine, and can be found in Appendix G. As will become apparent through further discussions, the purpose was successfully fulfilled for a test area without obstacles, by the implementation of iterative coordination with sequential movement described in Section 4.2.1 and simultaneous coordination with smooth movement described in Section 4.2.2.

6.2 Equipment and software

This section will discuss the equipment and software that were used throughout this project.

6.2.1 Equipment

The relay construction was comprised of a robot, an RCM transceiver and a laptop, as shown in Figure F.1 at page 64. Since the more heavy computational code was run on the base station, the relay computers could have been replaced by single board computers such as Raspberry Pis. This would make the relay constructions simpler and more workable.

The RCMs produced accurate measurements, although they were perceived as unstable because unexplainable errors sporadically occurred during measurements. This could sometimes cause delays, which had an effect on the performance of the iterative coordination algorithm since the relays frequently had to stop and wait until the measurements were successful. It did however not have a noticable effect on the simultaneous coordination algorithm because if a measurement error occurred, a predicted position using the Kalman filter would instead be used.

There was also a limitation discovered regarding the RCM-transceivers. When measurements were made with two RCMs at the same time, interference would occur and the retrieved values would be faulty, requiring that only one relay measured its position at a time. Solutions for this were developed and are described in Section 4.3 for each coordination algorithm.

6.2.2 Software

When it comes to the software used, mainly ROS, its functionality has been feasible for this project. It supported a several things, such as python libraries and simple functionality explained in Section 3.2.4. As mentioned in Section 6.1, native support for communicating between roscores would undoubtedly have simplified the decentralized network configuration.

Furthermore, there was a bug in hostapd for the version (and multiple other versions) of Ubuntu that was used for the relay computers. The bug prevented the WAP to be started, which was fixed by restarting the system. Although the bug increased the testing time, it did not have an effect on the end outcome of the project.

6.3 Position Estimation

The position estimation was a crucial part of the project since it was used for both movement and coordination algorithms. One could argue that the position does not necessarily need to be known to maintain connection as long as it is possible to measure the connection quality. This is true but without information about the orientation or position the robot would have to measure the connection quality and move until it increases. This process would be very slow as the chance of moving in the

direction where the quality increases is very small. In our case we did not measure the connection quality so the position was the only variable we could use to change the connection quality, thus forcing us to solely rely on out ability to estimate it. For this reason we wanted to estimate the positions of the robots with as great precision as possible.

6.3.1 Using trilateration only

The performance of the software (or hardware or both, we do not know) of the RCMs seemed to be unreliable which made position estimation relying solely on the RCMs inaccurate and could cause the program to crash. We noted that we received a lot of errors when trying to measure distances to the anchors (RCMs) and we thought this was because every time the the robot would measure its distance from an anchor the program would open a connection, measure distances and calculate its position an then close the connection. As this happened several times every second we, not knowing a whole lot about socket programming, thought this problem might be solved by keeping the connection open for as long as the main program coordinating the relays. This proved to cause more errors as quite soon the RCMs started returning very weird measurement results and as soon as one error occurred, the preceding measurements would return the same error. This problem was solved by reverting to opening and closing the connection every time the position is measured. An important note to bring up is that the RCMs did return pretty accurate results when they worked which was most of the time.

6.3.2 Extended Kalman filter

It was difficult to construct a Kalman filter because the way we choose to (or had to) control the robots was by sending controls on the form of absolute values of the velocity and rotation rather than changes in these meaning that the state velocities were discarded. Because of this, we had some difficulties to compute the state transition matrix $\mathbf{F}$ and the $\mathbf{L}$-matrix since $\hat{\dot{x}}_{k|k-1} = \hat{\dot{y}}_{k|k-1} = \hat{\dot{\theta}}_{k|k-1} = 0$. At first we tried to keep the velocities in the state and let the controls instead be $\hat{X}_k = \hat{X}_{k,\text{input}} - \hat{X}_{k-1,\text{input}}$ and $\hat{Z}_k = \hat{Z}_{k,\text{input}} - \hat{Z}_{k-1,\text{input}}$ but this generated bad estimations so we had to scrap that solution. Our work-around to this problem was to let $\hat{\dot{x}}_{k|k-1} = \hat{X}\cos\hat{\theta}$, $\hat{\dot{y}}_{k|k-1} = \hat{X}\sin\hat{\theta}$ and $\hat{\dot{\theta}}_{k|k-1} = \hat{Z}$ whenever calculations involved these.

