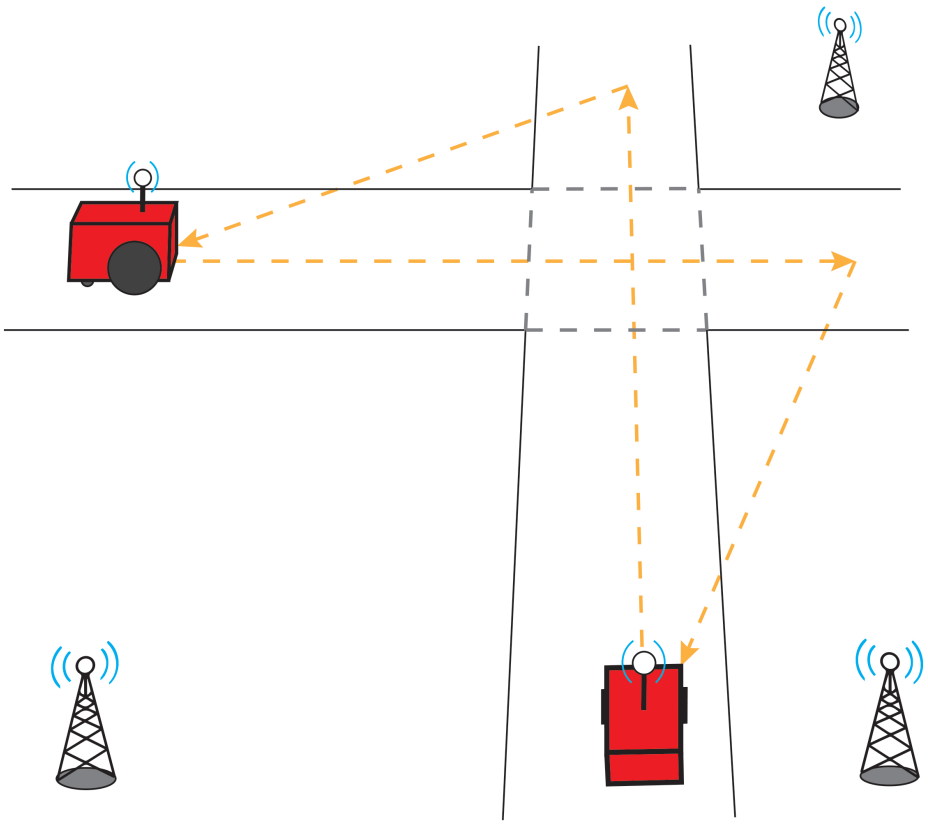
CHALMERS
UNIVERSITY OF TECHNOLOGY



Remote Automatic Control of Robots via Wireless Communication Networks

Intersection Coordination of Robots

Bachelor's thesis in Electrical Engineering

Arsam Shokrian
Sebastian Ylander Mikkelsen
Olof Düsterdieck
Erik Higbie

Supervisor: Mohammad Ali Nazari
Co-supervisor: Markus Fröhle
Examiner: Henk Wymeersch

Department of Electrical Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2017

Bachelor's thesis 2017:SSYX02-17-02

Remote Automatic Control of Robots via Wireless Communication Networks

Intersection Coordination of Robots

Arsam Shokrian
Sebastian Ylander Mikkelsen
Olof Düsterdieck
Erik Higbie

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Chalmers University of Technology
Gothenburg, Sweden 2017

Remote Automatic Control of Robots via Wireless Communication Networks
Intersection Coordination of Robots
ARSAM SHOKRIAN
SEBASTIAN YLANDER MIKKELSEN
OLOF DÜSTERDIECK
ERIK HIGBIE

Supervisor: Mohammad Ali Nazari, Department of Electrical Engineering
Co-Supervisor: Markus Fröhle, Department of Electrical Engineering
Examiner: Henk Wymeersch, Department of Electrical Engineering

Bachelor's Thesis 2017:SSYX02-17-02
Department of Electrical Engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Two robots moving in a path with an intersection, simultaneously communicating with the UWB sensors. The continuity of the path is visualized with the orange arrows.

Typeset in LaTeX
Gothenburg, Sweden 2017

Remote Automatic Control of Robots via Wireless Communication Networks
Intersection Coordination of Robots
ARSAM SHOKRIAN
SEBASTIAN YLANDER MIKKELSEN
OLOF DÜSTERDIECK
ERIK HIGBIE
Department of Electrical Engineering
Chalmers University of Technology

Abstract

The purpose of the project was to create a system for the coordination of two vehicles in an intersection. The system should prevent a collision between the vehicles as they pass the intersection with the help of a controller. It was decided that 50 intersections without collision would prove the correctness of the algorithms. The controller should use filtered position data in order to calculate its commands. The control commands should be optimized in order to increase the efficiency and comfort.

To develop and test the finished product, two Pioneer P-3DX robots together with five PulsOn 400 'Ultra-wideband' sensors were used. Three computers with 'Robot Operating System' installed were used where two of them were connected to the robots. On the third computer, the central-computer, the programs used to control the robots were ran. The communication of information between the robots and between the positioning system and the central-computer was done through Wi-Fi.

A robust and flexible system was developed that made the robots travel through a self-intersecting path and that satisfied the goal of 50 intersections without collision using optimized control commands.

Keywords: Robots, intersection, autonomous, collision avoidance

Sammandrag

Syftet med projektet var att skapa ett system för koordinering av två fordon i en korsning. Systemet skulle förhindra kollision av fordonen med hjälp av en beräknare medan de passerade korsningen. Det bestämdes att 50 korsningar utan kollision skulle påvisa att algoritmerna var korrekt. Beräknaren skulle använda filtrerad positionsdata för att beräkna passande kommandon. Utöver detta skulle styrsignalerna optimeras för att öka effektivitet och komfort.

För att utveckla och testa den färdiga produkten användes två Pioneer P-3DX robotar tillsammans med fem PulsOn 400 'Ultra-wideband' sensorer. Tre datorer med 'Robot Operating System' installerade användes där två av datorerna kopplades till robotarna. På den tredje datorn, den centrala datorn, kördes programmen som kontrollerade robotarna. Kommunikationen mellan robotarna och mellan positioneringssystemet och den centrala datorn skedde genom Wi-Fi.

Ett robust och flexibelt system utvecklades som körde robotarna genom en självkorsande bana och som uppfyllde målet av 50 korsningar utan kollision med optimerade styrsignaler.

Nyckelord: Robotar, korsning, autonom, kollisionsundvikning

Acknowledgements

We extend our deepest gratitude to our main supervisor Mohammad Ali Nazari and co-supervisor Markus Fröhle for guiding us through this project and always providing us with assistance and help. We would also like to thank our examiner Henk Wymeersch for enabling and administrating this project.

Arsam Shokrian, Sebastian Ylander Mikkelsen, Olof Düsterdieck, Erik Higbie, Gothenburg, May 2017

Contents

List of Figures

List of Tables

Glossary of Acronyms and Terms

ROS	Robot Operating System
ARIA	Advanced Robot Interface for Applications
UWB	Ultra-wide band
EKF	Extended Kalman Filter
Gradient	The multi-variable generalization of the slope of a curve at a given point
Riemann sum	An estimation of the area under the curve
Guard distance	Used when describing the priority protocol. Is the distance from the middle of the intersection to a displaced point in the y-axis were the controllable robot should not move farther than if the uncontrollable robot is in the intersection
Stable robot	Robot with constant velocity. Used in Priority and Hybrid solution
Variable robot	Robot that receives control actions thus changing its acceleration and its velocity. Used in Priority and Hybrid solution

1
Introduction

In a world with continuously increasing automation, many tasks that could only be done by humans can now be accomplished by machines. A clear area of this advancement is in transportation systems, and most relevantly, the autonomous vehicle. This can be seen even today with the widespread adoption of active safety systems and driver aids in vehicles, making not only travel safer but easier and more efficient. As these automated systems become more commonplace in vehicles and have a greater responsibility for aspects of driving, the time may come when these vehicles take over traffic coordination from their human occupants. This will make it possible for vehicles to either partially or entirely drive themselves on the roads of the future. Before that can happen however, there are a variety of challenges that must be addressed and solved. One of them is how to handle intersections of vehicles. As human drivers are removed from the intersection problem, the possibility for more optimal solutions to this problem appears.

There are different ways to handle intersections today. Some are purely human-based and were created to optimize safety, like the 'priority to the right' system where, when two vehicles meet at an intersection, the one on the right has priority. Traffic lights are another such rule based solution which try to optimize throughput by maintaining traffic flow through the intersection. Other intersection techniques are more structural and priority based, like the roundabout, created to try to give better throughput while maintaining high safety.

Unfortunately, these solutions may be far from optimal in regards to energy consumption, passenger comfort, time and foremost safety. In many situations, vehicles may have to wait for other vehicles by slowing down or coming to a complete stop, while waiting for the intersection to clear or a traffic light to turn green. In combination with human responsiveness, reflexes and movement capabilities, these factors lay the basis for many forms of traffic congestion and potential collisions. Autonomous vehicles can minimize or even eliminate these shortcomings by having vastly superior response time and the ability to communicate to each other wirelessly [1]. These abilities could solve the coordination problem that intersections pose in a faster and far more efficient way.

The idea of optimal traffic flow is not only specific for intersections but is one of the main arguments for autonomous transportation. Enhanced safety, efficiency, pollution, and comfort are among the clear advantages of automating the transportation

process [2]. These advantages allow the vehicles in use to enhance the driving experience in a way that is difficult to implement in the human controlled vehicles used today.

1.1 Purpose

The goal of this project is to have a completely implemented solution for controlling autonomous vehicles crossing an intersection, emulating a real world scenario. The solution is able to prevent a collision between two vehicles that are passing through the intersection. In addition to this, it performs optimization through minimizing control commands, i.e. acceleration/deceleration, and also moves as smoothly as possible.

The algorithm was developed with the goal of solving the intersection coordination problem robustly enough for the given scenario yet flexibly enough to be implementable on different intersection problems involving two vehicles. It is therefore structured in a way that its modules are as interchangeable as possible and allow for changes to be made without the need to make larger alterations to the program itself. These changes can include, but are not limited to, different sensors, vehicles, processing units and so on.

1.2 Problem statement

The main problem is to develop and implement a functioning solution for autonomous robots to travel from one side of the intersection to the other without colliding with another vehicle. As this problem is broad and complicated, it is divided into smaller tasks.

- Program the vehicles to move in a shape containing an intersection. It is important to be able to vary the speed of the vehicles along the path, so that the actual controller can later optimize the movements in a later stage.

- Measure and estimate the states of the vehicles around the intersection with the aid of sensors and accuracy-enhancing technology such as data filters. This sets a basis for sensing the positions of the vehicles as well as accurately controlling them in the intersection area. Robustness of the intersection coordination algorithm partially depends on the accuracy of the location of each vehicle.

- To have a central-computer that calculates as well as adjusts the course and speed of the vehicles near the intersection to avoid collision. This is done using the states such as location and speed measured in the previous task.

- Establish a communication link between the central-computer and the vehicles so that they can communicate to each other.

- Evaluate different types of algorithms being used by the controller and determine their characteristics.

- The final solution should be able to compute and perform 50 successful crossings with different starting parameters.

The final problem is to combine the solutions to all these smaller solutions into one system. The vehicle paths and positions will be pinpointed by the controller using the accurate location data. The controller will then be able to coordinate the movement of the vehicles in an optimal way and that avoids collisions. Once the risk of collision is over, the optimization demand is removed in order to expedite the run-time of the experiments. The controller should communicate these control commands to the vehicles over the communication link. Finally the robots should be able to implement these commands accurately and then give feedback from their new state to the controller to start the process over again.

1.3 Scope & Limitations

In the project, the focus is on two robots in a one lane intersection. The robots will have predefined paths that they should follow. The only requirement is that the path contains an intersection and that it is continuous. Therefore scalability will not be tested in this project.

UWB sensors will be used to get location data and accordingly the trajectory of each robot. There will however be measurement errors. Because of this a filter will be used to increase the accuracy. Other ways of locating the robot will not be used in order to simplify potential solutions and emulate a more realistic scenario.

The controller will optimize control commands sent to the robots to make the passing of the intersection smooth. As there exists uncertainties and randomness in the scenario, an optimization problem guaranteeing collision avoidance with an acceptable probability will be implemented. The acceptable probability is set 1% for 50 intersections, using solvable starting parameters. Several ma constraints are also placed on the robot to mimic comfort requirements, these being a maximum velocity of $1\,\mathrm{m/s}$ and a maximum acceleration of $1\,\mathrm{m/s^2}$.

1.4 Similar projects and their relevance

The project that was done here is a part of larger area of study both for the institution but also the wider world. Not only are there other bachelor projects that have developed other areas of automating the robots but there are a variety of papers that theoretically describe potential solutions for intersection control. These are quite advanced but provided significant inspiration for the types of controllers that were implemented and emulated in this project.

For instance, the two ideas presented in the report on how to build up the control algorithms were influenced by two papers [3] [1] which describes potential optimization of similar setups and the implementation of model predictive control in concert with these types of algorithms respectively. Additionally, setting constraints on the control action sets that are carried out by the robots and further along, the concept of guard distances in the controller algorithms was spawned by another paper on intersection control using a convex optimization, something well outside of the bounds of this report [4]. The way this paper differs from many papers of this type that from the onset have solved the problem of intersection control and in a much more efficient way is that these are highly theoretical whereas the paper at hand is dealing with an emulation of the system and all the challenges that can provide.

It is important to mention that a similar type of project has been attempted in the previous year at the electrical engineering institution that handled these robots and gave a solid base upon which this report builds upon, particularly in regards to the UWB sensor system and the ROS utilization [5].

2

Theory

The theoretical basis for the project is undoubtedly broad and covers different areas of kinematics, positioning, filtering and optimization. Ignoring the depth of these fields and the possibility of delving into all the minute details, the necessary knowledge for understanding the project is presented in this section.

2.1 Trilateration

Trilateration is a concept that is used in a variety of localization applications such as the global positioning system[6]. In the scenario presented in the project, the world that is being mapped is in two dimensions. To describe the concept of trilateration, a theoretical structure of how the method works is presented.

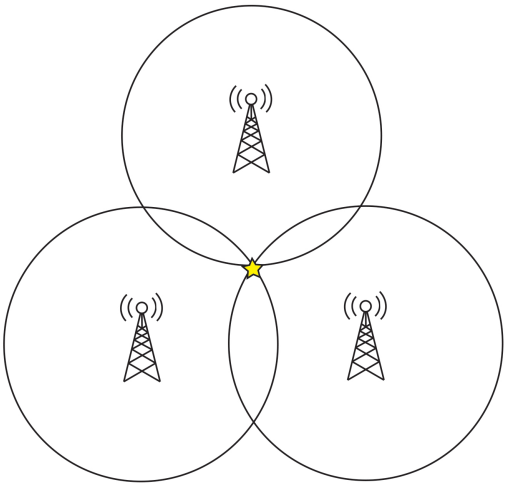
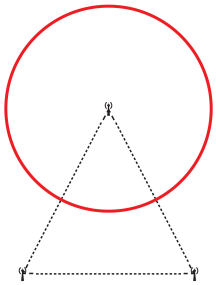


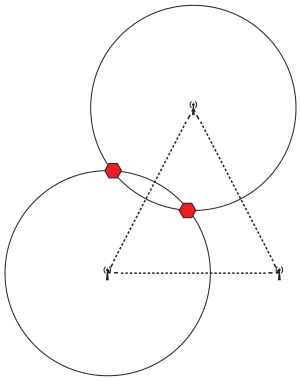
Figure 2.1: This figure describes how the distance of the point of interest from each anchor all cross to form a fixed location.

Suppose a scenario where there are at least three fixed anchors with known positions placed in the area where another object is also placed but whose position is unknown [6]. Assume that the object has the ability to accurately measure the distance

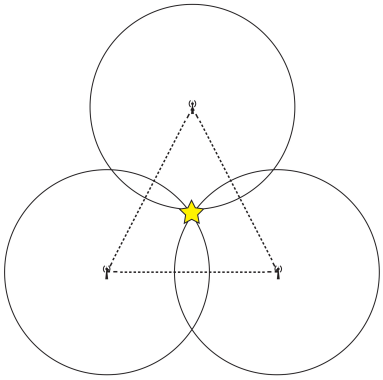
between itself and any known and fixed anchor in this area. This allows that object, using these measurements, to accurately determine where it is in relation to the anchors, whose position is known. This is shown in figure 2.1. In the figure 2.2a, the distance of the object is known but not the location as it can exist on infinitely many locations along the distance circle from the first anchor. In the next figure 2.2b, however, this uncertainty has been reduced to only two position where the two distance circles intersect. This is unless the two distance circles only intersect at one point. This, however, is considered unlikely enough to be a negligible concern. The location of the object is then made certain due to the intersection of all three distance circles resulting in only one possible location as shown in figure 2.2c. This technique in combination with gradient descent is what will be used to provide measurement data for the robot as it moves through the scenario.