Due to a shortage of time we did not come up with a good way to generate enough data to properly calculate the standard deviations σ_X, σ_Z and σ_{meas} so we had to approximate an upper bound for these. We also did now have a good way to approximate the inital state covariance matrix $\mathbf{P}_{0|0}$ so our solution was to run a number or iteration of the Kalman filter with $\mathbf{P}_{0|0}$, the initial state $\vec{\hat{x}} = (0,0,0,0,0,0)^T$ and the controls $\hat{X} = \hat{Z} = 0$. The state convariance matrix $\mathbf{P}_{n|n}$ after n iterations with these controls and initial state would be the inital state covariance matrix $\mathbf{P}_{0|0}$ for the actual iterations. The iterations to find $\mathbf{P}_{0|0}$ was executetd when σ_{meas} was altered (which in our case only happened when the Control class was instantiated).

6.4 Movement algorithms

For the two coordination algorithms, two different movement algorithms, explained in Section 4.2, were used. The first movement algorithm moves a relay to a specified target position sequentially as explained in Section 4.2.1. The second movement algorithm moves a relay to a specified target smoothly as explained in Section 4.2.2. The sequential movement algorithm was developed in combination with the iterative coordination algorithm early on in the project with the objective of providing a simple way to move a relay from its current position to a target position. Although this was a success and the target position is reached, it was not without difficulties.

The first problem with the sequential movement algorithm is that the method for calculating controls is solely dependent on position measurements, which can be unstable and have considerable uncertainty. The second problem with the sequential algorithm is that the method for calculating the angle to turn is based on a comparison of the inclinations ($\frac{\Delta y}{\Delta x}$) of the position vectors. This is not entirely reliable because inclinations are not well defined if the position vector is parallel with the y-axis and can cause runtime errors. The third problem with the sequential movement algorithm is that the distances of which the position is deemed within range of the target are not perfectly calibrated. This becomes apparent when the relay approaches its target, where it sometimes has to turn around up to 180 degrees several times before a satisfactory position is found. This can be seen in Figure 5.5 where the relay starting in $(1.9, 1.4)^T$ has difficulties reaching its target position, causing the execution time to reach a total of 89 seconds. This could partly be solved by setting a less strict condition for the target, but it would lead to the relay settling a bit further away from the optimal placement line. The last problem with the sequential movement is that it is very seldomly called, as compared with the smooth movement algorithm, which means that it can not be updated with new information, such as target, during runtime. In other words, if a relay is moving towards a target using sequential movement and a new target is calculated while it is still moving, it has to finish positioning itself to the now obsolete target before the new target can be used. An example of such a situation is shown in Figure 5.7.

The smooth movement algorithm was developed later in the project in combination with the extended Kalman filter. The smooth movement algorithm was essentially developed to solve the problems with the sequential movement. Problems such as unreliability in calculating the controls were taken care of by calculating the angle between the node's current heading direction and the direction to its target position as this angle is well defined. Another challenge that arose when calculating the controls for the smooth movement algorithm was that in scenarios where the robot was facing away from its target position it would then move further away from the target while rotating. This was solved by multiplying the velocity by a factor $\frac{\pi-\theta}{\pi}$. It is worth mentioning that the method for calculating controls was not created with minimizing some variable (time or travel distance for example) in mind, making it a subject for improvement. The inefficiency problem was also dealt with since the target position is continuously updated with the minimizing of the target function and choosing a target position which headed towards the current optimal position closer to the current position, as explained in Section 4.3.1. The relays would then make their way towards the final optimal placement frequently and thus maintaining the connection would probably not be an issue. As explained in Section 4.2.2, the linear velocities and the angular velocities are simply sent to the relays and directly posted to the ROSARIA node before the function call returns. It doesn't wait for any movement execution to be done, which can be clearly seen in the control plots in Section 5.3. This is a positive feature when it comes to taking execution time from the coordination to the movement (it is beneficial to update positions frequently to check if the connection quality is worsening or improving) but it does require the service caller to frequently call the service with new controls. If velocity controls are sent to the robot and not updated with the supplied timing the relay would head in the wrong direction towards an outdated target position. For example if the updated controls tells the relay to simple rotate with

a given rotation speed and the controls aren't updated within a reasonable time frame ($\tilde{\mathrm{d}}\mathrm{t}$), the relay would spin on the spot. This is not a problem in our case since the implementation of the coordination algorithm updates the correct timing withing a reasonable time frame. The actual smooth movement in two degrees of freedom in combination with the above reasons shortens the execution time in comparison to the sequential movement algorithm. So for example in the scenario in Section 5.3.2, the total execution time was 12 seconds.

Even though the results seem to favor the smooth movement algorithm (which clearly follows the target positions better than the sequential movement algorithm does) there are variables such as movement speed and rotation speed that in the scenarios tested could be altered in favor of the sequential movement algorithm. Even if these variables were optimally adjusted to the scenario, the sequential movement algorithm would practically always be out-performed by the smooth movement algorithm.

6.5 Collision Avoidance

There was also collision avoidance implemented for the simultaneous coordination algorithm as explained in Section 4.3.3. The original plan was to implement collision avoidance which could alter the controls, so that the relays moved along a new path, towards their the optimal positions. This plan had to be reduced to only detecting collisions because the mathematical function used to calculate new controls was nonlinear and it sometimes took too long to compute the new controls (even when only two robots were used). This may be caused by us having the wrong approach to this problem and lack of time to investigate it further. The robots had support for sonars so we figured it would be better to leave it with a collision detection that prevented damaging the robots without interfering with the alignment process more than necessary, and leave the sonar as further development. The algorithm consisted of two levels of collision avoidance. The first was to stop one of the robots if the algorithm detected a collision and the second was to stop both robots if the algorithm detected collision even with one of the robots stopped as can be seen in Section 5.3.4.

6.6 Iterative coordination algorithm

The first coordination algorithm, the iterative coordination algorithm, was as previously mentioned the first stage of a working coordination algorithm. It makes use of the sequential movement algorithm discussed in Section 6.4 as well as position estimation with trilatiration and RCMs discussed in Section 6.3.1. Overall the iterative coordination algorithm works rather well, it manages to coordinate the relays to the positions defined by the target function in a simple way.

The main problem with the iterative coordination algorithm is that it only moves one relay at a time, which can lead to suboptimal pathing. An example of such a situation is in Section 5.2.2, where the relay with starting position $(1.1, -1.2)^T$ moves first and ends up moving to a target position unnecessary for the final optimal placement but is the current optimal placement since the other relay hasn't moved.

6.7 Simultaneous coordination algorithm

The simultaneous coordination algorithm is an improvement of the first coordination algorithm. The simultaneous algorithm, while making use of the extended Kalman filter discussed in Section 6.3 and the smooth movement algorithm discussed in Section 6.4, manages to simultaneously move multiple relays towards the frequently updated optimal placement in a smooth motion. Naturally, as with the iterative coordination algorithm, the pros and cons of the smooth motion as well as of extended Kalman filter affects the outcome of the simultaneous coordination method. The smooth movement algorithm enables fast movements in a reasonable path and the Kalman filter provides a better position estimation than using only measurements. The ability to use a predicted position when another relay is busy measuring a position enables the simultaneous coordination algorithm to become just that, simultaneous. This allows the system of relays to have continuously updated positions (whether they are predicted or corrected by measuring) at hand for calculating the next target position. This not only provides a continuously updated target position, but also a path to the final optimal placements where the relays always head in the direction of their targets, which can be seen in either of the plots provided in Section 5.3. Naturally, there can still be faulty position estimations and other disturbances which may provide a target position which is perhaps not in the exact right direction. However, since the simultaneous coordination algorithm uses the smooth movement algorithm, which updates new controls frequently, a faulty target position is not as fatal as it is for the iterative coordination algorithm. An example of where the usefulness of the frequently updated positions can be seen is in Section 5.3.5 where the base station is moved. Here it is quite clear that the frequently updated targets positions in relation to the base station provides a relay path that is well adjusted. This is also very clearly shown in Section 5.3.3 where the relay starting at $(-1.1, 1.3)^T$ seeks the other relay out so that a hypothetic connection between the relays would remain optimal.