(a) Single anchor active, infinite points possible



(b) Two anchors active and two potential points found.



(c) All three anchors active and point found

Figure 2.2: This figure depicts the stages in finding a location using trilateration starting with one anchor and moving up to three.

2.2 Gradient Descent

Due to potentially inaccurate data that fails to produce an exact result, it is necessary to have an algorithm that is able to find a best fit solution. The gradient descent algorithm can be used for this purpose.

Start by considering a problem P which describes a function that ideally needs to be minimized [7].

$$\mathrm{P} : \min(f_x), \tag{2.1}$$

where $x \in \Re^n$. It is assumed that this function is continuously differentiable in $\Re^n$ and therefore the gradient can be found for all points. Starting at a point x_k the gradient of the function at this point can be described as $\nabla f(x_k)$. The gradient descent is then described as follows:

$$f(x_{k+1}) = x_k + \lambda \cdot \nabla f(x_k). \tag{2.2}$$

This iterative process will then approach a local minimum of function f_x as long as $\lambda < 0$ and $\nabla f(x_k) \neq 0$. The minimum of the equation 2.1 is found due to the negative descent of the function. This means that when $\nabla f(x_0)$ approaches zero, the problem moves closer to its minimum x_k as long as the descent $\lambda \cdot \nabla f(x_k)$ maintains a negative value.

It is important to note that using this method and depending on the value of λ will make the function approach its minimum more rapidly at first. It will then become slower the more iterations are done, making the function efficient if the precision of the solution is not of the highest priority[8].

2.3 Extended Kalman filter

The Extended Kalman filter (EKF) is an extension of the Kalman filter that can be applied on non-linear state models[9]. After linearizing the state-space model, the EKF will act like a regular Kalman filter. Both the linearization and Kalman filter will be described later in this section.

When using a Kalman filter it is important to note that while using a model of a system to calculate its states, errors will occur due to external factors. It is therefore not possible to model a system with absolute certainty and it is this uncertainly that is called process noise. The purpose of the filter is to reduce the process noise that comes from predicting new states each iteration from the model of the system [10]. This can be done by using an observation about known states such as a position and the covariance, considered to be the error, of the observations together with the prediction.

This is graphically described by the figure 2.3 where it is possible to see that error is reduced due to the fact that it has been overlayed with the possible position and

noise of the observation. This gives the final position a significantly lower value for its noise than either the model prediction or the observation would have had on their own.

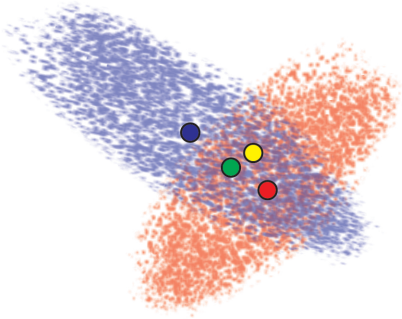


Figure 2.3: The blue dot and its accompanying error ellipsiod is the measured position and its associated noise. The red dot and haze represents the predicted position and its associated noise respectively. The green dot represents the estimated position after filtering this data, while the yellow dot represents the actual position.

In order to apply the equations corresponding to the Kalman filter, a linear model needs to be obtained. To linearize the model, the Jacobian matrix of the non-linear functions is derived at a linearization point. With the non-linear state-space differential equations on the form

$$\begin{aligned} x_1 &= F_1(x_1, ..., x_n, u_1, ..., u_m), \\ &\vdots \\ x_n &= F_n(x_1, ..., x_n, u_1, ..., u_m), \end{aligned} \tag{2.3}$$

where $x_1, ..., x_n$ are the states of the system and $u_1, ..., u_m$ are the inputs, the linearized model can be written as

$$\boldsymbol{x}_{k|k-1} = \boldsymbol{A}\boldsymbol{x}_{k-1|k-1} + \boldsymbol{B}\boldsymbol{u} \tag{2.4}$$

where

$$\boldsymbol{A} = \begin{bmatrix} \frac{\partial F_1}{\partial x_1}\big|_{x_s,u_s} & \cdots & \frac{\partial F_1}{\partial x_n}\big|_{x_s,u_s} \\ \vdots & \ddots & \vdots \\ \frac{\partial F_n}{\partial x_1}\big|_{x_s,u_s} & \cdots & \frac{\partial F_n}{\partial x_n}\big|_{x_s,u_s} \end{bmatrix}, \tag{2.5}$$

$$\boldsymbol{B} = \begin{bmatrix} \frac{\partial F_1}{\partial u_1}\big|_{x_s,u_s} & \cdots & \frac{\partial F_1}{\partial u_m}\big|_{x_s,u_s} \\ \vdots & \ddots & \vdots \\ \frac{\partial F_n}{\partial u_1}\big|_{x_s,u_s} & \cdots & \frac{\partial F_n}{\partial u_m}\big|_{x_s,u_s} \end{bmatrix} \tag{2.6}$$

[11].

The filter first creates a prediction of the new state $\boldsymbol{x}_{k|k-1}$ using the old state vector $\boldsymbol{x}_{k-1|k-1}$ and the linearized model of the system, the $\boldsymbol{A}$ and $\boldsymbol{B}$ matrices, as well as

the input signals, $\boldsymbol{u}$. The error covariance matrix $\boldsymbol{P}_{k|k-1}$ is also predicted using the same model, the previous error covariance matrix $\boldsymbol{P}_{k-1|k-1}$ and the process noise covariance matrix $\boldsymbol{Q}$. This is done according to the following equations

$$\boldsymbol{x}_{k|k-1} = \boldsymbol{A}\boldsymbol{x}_{k-1|k-1} + \boldsymbol{B}\boldsymbol{u}, \tag{2.7}$$

$$\boldsymbol{P}_{k|k-1} = \boldsymbol{A}\boldsymbol{P}_{k-1|k-1}\boldsymbol{A}^T + \boldsymbol{Q}. \tag{2.8}$$

After the state vector and error covariance matrix have been predicted, the Kalman gain will be used to correct the errors in the predicted state by adding the scaled difference between the observed states ($\boldsymbol{z}_k$) and the predicted states according to the equation

$$\boldsymbol{x}_{k|k} = \boldsymbol{x}_{k|k-1} + \boldsymbol{K}_k(\boldsymbol{z}_k - \boldsymbol{A}\boldsymbol{x}_{k|k-1}). \tag{2.9}$$

The error covariance matrix is also updated by taking the previously predicted error covariance and subtracting with itself scaled with the Kalman gain and the model matrix $\boldsymbol{A}$, resulting in

$$\boldsymbol{P}_{k|k} = \boldsymbol{P}_{k|k-1} - \boldsymbol{K}_k\boldsymbol{A}\boldsymbol{P}_{k|k-1}. \tag{2.10}$$

.

The Kalman gain is calculated as follows:

$$\boldsymbol{K}_k = \boldsymbol{P}_{k|k-1}\boldsymbol{A}^T(\boldsymbol{A}\boldsymbol{P}_{k|k-1}\boldsymbol{A}^T + \boldsymbol{R}_k)^{-1} \tag{2.11}$$

where $\boldsymbol{R}_k$ is the observed measurement's error covariance matrix which describes the inaccuracies in the observed states.

After the state has been updated according to the above calculations, it should output a more accurate state ($\boldsymbol{x}_{k|k}$) and its error covariance matrix ($\boldsymbol{P}_{k|k}$) than from the prediction or the observed state by themselves.

2.4 Optimal Coordination Problem

There are a variety of different control schemes that can be used to prevent vehicles from colliding when approaching an intersection. Each of these schemes have their advantages and disadvantages with the major factors being the robustness of the system, the efficiency of the system, and the comfort of potential passengers in a vehicle. [1]

2.4.1 Definition of robustness

The main purpose of the controller is to avoid collision. This goal falls under the concept of how robust the system is. While it might seem ideal to create a system that prevents collisions at all possible times, realistically this is not feasible due to constraints on both the maximum control actions, acceleration and deceleration, that can be taken by the cars.

For the purpose of this report, robustness is defined as the probability of a crash. Simply put the average number of intersection without a collision.

2.4.2 Definition of efficiency

Efficiency in the terms of intersection control have several aspects. Firstly, efficiency can refer to the control actions, which require power, taken by each vehicle and ideally minimize the total control actions taken. Additionally the efficiency can refer to the entire system's efficiency in regards to how rapidly the intersection is cleared.

For the purpose of this report, efficiency is only considered during optimized solutions and is defined as minimizing control actions.

2.4.3 Ruled based solutions

Many solutions use a clear set of rules to define how vehicles are able to move through the intersection [1]. These focus mainly on guaranteeing a clear and safe path for all vehicles through the intersections and can therefore sometimes neglect efficiency.

2.4.3.1 Reservation solution

The reservation solution is built upon the idea that both intersecting robots reserve time slots in which they are present at the intersection. By choosing non-overlapping time slots, the robots are able to be in the intersection at different times hence avoiding a collision. A controller is used to tailor the control actions of each vehicle in order to both enter and exit the intersection during each vehicle's reserved time slot. This system has the advantage of simplicity as both vehicles can have individual paths and control action set to make clearing the intersection as safe as possible. Another advantage of the system is that it can be easily and effectively scaled as each vehicle needs only to know its assigned time slot and how to get there.

2.4.3.2 Priority solution

The priority solution is where two vehicles approach an intersection and one is given priority over the other. For example, car A and car B approach the intersection. Car A enters first and has a certain acceleration and speed. Due to the lack of any other cars that could possibly create a collision, the controller takes no action. Car B then enters the intersection and queries the controller on whether or not the car can cross collision free. The controller then looks at this situation and determines whether or not a collision would occur. If so, car B will then brake or accelerate depending on constraints such as passenger comfort, limits on control actions, and efficiency. If collision is unavoidable with control actions on car B, the controller could then attempt a solution for Car A.

2.4.4 Optimization solution

These are solutions that find the most efficient control action set for all vehicles passing through the intersection. They have no rules but are able to calculate the solution to the problem resulting in the most efficient solution for sending both vehicles through the intersection without collision [1].

These solutions tend to be incredibly difficult to solve particularly with increasing amounts of vehicles in the problem requiring more processing power and longer computation times. This is not ideal in the small window for computation that is afforded in an intersection control problem [4].

2.4.5 Hybrid solutions

There are solutions that use rules to build a solution for the vehicles to follow but then optimizes how each vehicle completes its assigned solution. This has the advantage of possibly being more efficient than a strictly rule based system but is not as difficult to calculate as a purely optimization based approach. This is the approach we follow in this project.

3

Equipment

In this chapter the equipment used is presented and described. Overall this equipment is modern and relevant with the possibility of other more advanced or complex systems replacing them if the experiment were to be scaled up. The equipment is classified into two major categories, hardware and software, in which it is the software where the majority of the work in the project is conducted.

3.1 Hardware

In this section the hardware used for the project is presented. This includes hardware for positioning, the robots themselves as well as standard equipment such as computers and routers.

3.1.1 Ultra-wideband range sensor

The UWB ranging sensoring is a system used for measuring the distance between two points using a wide radio frequency band. This sensors are designated the PulsON 400 Ranging and Communications Module and are made by the United States based company TimeDomain [12]. They feature a built-in specialized high frequency chipset, ethernet NIC port, and ultra-wideband transceiver antenna. This allows for localization, limited radar and communications for a wide variety of applications. For this project, the localization aspect of the unit will be used which accomplishes this via a Two-Way Time-Of-Flight method [13]. The base will start by transmitting a range request to one anchor, which is a fixed sensor in the room. This request will be a wave-form pulse that propagates out from the antenna in all directions[14]. The base then starts an internal timer on time T_1. The anchor, after listening for such a pulse, will then reply with a wave-form back to the base which will arrive at time T_2. The total time between send and receive will be T_r as shown in the equation

$$T_r = T_2 - T_1. \tag{3.1}$$

Assuming the medium the wave travels through is air, the speed of the wave is extremely close to the speed of light in vacuum c_0. The distance d traveled can then

be calculated by

$$d = \frac{c_0 \cdot Tr}{2} \tag{3.2}$$

where you divide by 2 because the wave has traveled both to the anchor and back.

Using these times, the base can calculate the distance traveled by the wave, leading to a measurement accuracy of 3.5cm with line of sight, effectively. It is important to note that for this project, the measurement units used were in varying stages of performance due to their age and could have affected the accuracy of the location data, but this is disregarded. Newer versions of these antennas have an increased accuracy of up to 2 cm with line of sight [15].

3.1.2 Wireless Communication link

The fact that the project utilizes multiple robots necessitates the need for a data link between these robots and the controller. To cover this, a Wi-Fi router is used. Wi-Fi is well within the requirements of this project giving low latency and high data rate with acceptable stability [16]. It is this network that allows the Robot Operating System (ROS) to function across several different computers and consequently different robots simultaneously.

3.1.3 Pioneer 3-DX

The robots used for this project are Pioneer 3-DX mobile robots [17]. These are compact, robust, wheel mounted robots that are suitable for emulating vehicles in the constrained testing environments used for this project. They have two drive wheels giving two degrees of freedom with a third wheel to give stabilization that has a free floating axis. It additionally has a front and side facing sonar system but these are not used in this project.

3.1.4 Computers

In order to connect all the various pieces of equipment and control them, several laptops (Dell Latitude E5450) where used running Ubuntu Linux version 14.04 or 16.04 *Trusty Thar* [18]. Each robot has an on board laptop on board coordinating various measurements and control commands. There is another laptop that serves as the remote controller, which runs all of the coordination software.

3.2 Software

The software package used is called ROS and a variety of programs coupled with this operating system.

3.2.1 ROS and ROSARIA

ROS is an operating system that allows the smooth operation of each robot and communication between them and the main controller [19]. This is a highly flexible system allowing the smooth flow of information between many decoupled entities, subsequently called nodes. It is licensed under the Creative Commons Attribute 3.0 [20]. Communication between the Pioneer robots and ROS is handled using a library called ROSARIA. It merges the robot's manufacturer's *Advanced Robot Interface for Applications* software development kit, which allows the direct control of the robot's systems with the ROS. This allows for high level commands to be performed by the robot such as velocity and angle changes. ROS has multiple packages allowing for the operating system to run different programming languages such as C++, Lisp, and the language which this project is programmed in, Python. In order to run the latest stable version of ROS, *Kinetic Kame* Linux is required and can be run through either an Debian or Ubuntu operating system, the later of which is used as specified earlier. ROS can be found on `ROS.org` and the ROSARIA library can be found and installed from here [21]. Without ROSARIA, running the robot would be significantly more difficult and would require directly controlling all of the robots systems, for example its wheel motors.

To give the reader a basic understanding of the implementation of ROS and subsequently the software behind the controller, a brief overview of key concepts are contained below. This have been primarily sourced from the ROS Wiki which contains all the documentation used in this project and has been the primary guide for the utilization of ROS. The sections below and further on in the paper refer to this Wiki.

3.2.1.1 Nodes

ROS divides up its active functions into packages called nodes. These nodes are what can be considered the "face" of a certain set of programs in the larger ROS network. For instance a node may be programmed to accomplish a certain goal, such as driving a robot, and as such will handle all the code devoted to fulfilling that function. It will also communicate with other nodes for information on such things as positioning data, desired control actions, or status updates that the other nodes either give or require for their continued function. In this way the code structure of ROS becomes highly distributed allowing nodes to be added and removed as necessary for the project. All of these nodes are held together by the *roscore*. This is the master program that orchestrates the running of all the other nodes to prevent conflicts between them. It is this flexibility that makes ROS useful as a coding structure, especially for a project such as this with multiple computers running different equipment.

3.2.1.2 Topics and Services

Topics and Services are predefined methods used by ROS nodes to communicate via the ROS master, *roscore*, or with each other. The services and topics, which are both used in the project, operate in slightly different ways.

The Service is a function that is set up and connected to the node that initializes it. As long as the service is not called, the function will not be executed. If another node needs to use this service and is located on the same master, it can call on it to execute. Usually some variable or other data is used in combination with calling on a service. The service will then execute the function and optionally return some data depending on the function. To specify what data needs to be transferred with each service call, a service file (*.srv*) will define the variables that are sent with each call as well as the variables that will be returned.