6.8 Comparison of the coordination algorithms

The simultaneous coordination algorithm was originally intended to solve the problem of not being able to use several RCMs at the same time, but most of the other problems experienced from the iterative coordination algorithm were solved as well. The important part that shows why the simultaneous coordination algorithm is better than the iterative coordination algorithm is how much better the simultaneous coordination calculates target positions and follows these target position. What is noteworthy though when comparing the two coordination algorithms are that the iterative coordination algorithm could be improved in several different ways and still remain as an iterative coordination function. The smooth movement algorithm, some form of collision avoidance and the Kalman filter could for instance be implemented for the iterative coordination algorithm but since it seemed ideal that the relays move at the same time the iterative coordination algorithm was not improved further.

7

Conclusion

In conclusion, two methods for coordinating relays were developed. These include positioning with trilateration as well as kalman filters, movement algorithms, collision avoidance and objective functions. The second method, simultaneous coordination in Section 5.3 was dramatically more effective than the first method, iterative coordination in Section 5.2.

This project could be further extended upon and used in two ways; A more sophisticated collision avoidance algorithm could be developed in place of the currently used one, and the objective function that is minimized for target generation could be replaced with a different function, for example one that is based on signal strength and channel parameters, see [26]. Due to their inefficiency and primitive status, it is not recommended to use the iterative coordination method or sequential movement algorithm for further development.

Furthermore, if it is desired to extend this application for a decentralized network set up in ROS, it is possible to use the solution provided in Appendix A for enabling communication between ROS nodes.

Bibliography

[1] "Our Vision," *Amazon Robotics*, 2015, https://www.amazonrobotics.com/#/vision [Accessed: 09 May 2016].

[2] J. Rigelsford, "Autonomous robot system," *Industrial Robot: An International Journal*, vol. 30, no. 2, 2003. [Online]. Available: http://dx.doi.org/10.1108/ir.2003.04930bad.008

[3] B. Wallach, H. Koselka, and D. Gollaher, "Autonomous Multi-Platform Robot System," Patent US20 020 095 239, 18 Jul., 2002.

[4] T. Saitou, M. Nukada, and Y. Uchimura, "Deployment control of mobile robots for wireless network relay based on received signal strength," in *2013 IEEE Workshop on Advanced Robotics and its Social Impacts*. Tokyo: IEEE, 7-9 Nov. 2013, pp. 237–242, http://dx.doi.org/10.1109/ARSO.2013.6705535 [Accessed: 12 May 2016].

[5] G. B. Arfken and H. J. Weber, "Gradient," in *Mathematical methods for physicists international student edition*, 6th ed. Burlington: Academic Press, 2005, pp. 32–38.

[6] E. Polak, "Gradient methods," in *Optimization: algorithms and consistent approximations*, ser. Applied mathematical sciences. New York: Springer, 1997, vol. 124, pp. 56–70.

[7] "Trilateration," in *Encyclopaedia Britannica Online*, http://global.britannica.com/science/trilateration [Accessed: 11 May 2016].

[8] C. Chui and G. Chen, "Extended kalman filter and system identification," in *Gradient in Kalman filtering: with real-time applications*, 2nd ed. New York;Berlin: Springer-Vlg, 1991, pp. 108–130.

[9] C. K. Chui and G. Chen, "Kalman filter: An elementary approach in gradient," in *Kalman filtering: with real-time applications*, 2nd ed. New York;Berlin: Springer-Vlg, 1991, pp. 20–32.

[10] D. Simon, *Optimal state estimation: Kalman, H∞, and nonlinear approaches*. Hoboken, N.J. Wiley, 2006.

[11] *Data Sheet: PulsON 400 RCM*. USA: Time Domain, 2011, Available at http://www.timedomain.com/datasheets/320-0289A%20P400%20RCM%20Data%20Sheet.pdf, version 320-0289A, [Accessed: 11 May 2016].

[12] *Pioneer 3-DX*. USA: Adept Technology Inc., 2011, Available at http://www.mobilerobots.com/ResearchRobots/PioneerP3DX.aspx, version 09366-P3DX Rev. A, [Accessed: 11 May 2016].

[13] "N150: Wireless USB Adapter," *NETGEAR*, http://www.netgear.com/home/products/networking/wifi-adapters/WNA1100.aspx [Accessed: 12 May 2016].

[14] "Ubuntu 14.04.4 LTS (Trusty Tahr)," *Ubuntu*, 2016, http://releases.ubuntu.com/14.04/ [Accessed: 09 May 2016].

[15] "About ROS," *ROS*, http://www.ros.org/about-ros/ [Accessed: 09 May 2016].

[16] "Creative Commons: Attribution 3.0 United States," *Creative Commons*, http://creativecommons.org/licenses/by/3.0/us/legalcode [Accessed: 17 May 2016].

[17] "ROSARIA," *ROS.org*, 2016, http://wiki.ros.org/ROSARIA [Accessed: 09 May 2016].

[18] "ARIA," *Adept MobileRobots*, 2016, http://robots.mobilerobots.com/wiki/ARIA [Accessed: 09 May 2016].

[19] "Ubuntu install of ROS Indigo," *ROS.org*, 2015, http://wiki.ros.org/indigo/Installation/Ubuntu [Accessed: 09 May 2016].

[20] "How to use ROSARIA," *ROS.org*, 2016, http://wiki.ros.org/ROSARIA/Tutorials/How%20to%20use%20ROSARIA [Accessed: 12 May 2016].

[21] J. Malinen, "hostapd: IEEE 802.11 AP, IEEE 802.1X/WPA/WPA2/EAP/RADIUS Authenticator," *hostapd and wpa_supplicant*, 2016, http://w1.fi/hostapd/ [Accessed: 09 May 2016].

[22] "Creative Commons: Attribution 4.0 International," *Creative Commons*, http://creativecommons.org/licenses/by/4.0/legalcode [Accessed: 17 May 2016].

[23] "About hostapd," *Wireless Wiki*, 2015, https://wireless.wiki.kernel.org/en/users/documentation/hostapd?s[]=hostapd [Accessed: 09 May 2016].

[24] "Running ROS across multiple machines," *ROS.org*, 2016, http://wiki.ros.org/ROS/Tutorials/MultipleMachines [Accessed: 09 May 2016].

[25] D. K. Cheng, "Plane electromagnetic waves," in *Field and Wave Electromagnetics*, 2nd ed. Harlow: Pearson Education, 2014, pp. 354–426.

[26] Y. Yan and Y. Mostofi, "Robotic router formation in realistic communication environments," *IEEE Transactions on Robotics*, vol. 28, no. 4, pp. 810–827, Aug. 2012, http://dx.doi.org/10.1109/TRO.2012.2188163 [Accessed: 12 May 2016].

Appendices

A Network set-up

The WAP was configured in the terminal via the config file `hostapd.conf` described in Appendix D which is used by a user space daemon for access point configuration in Linux called `hostapd`. The config file configures the program to use the right settings, such as interface name (which in this case is what allows the computer to communicate with the adapter), driver, SSID (Service Set Identifier) of the WAP, password for the WAP, security protocol (WPA-PSK), etc. Another config file named

```
/etc/NetworkManager/NetworkManager.conf
```

is edited to add the adapter's MAC adress (Media Access Control) as an *unmanaged device*, meaning that the network manager program does not interfere with the adapter since we want it to be a WAP.