Topics are similarly used by two or more nodes where one of the nodes is a publisher. The publisher will send data onto this virtual data bus that can be accessed by all other nodes on the same ROS master. Other nodes can then subscribe to the relevant topics. When data then is published, the subscriber node will use the data in combination with a callback to execute a defined function. The data published on these topics are called messages and are defined with the message files (*.msg*). Topics are unidirectional, meaning that a node does not usually publish to the same node it subscribes to.

3.2.1.3 *RosAria/cmd_vel*

In ROSARIA, a predefined topic is created with each RosAria node, called */cmd_vel*. The topic is subscribed to by the robots, and allows other nodes to send messages of the type Twist, containing control commands in the form of linear speeds (x, y, z) and rotational speeds (x, y, z) to the RosAria nodes. The message used additionally contains the possibility to send covariances of the processes that are being performed to give a greater detail of the accuracy of the robot's actions. The RosAria node translates all published messages on the topic into low-level movement-commands that can be understood by the robot.

3.2.1.4 Odometry

Similar to setting up the topic */cmd_vel*, the RosAria node will also create an Odometry (*RosAria/odom*) topic where it will publish information about the current states of the robots in the form of Odometry messages. These messages include data about the robot's position, directional angle, speed and angular speed etc. The Odometry message type sent on this topic is a combination of Pose and Twist messages. Twist, as described earlier, contains data about the robots control actions. The Pose messages are calculations of the robots state made from the wheel rotations. When first set-up, the position (x- and y-coordinate) as well as the angle of the robot is 0, meaning the robot starts in the origin facing the x-axis.

3.2.1.5 MobileSim

To be able to accurately simulate and test the systems and algorithms used in this report, MobileSim was used. MobileSim is a simulation application that works together with RosAria and allows RosAria to be run in the same way as outside the simulation [22]. If a MobileSim process is running when a RosAria node is started all commands will be sent to the simulation instead. It is also possible to run several robots in the same simulation, using separate RosAria nodes.

3.2.2 Python and Rospy

Python is a free and open source programming language. Together with the Rospy packet, Python can be used to create programs that control the ROS nodes or read information published on the various topics. This can be for instance reading the Odom messages, or writing twist messages, etc.

Other additional packages are available for Python that can be used to solve matrix equations for the extended Kalman filter and also to solve trigonometric equations and optimization problems and other similar mathematical problems.

4

Methods

At the start of the project, the core problem was broken down into smaller tasks. An iterative mindset was then used throughout the project to tackle these issues one by one until they were solved and combined to form a solution to the final problem.

The first step was to form the most basic solution to the problem and see what was necessary to solve it. This was determined to be two robots simply crossing each other without collision, i.e. no demands on acceleration. In order to make the robots cross, it was decided that two things were required: knowledge of their positions and being able to give them general commands. After retrieving knowledge of their positions through the UWB sensors, and sending commands through ROS and RosAria, it was decided that a filter was needed to improve the positioning data. At the same time, more sophisticated algorithms were developed to not only send general commands, but to implement the control algorithms presented in section 2. In this section, these implementations will be presented in greater detail as well as the methods used to test the entire system.

4.1 Initial Setup

The wireless communication system is used to connect the controller with the robots. Over this connection, ROS is able to operate and send information including location and state data about the robots to the controller as well as control commands from the controller to the robots.

In order for the controller and ROS to communicate with each node, a simple router (Thomson TG784) was used as a base station. This router requires Internet connection and is used to enable the devices to communicate with the master over the WLAN, and vice versa[1]. To allow the nodes to communicate with the base station and the *roscore* of the controller, they need to be told the IP assigned to the controller as well as the host name of the ROS master.

[1] A mobile hotspot was used in place of the router because it lacked internet access.

4.2 Determining the state of the robots

This section describes the steps taken to find the state, primarily position and velocity, of the robots at any given moment during execution.

4.2.1 Robot state

The state of the robot in any given moment in time is made up of its two-dimensional position (x- and y-coordinates), the angle of the robot in relation to its original orientation θ as well as both its linear velocity v and angular velocity ω. These states are visualized in figure 4.1

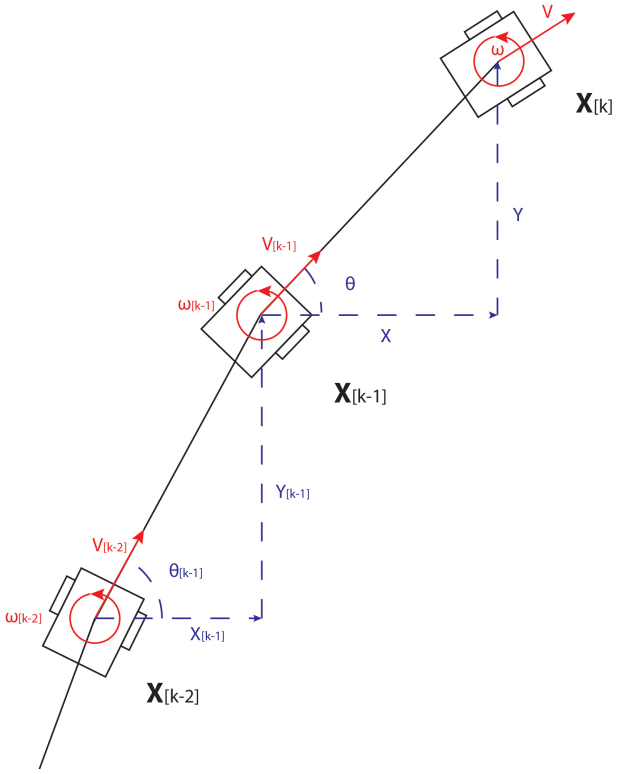


Figure 4.1: The figure shows the coordinate system and angle of the robot, which are used to create a model of movement. It also shows the iterative states of the system from $\mathbf{X}_k$ to $\mathbf{X}_{k-2}$.

The equations that describe the dynamics of the system from any moment $k-1$ to k, are

$$x_k = x_{k-1} + v \cdot \cos(\theta_{k-1}) \cdot dt \tag{4.1}$$
$$y_k = y_{k-1} + v \cdot \sin(\theta_{k-1}) \cdot dt \tag{4.2}$$
$$\theta_k = \theta_{k-1} + \omega \cdot dt \tag{4.3}$$

where dt is the time step between each iteration of the filter. This state connects

the robots linear and rotational velocity to the robots x and y position as well as orientation, making sure that all measurable attributes are accounted for.

4.2.2 Setup of components for determining the state

The robot's position will be determined by using a sensor grid that is comprised of three anchors, each with a UWB radio transceiver that will be pinged by a similar device on each robot, allowing the robot to gather a time of flight reading from each anchor. With these time of flight readings the sensor on each robot will determine the distance to each anchor in the grid. These measurements will be used to determine the position of the robot, which will be discussed in the section below. However these measurements are not perfect and therefore require estimation algorithms and a filter to be of a usable quality for the robot and the controller.

A Kalman filter is used in order to make the measurements of use to the controller. As described in section 2.3, it predicts the robot's next possible state using a model and then fixes this model using measurement data for use in predicting the next state. This allows for much more accurate location data that can be sent to the controller for use in solving the intersection problem.

4.2.3 Implementation of trilateration

In order for the position of the robots to be used, the data needs to be in a reference system which in the case of this project is an x- and y-coordinate system. This is possible to accomplish using a minimum of two anchors, however the system implemented here uses three, one as the origin of the system, and the other two measuring out the x- and the y-axis respectively. This is due to the fact that for the robot to use trilateration to determine its position in this coordinate system, it must have three points of reference to refer to.

An issue with the trilateration as it is described in the theory is that it requires all three distance circles to overlap perfectly in one point in order to gain a location. When the robot pings an anchor and gets back a measurement, this measurement has an inherent amount of noise. This can result in the possibility of overlapping circles creating a triangle of uncertainty or worse still, the circles do not overlap at all. This is described by the figure 4.2.

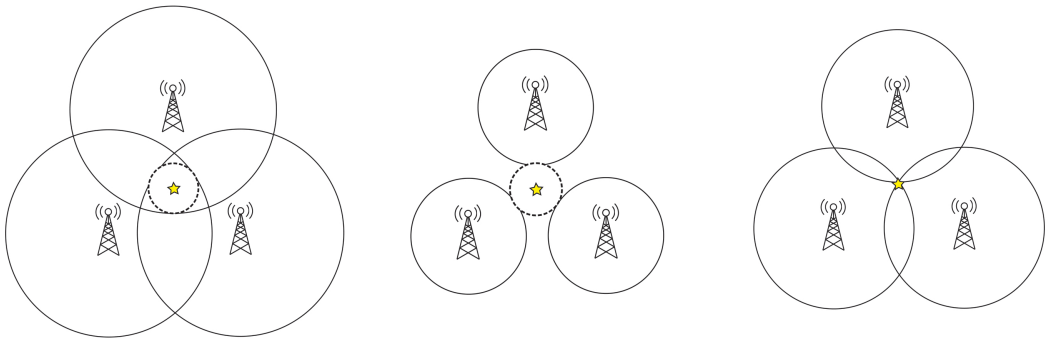


Figure 4.2: The leftmost subfigure describes the possibility of overlap causing a residual error. The middle one describes the opposite with zero overlap. The rightmost is the ideal case with perfect overlap

It is therefore important to reduce this residual that is left between the distance circles from each anchor. This is done using an iterative process that takes the previously measured point and moves this reference point such that it is reasonably close to a center point between the distance circles. The equation of this iterative process is

$$f(\vec{x}) = \sum_{i=0}^{N} (\vec{a_i} - \vec{r_i})^2 \tag{4.4}$$

where the $\vec{a}$ vector describes the distance between the anchor and its measured point and the $\vec{r}$ vector describes the vector between the anchor and the reference point that is being moved in each iteration to decrease the value of the function. This is described in the figure 4.3.

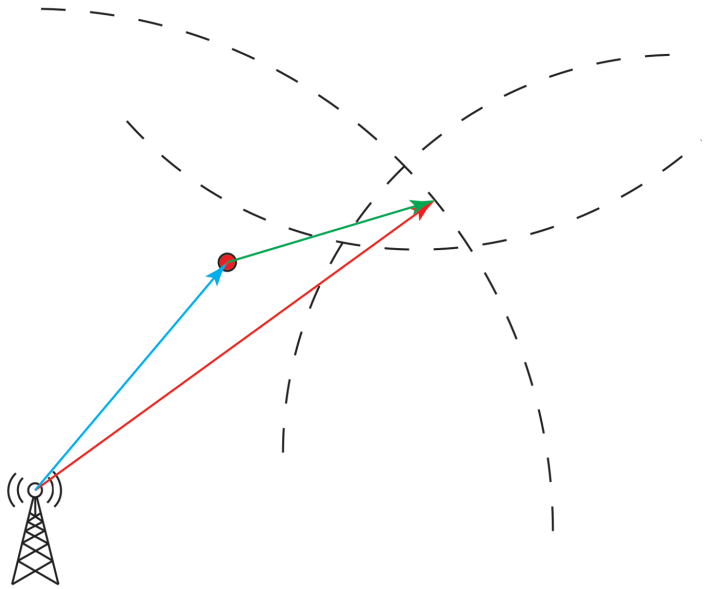


Figure 4.3: This figure describes the following procedure for one of the sections of the sum for the total residual. The red vector is the equivalent of $\vec{a}$. The light blue vector is the $\vec{r}$ vector and the green vector describes the vector whose magnitude is $f(\vec{x})$ or the residual of the system.

Unfortunately the value that is given by the UWB radio antennas is only a magnitude, not a vector so the vector $\vec{a}$ must be rewritten to use its magnitude instead. This is done by the following equation that puts $\vec{a}$ in terms of $\vec{r}$

$$\vec{a} = \frac{\vec{r}}{|\vec{r}|}|\vec{a}|. \tag{4.5}$$

Following this, the principle of gradient descent is used to minimize the value of the residual. However in order to optimize the runtime of the program, the derivation of the gradient at each time step was abandoned and the gradient was determined to be a function of the length of the residual vector divided by 2. The program starts by taking the previous point of measurement and then using that as the reference point. Then by using the methods described above, this point is drawn toward the place that minimizes the residual of the function.

This function would then continue to run until the residual would satisfy

$$f(\vec{x}) < A, \tag{4.6}$$

with A being a chosen tolerance. To prevent a problem where the residual never met A due to exceptionally bad spacing between the distance measurements, the program would also see if

$$f_{k-1}(\vec{x}) - f_k(\vec{x})) < R \tag{4.7}$$

was satisfied, where R was another chosen tolerance and both $f(\vec{x})$ were the residuals between the previous step and the current one. This allowed the system to determine an accurate location regardless of how uncertain the measurements were. The figure 4.4 visually represents this system and how it works through each iteration of the descent function.

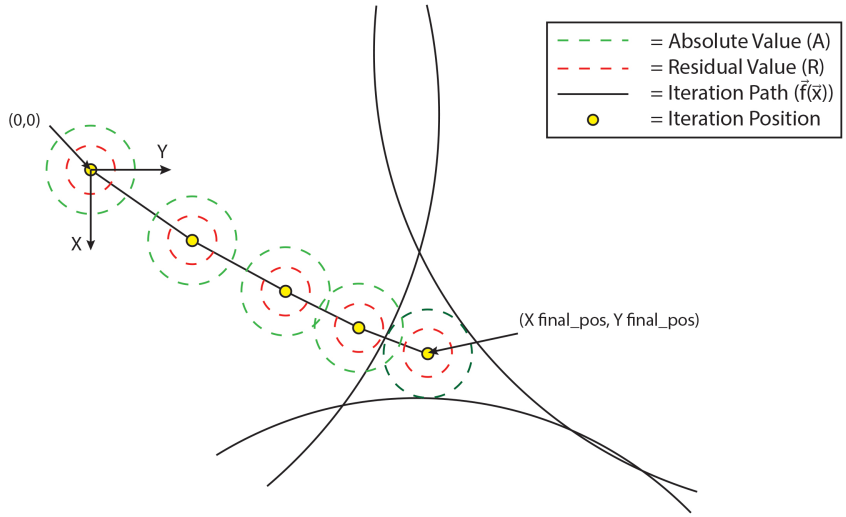


Figure 4.4: This figure describes how the criteria are met for the tolerances of the residual in the descent function. In the case above the chosen tolerance A was met before the secondary criteria R was met.

4.2.4 Implementation of the extended Kalman filter

Despite the optimization process for the trilateration to find the robot's position, the measurement data is still noisy and of little use for the controller in accurately modeling the intersection. This noise modeled in a covariance matrix in section 5.1. To mitigate this noise and make the location data usable, a Kalman filter has been implemented. How this filter is generally built is described in section 2.3.

In order to apply the filter, the equations describing the robots movement (section 4.2.1) need to be linearized in order to obtain the $\boldsymbol{A}$ matrix that describes the model of the system. The calculated matrix is

$$\boldsymbol{A} = \begin{bmatrix} \frac{\partial x_k}{\partial x_{k-1}}\Big|_{x_s,u_s} & \frac{\partial x_k}{\partial y_{k-1}}\Big|_{x_s,u_s} & \frac{\partial x_k}{\partial \theta_k}\Big|_{x_s,u_s} \\ \frac{\partial y_k}{\partial x_{k-1}}\Big|_{x_s,u_s} & \frac{\partial y_k}{\partial y_{k-1}}\Big|_{x_s,u_s} & \frac{\partial y_k}{\partial \theta_{k-1}}\Big|_{x_s,u_s} \\ \frac{\partial \theta_k}{\partial x_{k-1}}\Big|_{x_s,u_s} & \frac{\partial \theta_k}{\partial y_{k-1}}\Big|_{x_s,u_s} & \frac{\partial \theta_k}{\partial \theta_{k-1}}\Big|_{x_s,u_s} \end{bmatrix} = \begin{bmatrix} 1 & 0 & -v \cdot \sin(\theta_{k-1}) \cdot dt \\ 0 & 1 & v \cdot \cos(\theta_{k-1}) \cdot dt \\ 0 & 0 & 1 \end{bmatrix}. \quad (4.8)$$

The prediction of the new states can be calculated using the non-linear equations in 4.1. In addition the $\boldsymbol{B}$ matrix is not an input in any further steps in the filter and is therefore not needed to run the Kalman filter.