To configure a client, it is convenient to name the interfaces for the adapters similarly on all clients by editing the file

```
/etc/udev/rules.d/70-persistent-net.rules
```

and changing the interface name to e.g. *wlan2* for the line that corresponds to the adapter's MAC address. A network connection was configured in Ubuntu to connect to the network with the chosen SSID of the WAP, the adapter's device MAC address chosen to *wlan2*, preferably with automatic connection turned on, the *WPA and WPA2 Personal* Wi-Fi security option with the correct password for the WAP, IPv4 address setting to *192.168.2.(id+1)* where *id* is the ID of the computers, and netmask to *255.255.255.0*. For example, if the computers are called *multipos1* and *multipos2*, then the *id* would be 1 and 2 respectively, and the IP addresses *192.168.2.2* and *192.168.2.3* respectively. The computers acting as clients then connected to the WAP after the instructions above was carried out.

This README file gives step-by-step instructions on exactly how the network is set up in our project.
INSERT README

B ROS launch file structure

The launch file is built up by lines of code like

```
<machine name="robot1" address="192.168.2.2"
  env-loader="~/test.bash" user="multipos"/>
```

which uses the machine tag to define a computer called *robot1* and

```
<node machine="robot1" name="get_coord_server1"
  pkg="robotclient" type="get_coord_server.py"/>
```

which starts a service with the name *get_coord_server1* in the ROS package *robotclient* by running the file *get_coord_server.py*. This is equivalent to running the command

```
> rosrun robotclient get_coord_server.py
  __name:=get_coord_server1
```

The `.bash` file is necessary to load environmental variables to fulfill the last of the two criteria ROS has on the network, namely ensuring that every computer has a name by which it can advertise itself and that all the other computers can resolve. For a robot, it looks like following

```
#!/usr/bin/env bash

source /opt/ros/indigo/setup.bash
source ~/SSYX02-16-27/robot_ws/devel/setup.bash

export ROS_HOSTNAME=robot1
export ROS_IP=192.168.2.2
export ROS_MASTER_URI=http://192.168.2.1:11311

exec "$@"
```

where the first lines sources environmental variables for basic ROS functionality and package functionality, and the export lines setting more environmental variables declaring hostname, IP address for the hostname and IP address with port to the master computer. The first and last lines are important when using a `.bash` file. See the ROS wikipedia how to use `.sh` files instead.

C RCM set-up instructions

1. Connect Ethernet and power cable to RCM.

2. Configure Ethernet connection in Ubuntu similarly to the connection configured in Appendix A.

   (a) Set name (for example *RCM*).

(b) Choose an arbitrary IP address in the subnet *192.168.1.0/24* for the RCM.

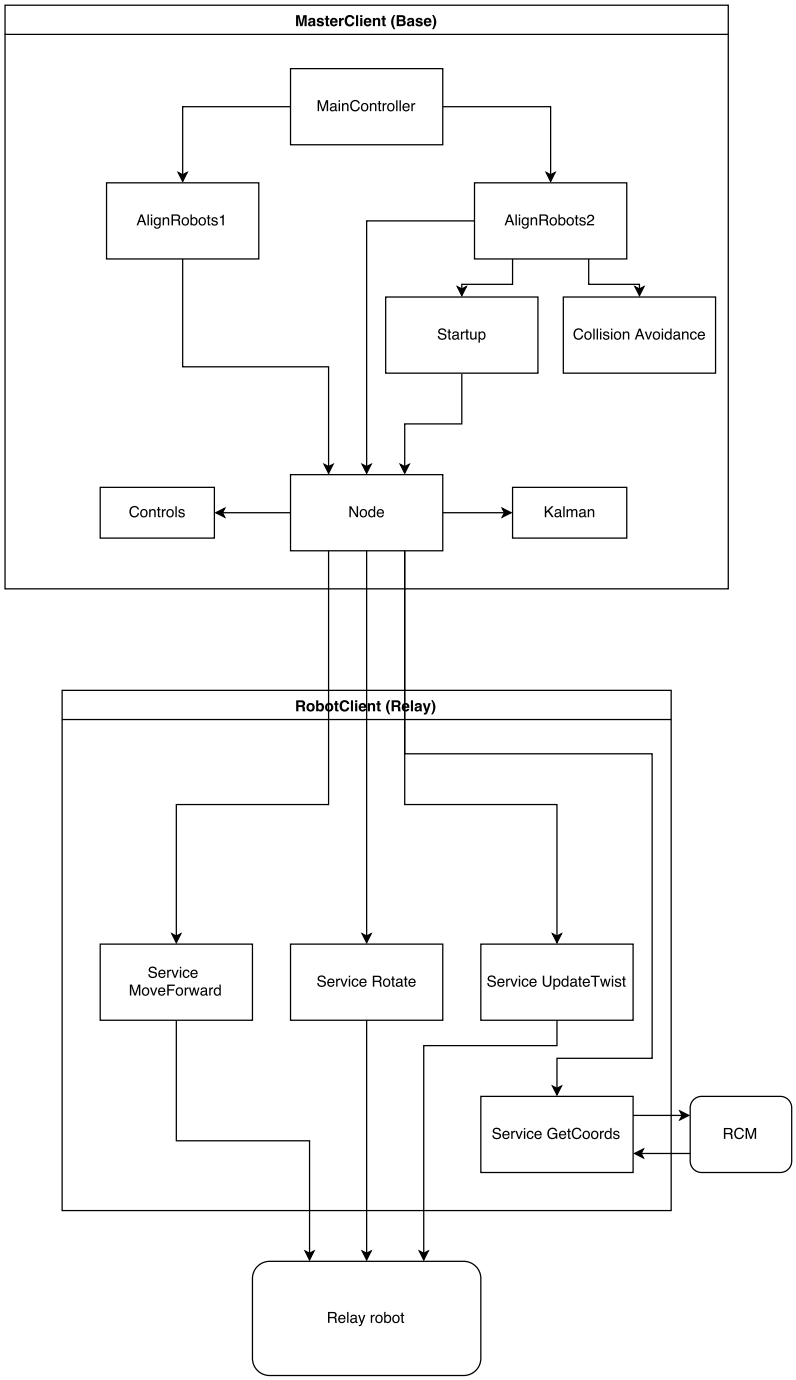
3. Try pinging chosen IP address to confirm that this procedure was done correctly.

D hostapd config file

Here, the config file `/etc/hostapd/hostapd.conf` for `hostapd` is displayed.

```
interface=wlan3
driver=nl80211
ssid=Base
hw_mode=g
channel=1
wpa=1
wpa_passphrase=risifrutti
wpa_key_mgmt=WPA-PSK
wpa_pairwise=TKIP CCMP
wpa_ptk_rekey=600
```