In this implementation, the filter uses the previous state of the robot, control actions sent from the controller (the linear and rotational velocity), current measurement data (x- and y-position) along with noise values for all of these inputs to estimate the current state of the robot in the emulation. The process noise w_k can be assumed to be zero mean Gaussian noise where $\boldsymbol{w}_k \sim \mathcal{N}(0, \boldsymbol{Q})$ and the covariance matrix

$$\boldsymbol{Q} = \begin{bmatrix} 0.05^2 & 0 & 0 \\ 0 & 0.025^2 & 0 \\ 0 & 0 & 0 \end{bmatrix}. \tag{4.9}$$

Similarly, the observed state also contains zero mean Gaussian noise $\boldsymbol{v}_k \sim \mathcal{N}(0, \boldsymbol{R})$ where

$$\boldsymbol{R} = \begin{bmatrix} \mathrm{Var}(x) & \mathrm{Cov}(x,y) & 0 \\ \mathrm{Cov}(y,x) & \mathrm{Var}(y) & 0 \\ 0 & 0 & 0 \end{bmatrix}. \tag{4.10}$$

The covariance values are based on a number of n readings where the covariance matrix is calculated from the sample measurements, see section 5. Lastly, the observed state vector $\boldsymbol{z}_k$ and the observation matrix $\boldsymbol{H}$ are

$$\boldsymbol{z}_k = \boldsymbol{H}\boldsymbol{x}_k + \boldsymbol{v}, \tag{4.11}$$

$$\boldsymbol{H} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{4.12}$$

where $\boldsymbol{x}_k$ is a vector containing the observed x and y coordinates from the UWB sensor grid. Using all these equations together as explained in section 2.3 the states that are calculated in the filter should be more accurate due to a decrease in total uncertainty and help the robots determine a more exact position.

However there is no observation made on the angle of the robot, so the resulting state will therefore be purely based on the prediction made in equation 4.1. The robots angular velocity will continuously and non-linearly increase/decrease between the given values, necessitating that the resulting angle be calculated according to a Riemann sum, see figure 4.5, with width dt, decided from the iteration time of the filter. This will therefore deviate from the actual angle, which is the area under the angular velocity curve. To minimize these errors, only three specific angular velocities are used (depending on the angle offset) to turn the robot rather than a regulator. The resulting method, visualized in figure 4.6 contains less errors due to these more discrete angular velocities and will lead to a more accurate prediction. Despite the discrete angular velocity inputs, the system is not instantaneous and has some lag resulting in the error seen in the visual representation below in figure 4.6

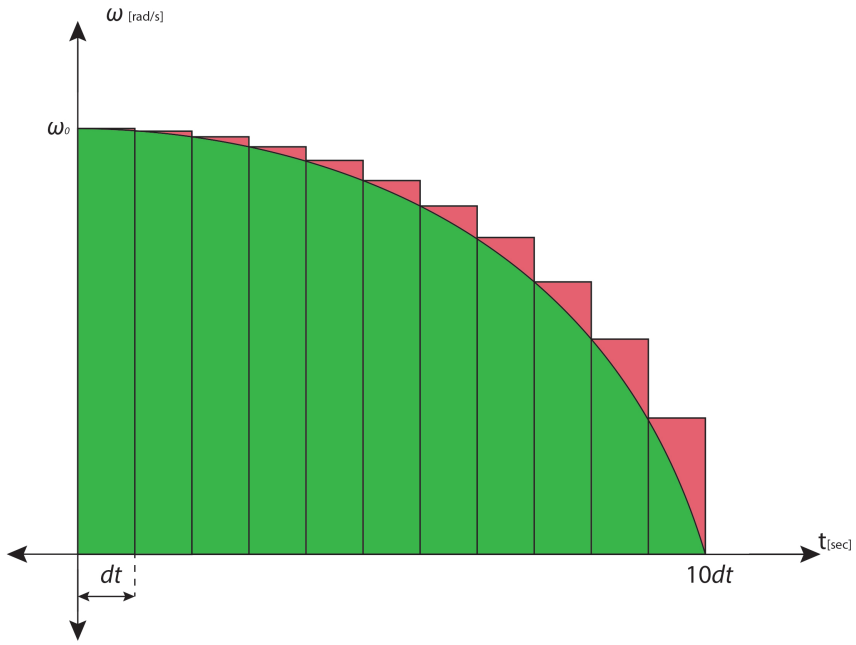


Figure 4.5: Riemann sum when using continuous angular velocity changes (according to a regulator). The area under the curve represents the actual angle of the robot, while the rectangles represent the predicted angle.

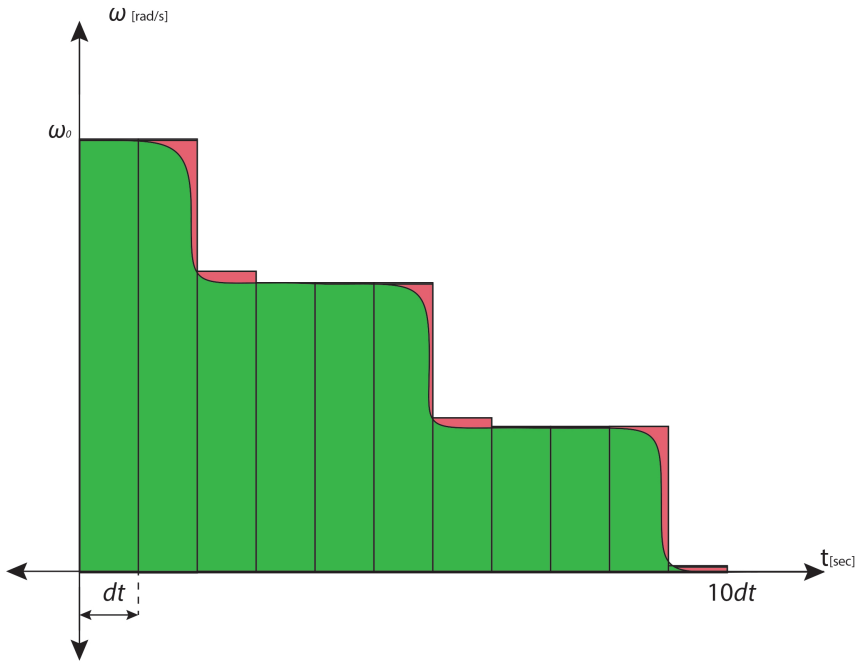


Figure 4.6: Riemann sum when using three specific angular velocities. The green curve represents the actual angle of the robot where the rectangles are predictions of the angle.

The filter was tested by moving one robot in a known path. The observed position, as well as the predicted position and filtered positions are plotted out, and the data is observed to see if the filter manages to correct the location as the robot moves along the path. The results are presented in section 5

4.2.5 Generating noise for Pose-data

For various reasons, including testing changes in the algorithm, the problem is simulated in MobileSim as described in section 3.2.1.5. In order to more accurately simulate the real world scenario, where the positioning system has noisy measurements, an option for adding noise to the Pose-data used to simulate the robots was created. The noise is Gaussian in nature with the user specifying the standard deviation. Since the UWB sensors have an estimated error of around 2 centimeters, a similar value was chosen for the standard deviation for the setup used to most accurately emulate the real scenario.

4.2.6 Calculating an intersection point

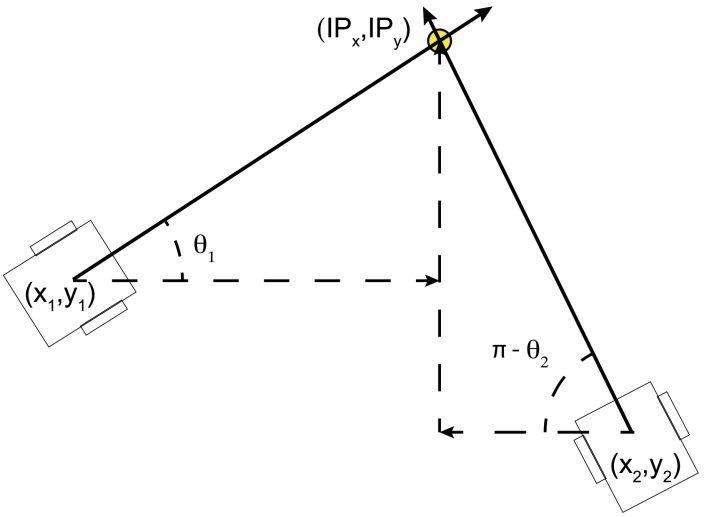


Figure 4.7: The figure shows an example of the robots, their position and angle data, and their resulting intersection point. x_1, y_1, θ_1 describes the data from the first robot, x_2, y_2, θ_2 is for the second robot and IP_x and IP_y is the intersection point.

In an ideal scenario the intersection point of the robots will be in the origin of the coordinate system. Given that there is always errors for the robots as they travel and try to aim themselves, a more general approach was taken. This approach was to calculate the intersection points of both the robots using their positions and angles.

Figure 4.7 shows the general situation and the parameters that the calculation uses, those being the first and second robot's position and angle, x_1, y_1, θ_1 and x_2, y_2, θ_2.

Using this information the lines describing the trajectory of the robots can be found. Considering the first robot, its slope will be $\tan(\theta_1)$. The equation describing the line is then $y = \tan(\theta_1) \cdot x + m$. The constant m is found by using the robots position data, resulting in $m = y_1 - \tan(\theta_1) \cdot x_1$. This gives the equation for the line in the form of

$$y = \tan(\theta_1) \cdot x + (y_1 - \tan(\theta_1) \cdot x_1). \tag{4.13}$$

Using the same method, the equation describing the line for the second robot is

$$y = \tan(\pi - \theta_2) \cdot x + (y_2 - \tan(\pi - \theta_2) \cdot x_2). \tag{4.14}$$

Setting these two equations equal to each other and solving for the x values, will give the intersecting point IP_x. The result is

$$\mathrm{IP}_x = \frac{y_2 - y_1 - \tan(\pi - \theta_2) \cdot x_2 + \tan(\theta_1) \cdot x_1}{\tan(\theta_1) - \tan(\pi - \theta_2)}. \tag{4.15}$$

Putting this x value in one of the equations describing the trajectories of the robots will give the corresponding value IP_y. This becomes

$$\mathrm{IP}_y = \tan(\theta_1) \cdot \left(\frac{y_2 - y_1 - \tan(\pi - \theta_2) \cdot x_2 + \tan(\theta_1) \cdot x_1}{\tan(\theta_1) - \tan(\pi - \theta_2)} \right) + y_1 - \tan(\theta_1) \cdot x_1, \tag{4.16}$$

and together with IP_x gives the intersection point.

4.3 ROS structure and data handling

The ROS structure for this project is divided into several different packages that run on different computers on different robots. Roughly speaking, ROS runs in three physical locations, the master computer running the main controller that organizes the intersection, and two slave computers, each running separate versions of the program that handles location data, and the program controlling the robots control actions (RosAria). Each of these computers run a node or combination of nodes that accomplish various tasks. Together these nodes work together to send information from robot to controller and allow the emulation of the intersection control. This code structure is explained in greater detail in the figure 4.8.

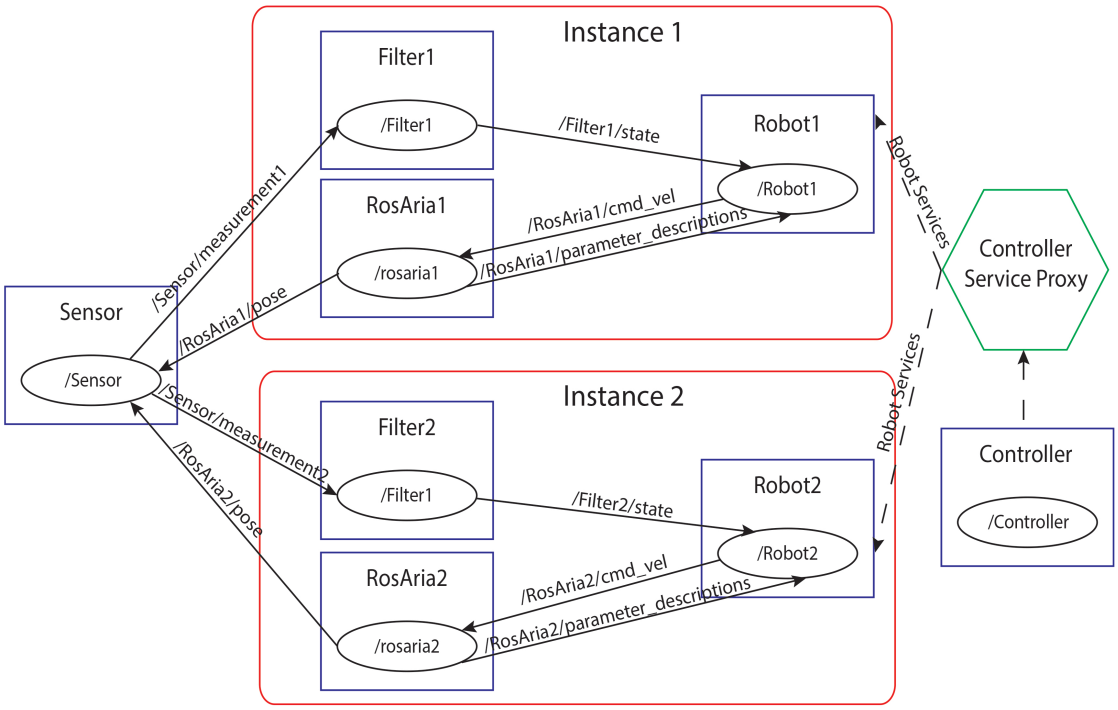


Figure 4.8: A figure modeling the basic structure of the code in the program that runs the robots, location system, and intersection control algorithms. The blue boxes represent ROS nodes, the circles inside these represent topics inside each of these nodes while the solid lines represent publisher and subscriber relationships between these different topics. The dashed lines represent services and the hexagon shape represents a service proxy that creates and handles these services.

The simplest way to describe the flow of the code is to start from the controller and go through the system step by step. The figure above will be referred to give a better idea of how everything connects. The controller sends commands to the rest of the program via services that are handled via the Service Proxy. These allow directions to be easily communicated to each robot since publishing to a topic is unnecessary for these command type messages. These service messages are then taken by the robot node which then interprets these commands and publishes in a topic what both the filter node and RosAria node need to know to fulfill their objectives. The interpretation that the robot node does is detailed in the following section.

Once the interpreted commands are sent to RosAria, the robot needs to update the controller with information on where it is or in other words, the robots state at that current time. To accomplish this the filter function, which includes the Kalman filter, builds up a topic for the state. It requires data however to compute the state of the robot and this is taken from the sensor node. This node takes two different types of information, location data from the UWB radios and linear and angular velocity from the telemetry module on board the robot. This telemetry data comes from the RosAria node.

Once all this data has come together, been filtered and sent from the robot to the controller, the controller can then use this new location data to create new control actions for the robot to follow. It is this loop that allows the robots to accurately navigate both the scripted paths but also the intersection crossing with help of algorithms detailed in section 4.5.

4.4 Communicating instructions to the Robot nodes

Table 4.1: Table defining the different missions the robot is able to execute.

Basic	Angular	Complex
Stop	*Aim at Point*	*Go to Point*
Set velocity		*Follow Path*
Set angular velocity		*Steer towards point*
Set acceleration		

Each robot is able to move around using the RosAria code but this requires an input of both a linear and angular velocity. The controllers however work using more high level concepts such as a *Go to Point*-function. The Service proxy and Robot node are the programs that are used to translate these high level commands to the inputs that RosAria needs to move the robots.

The handler states all the different types of commands that can be outputted by the controller and translated for RosAria. These commands are called in this project, missions and there are six in the program that fulfill roughly all the necessary motions for the controllers to accurately and safely get the robots to navigate an intersection.

The possible missions are described in Table 4.1. A full API can be found in Appendix A.3.1 of this report. The following sections (4.4.1, 4.4.2 and 4.4.3) will give a short overview of all the missions and short description of how to use them.

The simplest and most universal of the missions is the *Stop* mission. This mission will end any currently active mission and make the robots movement come to a stop.

4.4.1 Basic movement missions

The first three missions define the most basic instructions you can send to the robot. They let you freely maneuver the robot using any of the defined methods for the robot. Each of the missions are called with a float-variable as the argument, denoting either the speed or acceleration that the motion should be executed with.

Setting either acceleration or speed is exclusive of the other. If an acceleration is set the robot will move with that acceleration, regardless of whether a speed was set previously. The same does not apply to the angular velocity. Angular movement

can be combined with any of the other basic missions and the robot will execute a combination of the movements asked.

4.4.2 Angular control mission

Currently there is only one mission solely controlling the angle of the robot. This mission takes a point as an argument and rotates the robot so as to face the point.

As described in figure 4.5 and 4.6 and at the end of section 4.2.4, the predictions of the angle gets better if discrete steps for the angular velocity is used. As such the *aim at point* mission used three different velocities depending on how large the difference in angle is between the robot and the point to be aimed at.

4.4.3 More complex movement missions

These missions build upon the previous ones and let the controller ask for more complex actions of the robot.

The *Go to Point* mission is the simplest version of asking the robot to move to a point.

Next the *Follow Path* mission is an abstraction over several *Go to Point* call. Given a Path (a list of points) it will iteratively call *Go to Point* for each point one after another.

The last complex mission, *Steer towards point* is used to be able to abstract away angular movement on a path between two points, as such allowing the travel to be handled as one-dimensional. After calling this with a point, the robot will continuously and automatically orient itself toward as to follow the path between it's current position and the point. After this the user is free to call the previously described missions *Set velocity* and *Set acceleration* to move the robot over the path.

4.4.4 Polling robot for state

Besides the previously defined missions there are two other services provided by the Robot node. One lets you ask whether the robot is executing a mission currently and the other lets you ask for the current positional and dynamic state of the system.

4.5 Description and function of controller algorithms

In this section the different implementations of controller algorithms are presented. The one used to gather the final results is the hybrid version encompassing the priority and optimization solutions.

4.5.1 Implementation of reservation solution

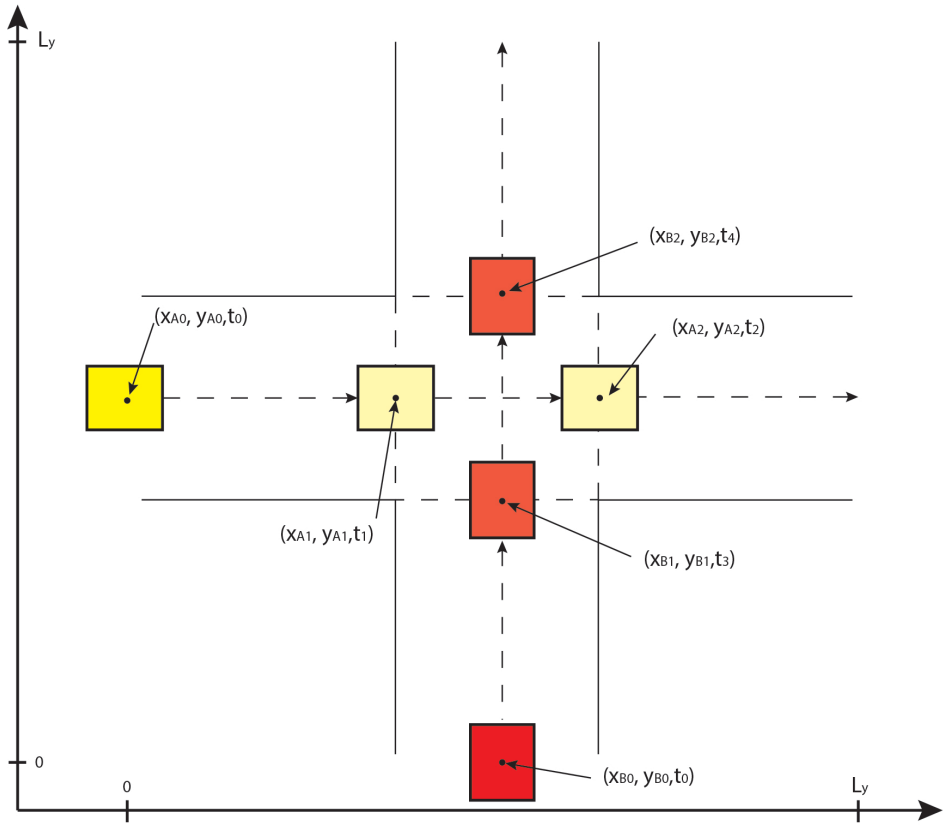


Figure 4.9: The figure depicts the two intersecting robots at different positions and times throughout the intersection, matching the time slots chosen by the user. In this case, both the robots are able to accelerate and decelerate and in general act more independently.

Table 4.2: The table shows the values specified by the user for the reservation based system. These values are in the form of time slots, intersection values and the total length of the travelled path.

User input for reservation solution								
Time slots	t_1	t_2	t_3	t_4				
Intersection values	x_{A1}	y_{A1}	x_{A2}	y_{A2}	x_{B1}	y_{B1}	x_{B2}	y_{B2}
Total length	L_x	L_y						

As explained in section 2.4.3.1, the reservation solution makes sure the robots are in the intersection at certain times. The chosen implementation of this system requires the user to specify three forms of parameters: the different time slots for the robots, the general intersection parameters i.e. when it starts and ends, and also the total length. The distance from the starting point to the intersection and from the intersection to the ending point also needs to be specified. Figure 4.9 and table 4.2 shows the required values, with the table showing the time slots, intersection values and total size that the user inputs. The t, x or y denotes whether the parameter is for time or position with corresponding coordinate axis. A or B is given depending on the robot and 1 or 2 represents the start and the end of the intersection respectively. The time slots t_1 and t_2 belong to robot A while t_3 and t_4 are for robot 2. The intersection itself is represented by the dashed lines in the figure. The total length of the path is L_x for the x-axis and L_y for the y-axis. The values x_{A0}, y_{A0}, x_{B0} and y_{B0} do not need to be specified by the user.

The algorithm contains two instances of the same code that run simultaneously; one for each robot. The algorithm first calculates the necessary velocity for a robot to reach the start of the intersection at the specified start of the reserved time interval. For example if we regard robot A, the yellow robot in figure 4.9 traveling in the x direction, the velocity sent to the robot would be

$$v_A = \frac{\sqrt{(y_{A1} - y_{A0})^2 + (x_{A1} - x_{A0})^2}}{t_1}. \tag{4.17}$$

This velocity needs to be smaller than 1.0 but greater than 0, because of the fact that the robots cannot travel faster than 1 meter per second due to comfort restraints placed on the vehicle and since we do not want negative velocity. If a velocity is calculated that does not meet those requirements, the user needs to input new time slots or intersection values.

At the same time that the velocity is sent to the robot, a timer is initiated. After t_1 seconds has passed, the robot has arrived at the start of the intersection. An acceleration is calculated by

$$a_A = \frac{2\left(\sqrt{(y_{A2} - y_{A1})^2 + (x_{A2} - x_{A1})^2} - v_A \cdot t_2\right)}{t_2^2}. \tag{4.18}$$

In theory, this acceleration value will then ensure that robot A has reached the end of the intersection at the end of the time interval. However, because of the fact that a possible acceleration that places the robot at the correct position at the correct time is one that moves the robot forward past the point and then backwards, it needs to be guaranteed that the velocity does not become negative. The final velocity v_{fA} is for that purpose calculated by

$$v_{fA} = v_A + a_A \cdot t_2. \tag{4.19}$$

If v_{fA} is negative, then x_{A2} will be increased by 0.05 and a_A recalculated with the new distance. If v_{fA} is still negative, x_{A2} will incrementally increase. If one such

solution is found, but v_{fA} is greater than the robots maximum velocity 1.0, then the robot will brake. If there is no distance that fulfills the requirement, robot A will also brake in order to avoid collision.

One can impose restrictions on how much displacement is allowed, depending on what is considered an appropriate definition for the intersection. Such a value can be found through experimental procedures and tests.

The same calculations described above are replicated for robot B with appropriate variables changed to their respective names in accordance with table 4.2.

4.5.2 Implementation of priority solution

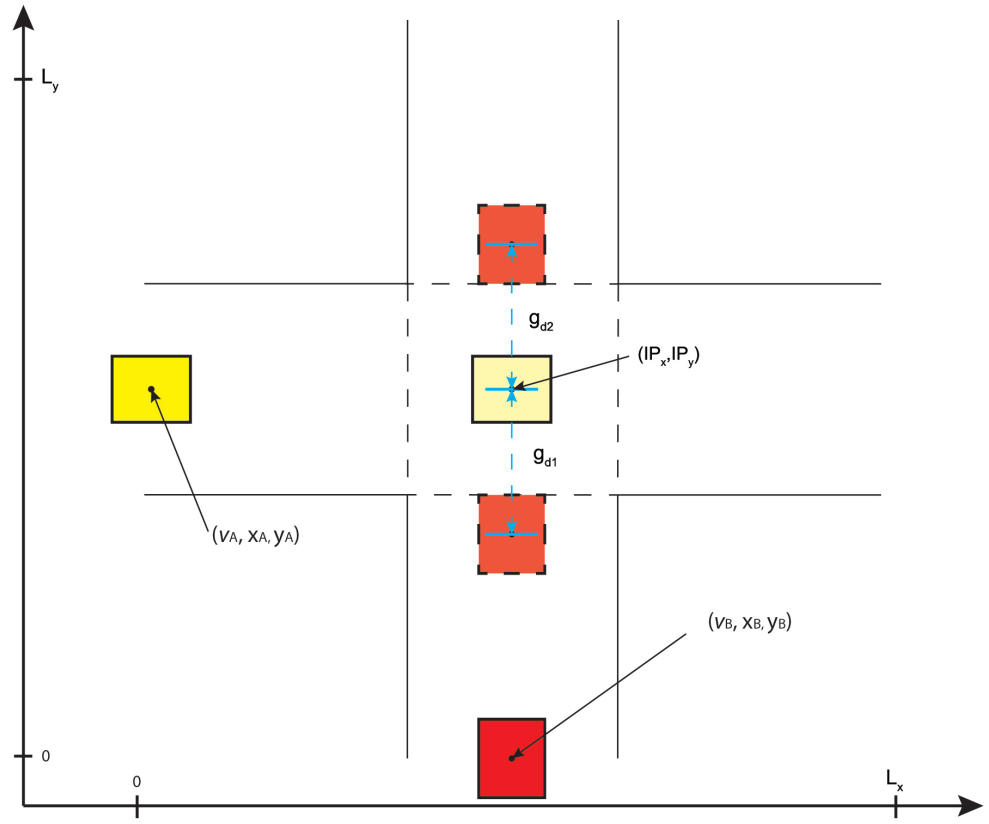


Figure 4.10: The figure depicts the two intersecting robots at different positions and times in the intersection. The yellow robot is the stable robot while the red one is the variable robot. The figure also shows the concept behind the guard distance and how the robots are at different positions when the uncontrollable robot is in the middle of the intersection.

Table 4.3: The table shows the values required from the user in the implementation of the priority solution. These are categorized into a velocity interval, a guard distance and the total size of the system.

User input for priority solution				
Velocity interval	$v_{\text{min}A}$	$v_{\text{max}A}$	$v_{\text{min}B}$	$v_{\text{max}B}$
Guard distance	g_{d1}	g_{d2}		
Total size	L_x	L_y		

As described in section 2.4.3.2, this problem assumes one robot can be controlled and the other cannot. The initial state of the problem is shown by figure 4.10, where the parameters given by the user is shown in table 4.3.

The implementation first randomizes the initial velocities of the robots based on an interval ranging from $v_{\text{min}A}$ to $v_{\text{max}A}$ for robot A and $v_{\text{min}B}$ to $v_{\text{max}B}$ for robot B. The randomized initial velocity becomes v_A and v_B respectively in figure 4.10. The intersection point of the robots is calculated, in accordance to the description in section 4.2.6. After the robots have reached the given velocities, the time T required for the stable robot to reach the intersection is calculated. This becomes

$$T = \frac{\sqrt{(\text{IP}_x - x_A)^2 + (\text{IP}_y - y_A)^2}}{v_A}. \tag{4.20}$$

When the intersection time has been determined, the acceleration required to reach the distance $\text{IP}_y - g_{d1}$ at time T is calculated and given to the variable robot. This is done by the standard formula correlating acceleration, velocity, distance and time, and results in

$$a = \frac{2\Big(\sqrt{x_B^2 + (\text{IP}_y - g_{d1} - y_B)^2} - v_B \cdot T\Big)}{T^2}, \tag{4.21}$$

using the notations from figure 4.10. g_{d1} used here is called the guard distance and must be sufficiently large for the robots to not inhabit the intersection at the same time. This does not preclude the further increase of the guard distance over the minimum necessary to prevent collision and this concept will be used later to adjust the robots control actions, ensuring a successful crossing. The usage of g_{d2} is explained in the next paragraph, but the same principles applies to it as well.

This implementation should avoid collision between the two robots since they will be at different positions at different times. Nevertheless there are limitations that must be imposed on the algorithm in order to accurately represent the real world scenario and the demands placed on it. For instance, the calculated acceleration that is given to the variable robot could be an acceleration that achieves the requirement of being at distance $\text{IP}_y - g_{d1}$ at time T by driving past the point with a deceleration and thereby later incurring a negative velocity in order to reach the point at the desired time. Another limitation is the fact that the robots cannot drive faster than 1.0 meters per second due to comfort constraints. Additionally, they cannot handle

acceleration values smaller than $0.006\,\mathrm{m/s^2}$ due to physical limitations of the robot. In order to deal with the aforementioned limitations, an adjusted algorithm has been developed. It takes different steps based on two different scenarios; if the final velocity is negative and the acceleration is smaller than $0.006\,\mathrm{m/s^2}$ or if the final velocity is greater than 1.0. The final velocity is calculated through

$$v_f = v_B + a \cdot T. \tag{4.22}$$

Since T and v_B are constant when the problem initiates, it becomes obvious that the acceleration needs to be altered in order to change the value of the final velocity. The acceleration itself is dependent on the same constant variables, except it also depends on the guard distance, which can be seen in equation (4.21). By altering the value of the g_{d1}, either with a positive or negative factor, the acceleration will have decreased or increased.

If it is the case that the final velocity is greater than $1\,\mathrm{m/s^2}$, it needs to be lowered. That in turn means that the acceleration needs to be decreased, which is achieved by increasing g_{d1} with $0.05\,\mathrm{m}$. After g_{d1} has been altered, the acceleration is recalculated. If the new final velocity value is still too large, g_{d1} will once again be increased by 0.05. This process will repeat itself until the final velocity is lesser than 1. There is a potential issue where increasing the guard distance too much, placing the point behind the position of the robot, results in negative velocity. If this happens, there is no proper value for the acceleration and the robot brakes to zero velocity almost immediately in order to avoid collision.

If it is the opposite case where the final velocity is negative, the acceleration needs to be increased and in order to achieve this, the relevant guard distance needs to be increased. However since the guard distance g_{d1} is at what is deemed a safe distance from the intersection, increasing it will bypass that safety and place the variable robot too close to the intersection. To combat this the algorithm will use the flipped guard distance g_{d2} instead of g_{d1} when the final velocity is negative, shown in figure 4.10. The guard distance g_{d2} will increase with $0.05\,\mathrm{m}$ and the acceleration will be recalculated, and if the final velocity still is negative, it will continue to increase with the same amount. Since g_{d2} is on the other side of the intersection, its value can be increased without limitation since the robot will stop once it reaches the total length L_y defined by the user at the start of the problem. If the acceleration is smaller than $0.006\,\mathrm{m/s^2}$, the same process will be performed, since its value needs to be increased.

The mentioned solutions to the dangers of the problem introduce another problem in themselves. The problem is that an acceleration that for example initially satisfies the first scenario but not the second, might not satisfy that scenario when it is recalculated to fit the second. This only in principle though, since both methods apply calculations that work against each other. However since there is a possibility of a solution that satisfies both conditions, the calculations are placed in a loop where the algorithm tries to find a solution in 50 calculations where both scenarios are satisfied. This value is arbitrarily chosen, but should be sufficient to guarantee

a solution if there exists one. If it is unsuccessful, the variable robot will brake to ensure collision avoidance.