E The overall code structure



F Picture of the constructed relay

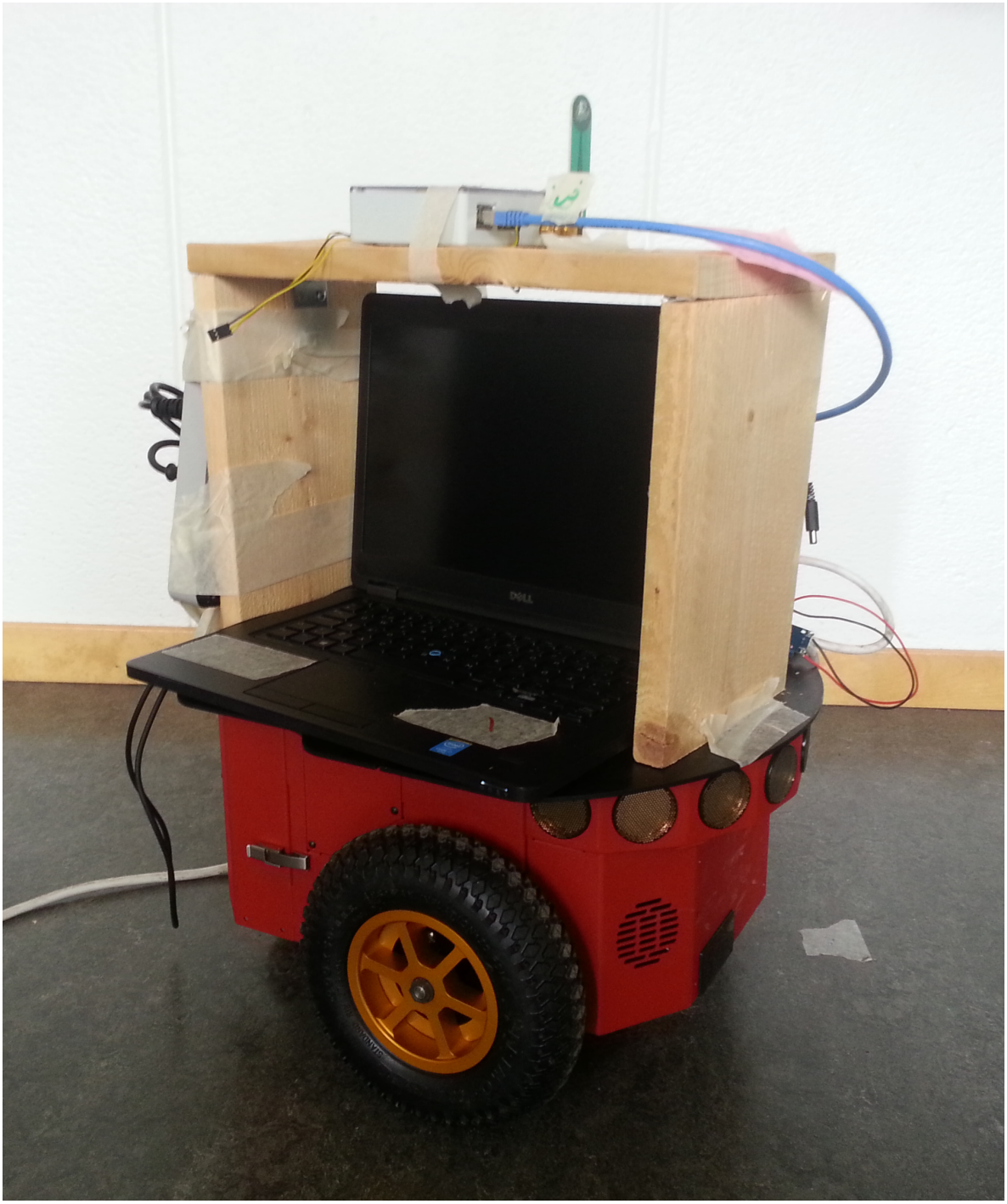


Figure F.1: Picture of the actual relay construction used. The RCM and node (laptop computer) mounted upon the Pioneer robot. There is a battery behind the laptop feeding the RCM power and thus allowing the relay to run without a power cord connected to it.

G Additional solutions

This chapter will describe discoveries that were not tied to the scope and delimitations for this project. As mentioned in the introduction and clarified in method, the focus of this project was to develop a control algorithm for autonomous coordination of robots, using a centralized network configuration. It was however also discovered how a decentralized network, in combination with a ROS network configuration, could be implemented.

G.1 Configuring a decentralized network with ROS

Unlike a centralized network, the idea of a decentralized network in this project means that each relay has a ROS network configuration for each of its neighbors. In other words, there would have to be one active roscore for a relay and its right neighbor, which would be the means of communication for that pair, and likewise for the left neighbor of the relay. The main challenge was then that it was not possible for two roscores on seperate networks to communicate, which meant a solution to forward the signal had to be made.

To solve this, a server bridge between roscores was implemented, which retrieves some information from one roscore and then injects it to the other roscore on that relay. This was done by setting up a local server on the relay computer which simply forwards the information from one roscore to the other with the use of clients. The clients would be configured to have two sides, one side would be configured as a ROS node and communicate with the core and the other side would be configured as a client written in python which communicates with the server.