4.5.3 Implementation of hybrid solution

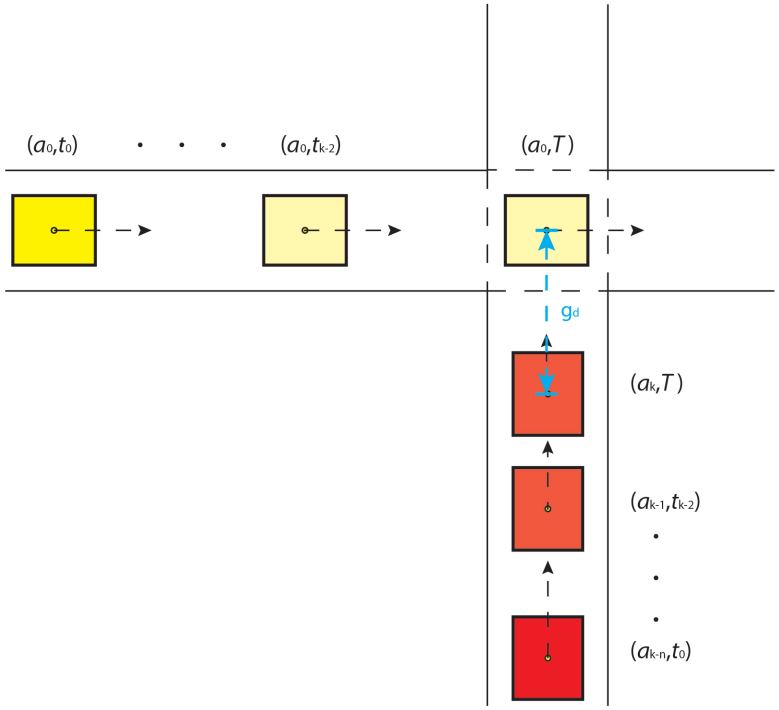


Figure 4.11: The figure shows the concept behind the hybrid algorithm between the priority and the optimization solutions. The arrows show the velocities of the robots, and as in the priority solution the yellow one is the stable robot while the red one is the variable robot. Important to note is that a_k is the smallest acceleration solution that the variable robot can find.

It is possible to optimize the algorithms for the priority solution as presented in the previous section in regards to the control actions chosen. The resulting algorithm is a hybrid between an optimization and a rule based solution, and is the one that the real world experiment was conducted with.

The idea that is implemented here is that the problem reevaluates continuously as time passes, with updated parameters from the various inputs to the system such as location data, creating a closed-loop control. There is a possibility that the acceleration found at the previous time step was not the smallest absolute solution for the system. This because of the fact that the algorithm does not incrementally increase in infinitesimal steps as well as the fact that it performs a limited number of calculations before being stopped. On top of this there is also the fact that the positioning is affected by noise. This means that the trajectory of the robot can change slightly, and by extension also changing the intersection point and thus

the distance to it, resulting in a new acceleration. The absolute value of the new calculated acceleration is compared to the previous one, and the smaller one of the two is given to the system. This is done for each time step, resulting in an optimization in regards to minimizing the acceleration.

The figure 4.11 demonstrates the concept behind the hybrid algorithm. Here the stable robot maintains a constant acceleration $a_0 = 0$ and arrives at the middle of the intersection at T. Simultaneously the variable robot calculates an acceleration. Initially at t_0 it finds the acceleration a_{k-n} that satisfies the demands in accordance to the priority solution. However later at time t_{k-2} it finds a smaller absolute acceleration a_{k-1} and therefore changes its acceleration value. At time T it will have avoided collision using the smallest absolute acceleration solution it could find. Note that it does not need to stop at g_d since the guard distance can be altered in order to find new acceleration solutions, as described in the previous section. The figure also shows one case, there is also the other case where it drives past the intersection before the stable robot. The important part to note is simply that.

$$a_{k-n} > \ldots > a_{k-1} > a_k. \tag{4.23}$$

Besides the fact that the acceleration becomes optimized, this approach also offers improvements in the sense that if a solution is not found, and the robot does not need to emergency brake, it can use the previous acceleration value and try again at a later time with different initial values. One needs to be careful in this situation since the previous acceleration value can be one that is results in a negative final velocity. The algorithm therefore disregards old acceleration values if the robot has passed the guard distance and the old acceleration value was aiming behind the intersection.

In general this results in a greater possibility of finding solutions that satisfy all conditions and results in a successful intersection. It can however also present a danger if it takes too many tries to find a solution and the robot comes dangerously close to the intersection point. There is also a risk that there is too much noise and an erroneous acceleration is calculated that results in a collision. The emergency brake therefore becomes crucial in this implementation to decrease the probabilities of collisions. It is also of utmost importance to choose an appropriate guard distance to further reduce the risks if there is substantial noise in the positioning.

4.6 The setup of the experiment

The setup of the experiment is critical as this is how the intersection controller is tested in the way that fulfills the goals of the project. By testing the controller in this way, the system is exposed to the true noise of both the robots and how they execute control actions but also the noise/error of the location system which differs from the simulation software.

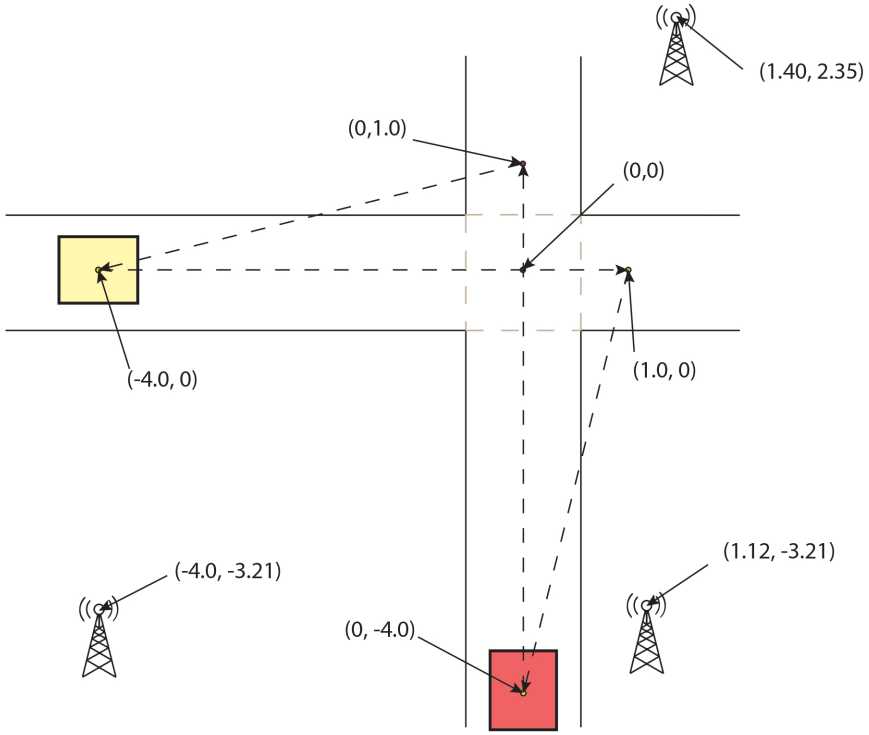


Figure 4.12: This figure represents the setup of the experiment to scale. The anchors are represented by antenna symbols and are placed in three locations around the intersection. Each robot has its path also mapped out using the dashed lines in the figure showing the complete. The figure does not depict the controller or the Wi-Fi network as their positions are not important to the experiment.

In figure 4.12 the full setup and paths that the robots are programmed to follow can be seen. The three antenna symbols match where the anchors are placed for the location system with each corresponding location. The paths of each robot are depicted by the dotted lines that go through the intersection and then back to a start point. This setup allows the robots to run the emulation over and over again, measuring the real world robustness of the controller in preventing collision. In the setup, the calculations are done on a centralized computer which communicates its instructions to the individual robots through the Wi-Fi network. At each start point the robots are given a random velocity between 0 and 0.3 m/s to replicate an intersection and the randomness of the vehicles entering the intersection. The top robot is programmed to hold its velocity as per the hybrid priority algorithm described in 4.5.3. The bottom robot is then controlled to avoid collision with the top robot as it travels through the intersection. Following the intersection crossing, the robots then travel to start positions, switching places and continuing with a new intersection crossing.

5

Results

The results of the different scenarios previously discussed are put forward. These scenarios aim to determine the quality of the algorithms, hardware, software and general functionality of the entire system. As aforementioned, the controller algorithm being tested is the hybrid priority optimization algorithm presented and described in section 4.5.3.

The following figures show the results of 10000 position measurements done by the UWB system around a single static point, which was placed as near to the center of the intersection of the following tests as possible. This test was meant to give a reasonable measurement of the accuracy of the positioning system around the area of greatest importance to the experiment, the intersection.

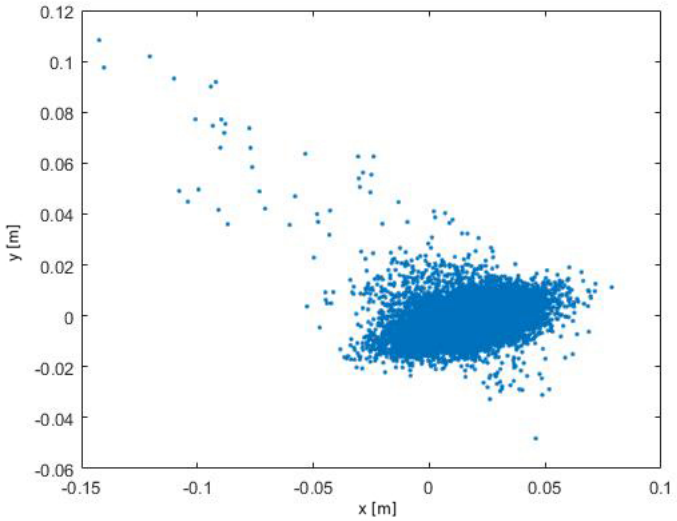


Figure 5.1: The figure shows the resulting x- and y-coordinates following 10000 location measurements from the UWB sensors.

Additionally, the covariance matrix that was derived from this test is the same used to represent the level of uncertainty for the observations used by the EKF. The covariance of the observed data was determined from the reading of UWB sensor at

the stationary point. The resulting covariance matrix is:

$$R = \begin{bmatrix} 0.00031 & -0.0000149 & 0 \\ -0.0000149 & 0.0000689 & 0 \\ 0 & 0 & 0 \end{bmatrix}. \tag{5.1}$$

The standard error that results from this ideal sample is 1.76 cm of the x-coordinate and 0.83 cm for the y-coordinate. The measurement data is also visualized in figure 5.2 to 5.3 where it is possible to see that the data fits well within a normal distribution curve.

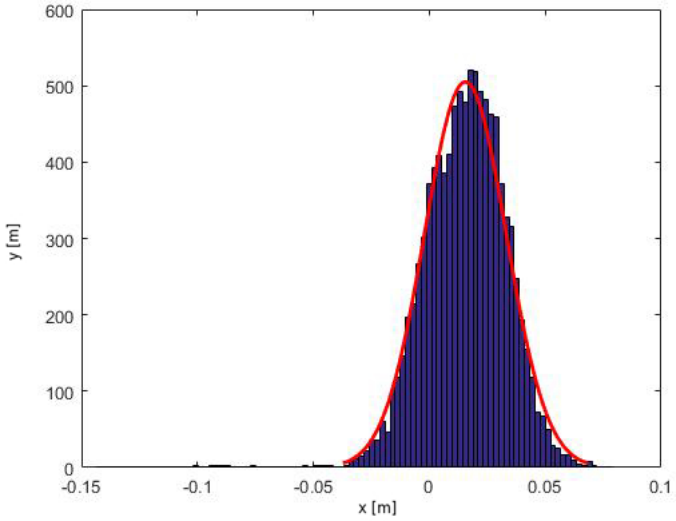


Figure 5.2: The distribution of these 10000 location measurements in the x-axis and a best fit normal distribution curve overlayed.

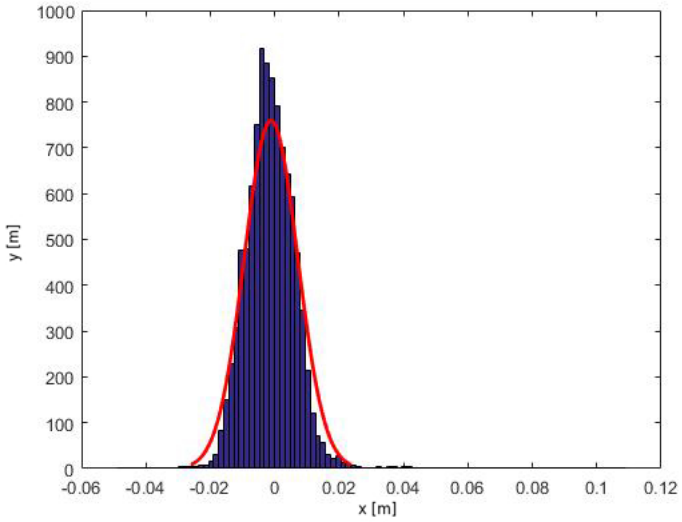


Figure 5.3: The distribution of these 10000 location measurements in the y-axis and a best fit normal distribution curve overlayed.

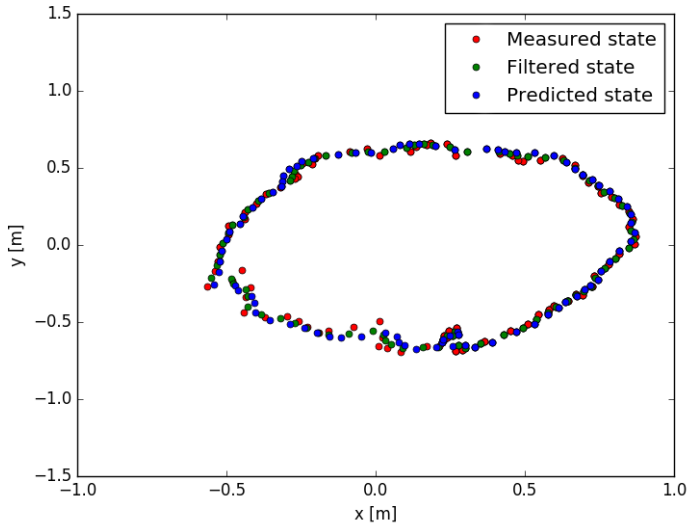


Figure 5.4: Measurements of the position from the predicted state, sensors and filter as the robot moves with constant angular and linear velocity. Note that the axis are shifted which deforms the otherwise circular motion.

To test the filter, one robot was first given a constant angular and linear velocity. This made the robot move in a circle. The position from the predicted states $\boldsymbol{x}_{k|k-1}$ (predicted from a previously filtered state $\boldsymbol{x}_{k-1|k-1}$), the position from the filtered state $\boldsymbol{x}_{k|k}$ and the observed position $\boldsymbol{z}_k$ are plotted out in figure 5.4.

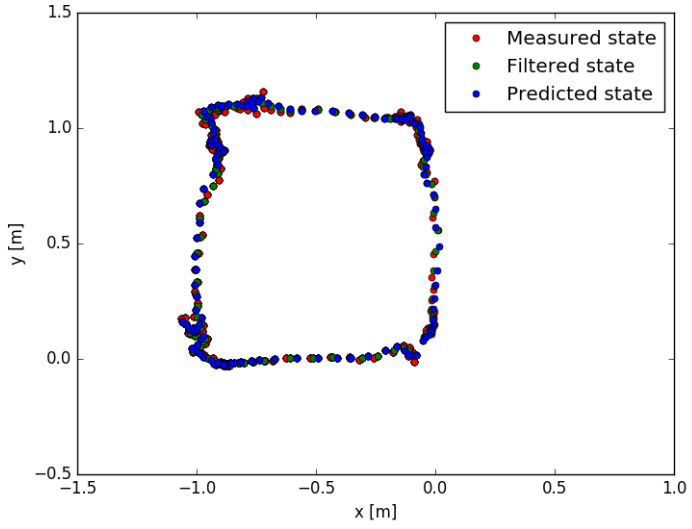


Figure 5.5: Measurements of the predicted state, sensors and filter as the robot moves in a square. When it reached a point, it stood still for a second before moving on, resulting in some point clusters around the points of the square.

A similar measurement was made when the robot moves in a rectangular shape where it is programmed to move to each corner in the rectangle. This result is shown in figure 5.5.

5.1 Results of the experiment

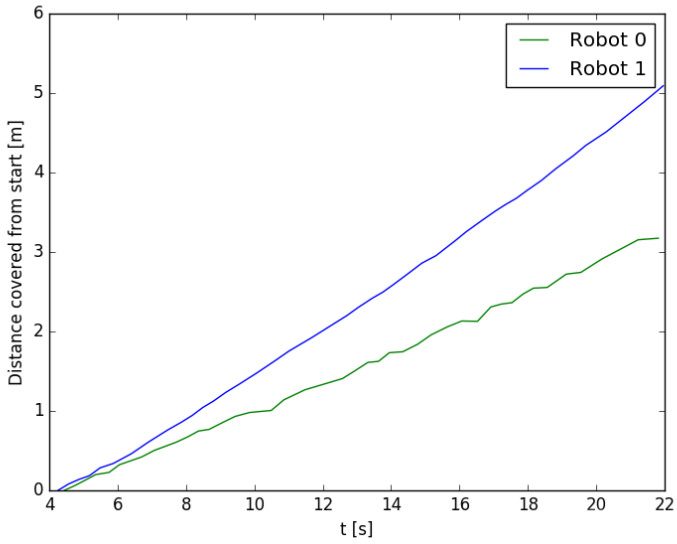


Figure 5.6: The figure shows the distances travelled for each robot, calculated from their respective starting points. In this case Robot 1 accelerates in order to avoid collision whilst Robot 0 maintains constant velocity.

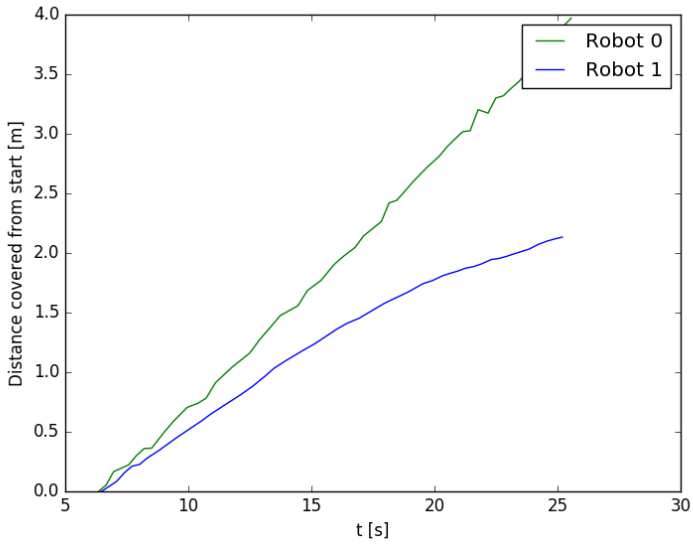


Figure 5.7: The figure shows the distances travelled for each robot, calculated from their respective starting points. In this case Robot 1 decelerates in order to avoid collision whilst Robot 0 maintains constant velocity.

To show that the robots accelerate and decelerate in the real world setup, graphs were produced showing distance they travelled from their starting points. Figures 5.6 and 5.7 show the results of running the hybrid algorithm with 2 meter guard distance. Figure 5.6 shows the variable robot accelerating whilst 5.7 shows it decelerating.

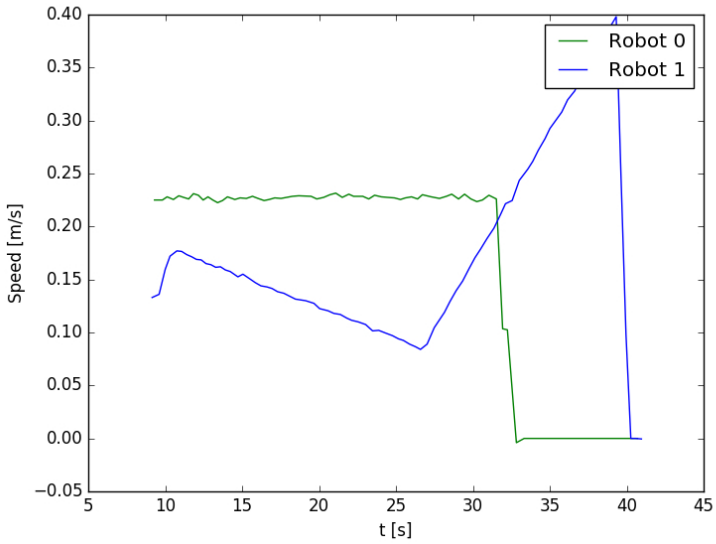


Figure 5.8: The figure plots the velocities of the robots against time. The green data represents the stable robot whilst the blue data is the variable robot. The initial velocities are randomized and both travel to the ends of the intersection. After the possibility of a collision disappears, the optimization of acceleration is disregarded.

Figure 5.8 shows the different velocities of the two robots as they travel through the intersection. It is important to remember that as soon as the risk of collision is zero, the variable robot will disregard the demand of minimizing its acceleration and will instead travel with a high acceleration to the end of the path. For the measurements from the opposite situation where the variable robot reaches the intersection first, and therefore does not need to accelerate with a high constant value, see figure A.3 in appendix A.4.

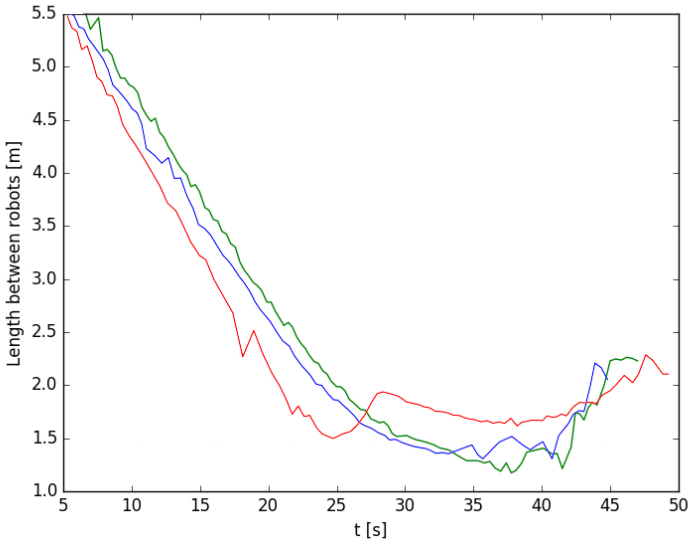
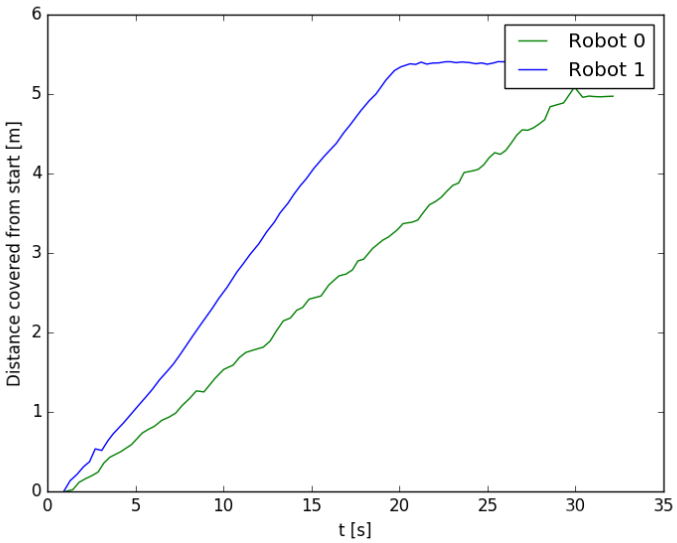


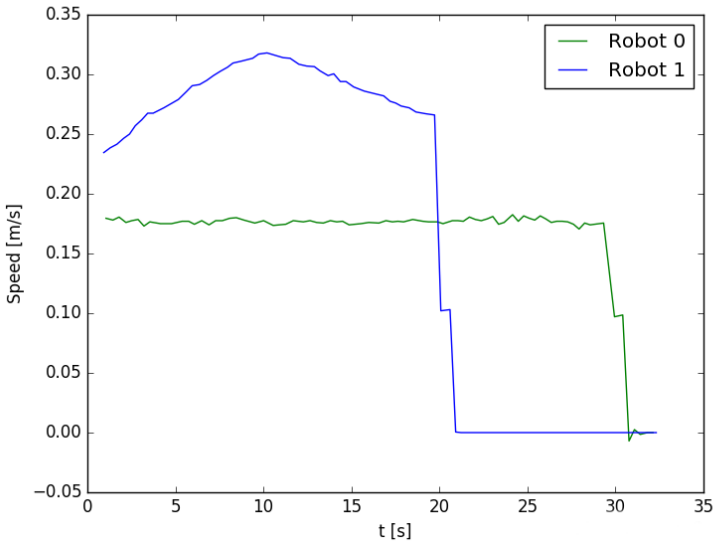
Figure 5.9: The figure shows data gathered from three different measurements. The data itself is the length between the robots plotted against time. The blue and green data were collected when the variable robot decelerated whilst the red data was from when it accelerated.

In order to verify that the robots do not collide with each other, data was gathered that shows the distance between the robots. Several measurements were taken and three of these are presented in figure 5.9. As shown the distance never gets smaller than 1 meter. The blue and green colors represent data gathered from when the variable robot decelerates, whilst the red data is from when it accelerates.

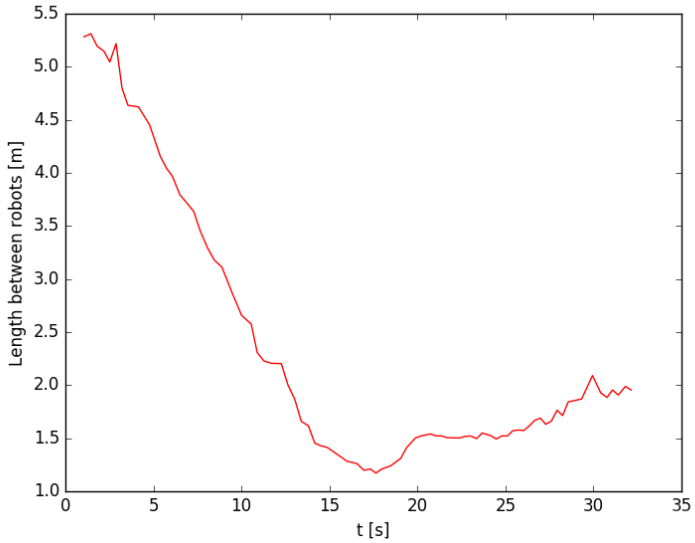
5.1.1 Controller results from a single measurement



(a) The graph shows the total travelled distance of the robots as a function of time.



(b) The graph shows the different speeds of the robots as a function of time.



(c) The graph shows the length between the robots as they traverse the path.

Figure 5.10: The figures 5.10a, 5.10b and 5.10c show different data collected from the same measurement. Figure 5.10a plots the distance covered, 5.10b is velocity of the robots and 5.10c is the distance between them. The blue lines represent Robot 1 which is the one that is controlled, while the green lines represent Robot 0 which is the stable robot and travels with constant velocity. In this case the variable robot mostly accelerated.

The figures and results presented in the previous section are all important and relevant in their own regard, however showing those interpretations from the same measurement becomes an interesting option as well. This is presented in figure 5.10 and 5.10a, 5.10b and 5.10c. This measurement covers the case where the variable robot crosses the intersection first. For measurements regarding the other case where the stable robot crosses first, see figure A.2 in appendix A.4.

5.2 System robustness for different scenarios

Multiple tests were conducted in order to test the robustness of the system in the real world. First and foremost, the initial goal of 50 intersections without collisions was achieved. This with randomized initial velocities from 0.1 to 0.3 m/s and positive and negative guard distance of 2 meters. The test was repeated several times, and although 50 intersections were not observed each time because of time limitations, the observed intersections did not result in collisions.

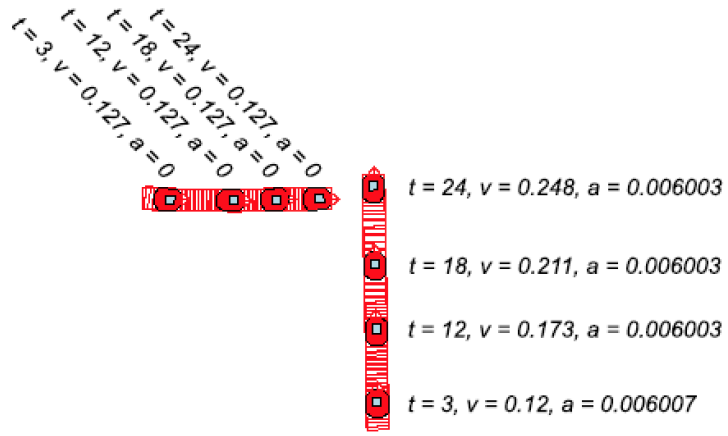
Furthermore, the initial velocity interval was changed to 0.1 to 0.5 m/s with the same guard distance of 2 meters, in order to test the robustness. Although a collision was not recorded over several continuous tests, it was still resolved that a larger guard

distance was required to have more comfortable margins.

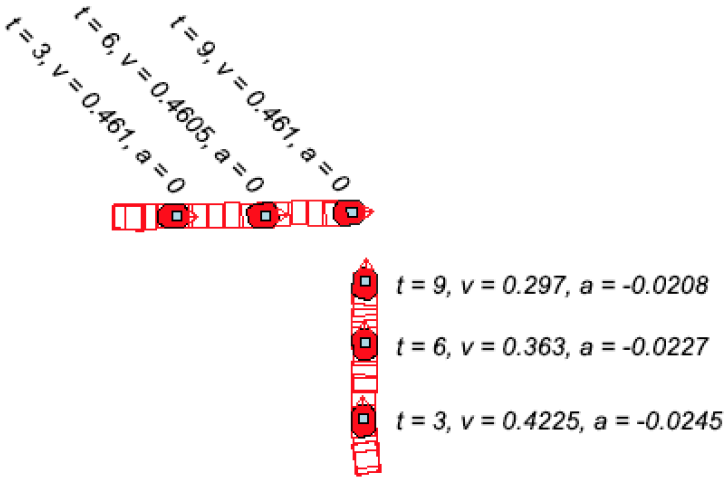
5.3 Simulation results

Using MobileSim, as presented in section 3.2.1.5, more results were able to be gathered covering some areas that were not tested with the real robots. This includes testing the robustness of the algorithms without having to consider real world limitations such as space, equipment and setup time. The optimization of the controller algorithm was tested, as well as different noise levels and initial parameters. In the real experiment, the goal was to achieve at least 50 intersections without collision, but in the simulation it was downsized to 20 in order to gather more results.

5.3.1 Controller in simulation



(a) Optimized acceleration

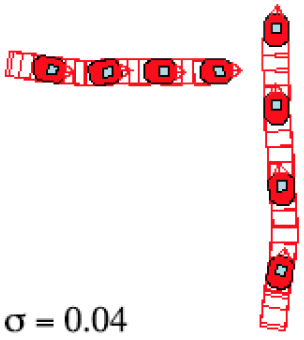


(b) Optimized deceleration

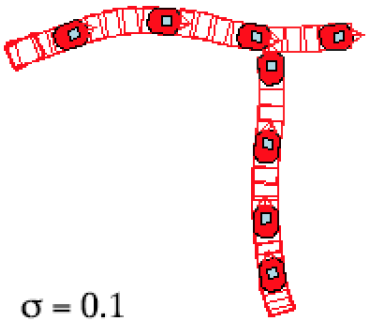
Figure 5.11: Simulation results of hybrid algorithm using 4 meters distance to center, 1 meter negative guard distance and 2 meter positive. Randomized starting velocities in interval 0.1 to 0.5. Figure 5.11a shows the case in the simulation were the variable robot crosses the intersection first while optimizing its acceleration. Figure 5.11b shows the same thing but with deceleration and the stable robot crossing first.

Figure 5.11 shows two primary scenarios; one where the variable robot passes the intersection first, and one where the stable robot passes first. This is shown in figures 5.11a and 5.11b respectively. The robots are shown at several different time steps in the simulation, and together with the specific time step their velocity and acceleration at that time is also shown. Through this information, the optimization of their accelerations becomes clear. The data was collected using the parameters of 4 meters distance to center, 1 meter negative guard distance and 2 meter positive. Randomized starting velocities in interval 0.1 to 0.5 were given to the robots.

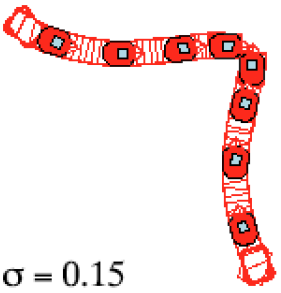
5.3.2 Controller in simulation with different noise levels



(a) Noise 0.04 and no collision



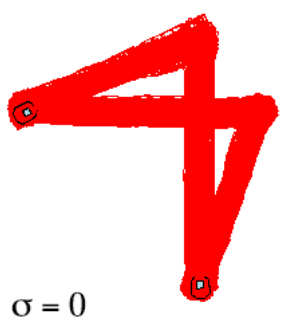
(b) Noise 0.1 and no collision



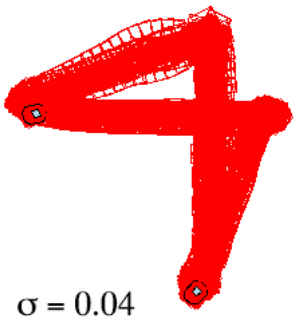
(c) Noise 0.15 and collision

Figure 5.12: Simulation results of hybrid algorithm using 4 meters distance to center, 1 meter negative guard distance and 2 meter positive. Randomized starting velocities in interval 0.1 to 0.5 and different noise levels.

Simulating different noise levels becomes an important feature of the simulator as replicating a similar situation in the real world is difficult. Figure 5.12 was generated with the motivation of testing the robustness of the system in a different way. It shows how the different values for the standard deviation of the noise affect the intersection crossing drastically. Figure 5.12a shows a typical path of the robot when the standard deviation is at 0.04, while 5.12b shows the same thing with standard deviation at 0.1. Figure 5.12c with 0.15 standard deviation follows the pattern but unlike the others, shows a way for a collision to manifest itself.



(a) Noise 0 and 10 intersections



(b) Noise 0.04 and 10 intersections

(c) Noise 0.1 and 10 intersections

(d) Noise 0.15 and 5 intersections

Figure 5.13: The figure shows the hybrid algorithm traversing through the intersection 20 times with different noise levels. The guard distance was 1.0 in the negative direction and 1.5 in the positive, whilst the velocity interval was from 0.1 to 0.5. In the case with standard deviation of noise 0.15, shown in figure 5.13d, only 5 intersections were recorded since collision could not be avoided with the selected guard distance.

Figure 5.13 shows what was recorded after 20 intersections of the controller for different noise levels and using the hybrid algorithm. In the case with 0.15 standard deviation, shown in 5.13d, only 5 intersections were recorded since collision avoidance could not be guaranteed for more crossings.

5.3.3 Controller in simulation with different velocity intervals

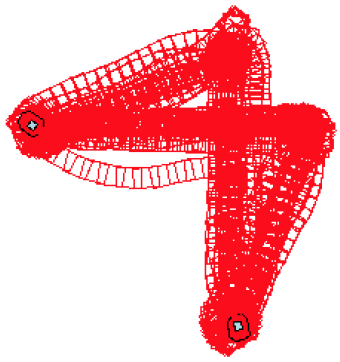


Figure 5.14: The figure depicts the path travelled by the two robots when the starting velocity interval goes from 0.1 to 0.8 m/s. The pathing is a result of 10 intersections.

Another aspect of interest is testing different velocity intervals. This because of the fact that travelling faster can result in a small angular error not correcting itself quickly enough, resulting in a higher probability of collision. On top of that it can also lead to overshooting which can lead to aggregated errors in the calculated angle. Figure 5.14 shows the results of a simulation with an initial velocity interval ranging from 0.1 to 0.8 m/s. The plotted path is a result of 10 intersections.

5.3.4 Controller in simulation with decreased starting distances

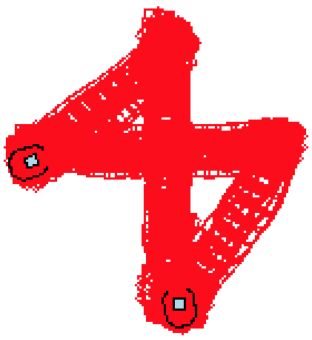


Figure 5.15: The figure shows the pathing of the two robots as their distances to the intersection point are set to 2 meters. The starting velocity interval is from 0.1 to 0.5 and the figure shows the results from 10 intersections.

It is also worthwhile to try to test the controller with decreased distance to the intersection point. Increasing the distance would only improve the controller since it would have more time to correct itself from the filters point of view, and it would also be more likely to find better acceleration solutions. On the other hand decreasing the distance could prove to be troublesome since it could not have enough time to find a solution, or if its initial angle is inaccurate, it could take too long to correct it. The room for error also decreases given that the robots are always close to each other. The results of a simulation with decreased distance of 2 meters to the intersection point, is shown in figure 5.15. The velocity interval was from 0.1 to 0.5 and it shows the result of 10 intersections.

6

Discussion

This chapter contains an analysis of the gathered results. Different reasons and causes for the successes and failures of the system are discussed and potential error sources are presented.

6.1 General analysis of the system

The system and the implementations used proved to be as efficient, flexible and robust as desired. Several limitations arose as consequences of the equipment and software, regardless these were solved by specific algorithms.

Several interesting conclusions can be reached from reviewing the gathered results. Looking for the guard distance in the results is a way of evaluating the quality of the algorithm as its magnitude should never be less than the chosen value of 2 meters. Figures 5.6 and 5.7 show that, in the case with deceleration, the guard distance is found exactly at 25 seconds. The reason why it is not visible in the same way when it accelerates is that the robot stops when it reaches it destination on the other side of the intersection, which in this case was placed before the guard distance. It is also worth remembering that because of the possibility of iterative decrease or increase of guard distance, that it is not certain that the robot will always land on the initial guard distance, in spite of where the final destination is. Regardless, the fact that the differences between the robots are not smaller than the magnitude of the guard distance shows that the acceleration values chosen by the algorithm are desirable.

To clearly show that the robots do not collide with each other as they travel through the intersection, graphs showing the length between them were produced and presented in the results. Looking at figure 5.9, that shows this from three measurements, it is clear that running multiple measurements still presents reliable and similar results, also regardless if the variable robot accelerates or decelerates. The lines showing data from the case of the stable robot passing the intersection first are relatively straightforward. The distance between the robots decrease and then plain out when the stable robot reaches the intersection point, until they both reach their end points and lead to the increase in distance between them. Looking at the red line representing the case where the variable robot accelerates, one notices a soft

spike at 28 seconds before the distance is again decreased followed by an increase, like in the other measurements. The valley most likely represents the case that the controllable robot passes the intersection point, leading to the robots being as close as possible before distancing themselves again until the controllable robot reaches its final point. This would be the spike at 28 seconds, and afterwards the uncontrollable robot will have moved through the intersection, again decreasing the distance, until it passes it and reaches its final point leading to the increase of length between them.

Also notable is the fact that the smallest distance between the robots is less than the guard distance of 2 meters. This is however due to the fact that the guard distance does not result in the smallest distance between the robots. Depending on the difference in speed between the robots after one has crossed the intersection point, the distance between them might get smaller.

Having analyzed the graph mentioned in the previous paragraphs, it becomes clear that having other information of the intersection could be helpful in figuring out what transpired at certain times. This was the reason figure 5.10 was created and shown. Having the same times present in all of the graphs makes it tremendously easier to link them together. The measurement covers the case where the variable robot passes the intersection first, and immediately it is possible to verify the explanation given for the spike in the red line from figure 5.9. Here it is clear that at 20 seconds, the controllable robot has reached its end point since 5.10a shows that it does not cover more distance after that time, and 5.10b shows that it has 0 velocity after that time as well. Now looking at figure 5.10c it can be seen that the graph shows a soft spike at 20 seconds, precisely the same as with the red line in 5.9.

6.1.1 The performance of the UWB sensor grid

As is shown in the results and particularly figure 5.1, the precision of the sensor grid is lacking especially when considering the distances that are being measured. Having such an error for the x- and y-coordinates over a distance of only few meters can be seen as disappointing when considering the needs of the project. There are two factors that make this system poor.

The first factor is the issue of error in the x- and y-coordinates. These in the ideal scenario measured were around $\pm 1.5\,\text{cm}$ as seen in figure 5.1. It is possible that this can become worse as presented in the diagrams 5.2 and 5.3. This error can be an issue when compared to the size of each robot, which is 38 cm, resulting in this error being 8%. When compared to the guard distance however this is well covered by the 2 meter boundary.

A second factor that effects the robots is the inability of the location system to determine an accurate orientation of the robots. This is referred to and discussed in section 6.1.2.

It is interesting to note the difference between the uncertainty on the x-axis and y-axis. The spread of the points in the negative x- and positive y-direction happens to be the same direction where an anchor was not placed. This is likely explained by the positioning of the anchor antennas in a triangle resulting in a greater precision in certain directions due to the unequal orientation of the anchors and the robot's antenna.

Despite these limitations, the UWB system still has advantages. When looking outside of this emulation, into larger and more practical aspects of implementing a system such as intersection control, the most common areas of usage are traffic flow on public roads and other forms of transport and logistics systems. Due to the larger scale that these could be implemented with, the positioning systems also needs to scale with said systems. These ranging from similar radio based positioning systems all the way to the GPS system. Additionally, these systems need to have a robust way of handling positioning data in anticipation of unexpected changes such as environmental obstacles. In this way, the tools that are used in this project are highly applicable to a real world scenario.

Many of these systems build upon the same concepts and functions, e.g. trilateration, using similar algorithms, such as gradient descent, to solve these location problems. To look at GPS more closely, the standard measurement error can be upwards of 3 meters [23]. As a matter of scale this makes it comparable to the system implemented in this emulation. Since these comparisons can be made, solutions found in this report could be more generally applicable as well.

6.1.2 Positioning and angle calculation during filtering

The results gathered in section 5 compare the sensor data with the predicted state and the filtered state that is used to more accurately obtain the location of the robots. In figure 5.4 and figure 5.5 it is clearly seen that the all of the states (x, y) plotted follow the actual path that is travelled by the robot. The points representing the filtered states also appear to be a mean of the predicted and the observed position, both which often differ from each other.

The purpose of the observation is to restrict the noise from the predictions to aggregate to unrealistic values. While the predicted state seems to deviate from the actual path at times, the observed state corrects this error. Likewise, the prediction will prevent the observed state from recording an incorrect action such as a sudden sideways movement. This is seen in figure 6.1, where the observed measurement shows a sideways movement, most likely due to a noisy measurement.

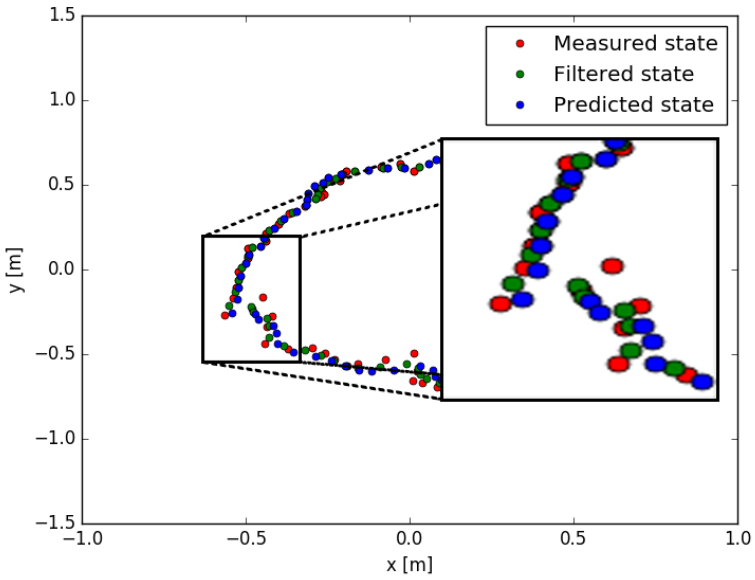


Figure 6.1: The figure highlights an area of where the predicted state prevents the observed state from recording an incorrect action.

The angle of the robot, which is the third state that needs to be calculated, does not have any correction in the filter, but instead is an estimation based on the previous angle, a time step and the robot's angular velocity. Therefore this state allows for errors to aggregate, where eventually the angle deviates too much for the system to work properly, ex. find it's target point. To correct this, a way to observe the robots actual angle would be ideal, such as an electronic compass. This could then be sent to the filter and prevent the aggregated errors from occurring. An alternative method for correcting the angle is by using the positioning data. If introducing a calibration mission, the actual angle of the robot can be derived by comparing the current position with the previous position using trigonometry. This has the effect of zeroing the system's accumulated orientation error as shown in the figure 6.2.

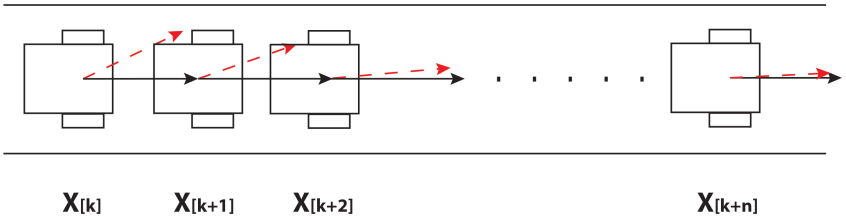


Figure 6.2: This is an example of a calibration mission that the robot can run after the intersection in order to fix faulty orientation data in the robot. The red dotted vector shows the angle of orientation that the robot believes it has when starting on its trajectory as shown by the black vector. Over time this vector is show to be fixed as the robot accumulates more accurate position data to compare its orientation with.

The solution implemented and described in section 4.2.4 effectively minimized the errors that came from using only the predicted angle of the robot in the system, however this requires the robot to be started in a known angle at the start of any run. Also, the more the robot changes between the fixed angular velocities, the more errors will aggregate in the system.

6.1.3 Evaluation of hybrid algorithm

The hybrid algorithm used to gather the results and that is the primary algorithm used throughout the project can be evaluated in many ways. The first and most basic evaluation is whether or not it results in collision avoidance, which was answered in section 6.1. Another more subtle aspect is whether or not there is a visible optimization, and if that optimization is desired and leads to an improved system. Looking at figure 5.8, which shows the speed of the robot, one can see the optimization algorithm minimizing the absolute acceleration value. At the start of the measurement the variable robot has a relatively high acceleration which leads to a rapid increase of velocity. However a couple of seconds later it finds a smaller acceleration value which it then applies until the risk of collision is gone, which leads to an end to the optimization and it simply assumes a constant high acceleration. Comparing the slope of the graph from 10 to 12 seconds to the slope from 12 to 27 seconds, makes it clear that the new acceleration value is smaller than the initial one. A similar occurrence can be noted in figure 5.10b. Here the variable robot switches acceleration values at about 10 seconds and once again a smaller absolute acceleration value was found, leading to the change in velocity.

A tangible advantage of using the hybrid algorithm as opposed to solely the priority protocol, is apparent when considering the advantage of reevaluating the acceleration continuously. Before heading to the intersections, the robots aim themselves to the points after the intersection. Due to noise and miscalculations, they can believe that they are aimed correctly and start moving, resulting in paths shown with the higher noise levels in figure 5.13. The algorithm will attempt to correct their movement, but if the acceleration is not recalculated, the distance travelled will not match what the robot calculated at the start of the problem, effectively meaning that the calculated acceleration value that would avoid collision does not necessarily do so. With the hybrid algorithm, the acceleration is recalculated as the robot travels to its end point, meaning that it considers new distances all the time. There can still be errors with this method, but the risk is reduced.

After having optimized the algorithm for the priority based protocol, one possibility worth considering is whether or not the same thinking could be applied to the reservation based system. Initially this might seem feasible and even preferable, but considerable difficulties arise when taking into account that acceleration commands are only given and calculated once the robot is in fact inside of the intersection. This means that it cannot choose to stop earlier in order to decrease the final velocity. If the final velocity is greater than 1.0 then you cannot decrease it and if you use the emergency brake, you will stop in the middle of the intersection, causing a collision.

Doing an initial calculation at the start of the intersection is possible since if you stop immediately there then a collision is impossible, however in order to optimize the control action you need to continuously evaluate the acceleration, which leads to the previously mentioned problem. This was a considerable factor when choosing the hybrid of the optimization and priority solutions as opposed to the reservation solution, for the algorithm used in this project.

6.2 Simulation analysis

The simulation results presented in section 5.3 can give more information regarding the algorithms that the gathered results from the real-world experiments cannot. Glancing at 5.11a, the optimization becomes even more evident. Remembering that a predefined lower limit of the acceleration is $0.006\,\mathrm{m/s^2}$, it becomes rather impressive of the system to first find a solution 0.006007 that close to the minimal value, but then to minimize that to 0.006003 truly shows the optimization working as intended. The optimization is also seen in 5.11b where it continuously decreases at all three time steps.

The simulations with the different noise levels is where the simulation completely separates itself from the real world experiment in regards to what is easily possible to test or not. Looking at the results gathered and shown with figure 5.12 and figure 5.13, the limits of the controller become apparent. For instance, trying to prevent an intersection with noise standard deviation at 0.15 becomes near impossible when the trajectory is as uneven and unpredictable as shown in figure 5.13d. Noise levels ranging from 0.04 to 0.1 seem however to be acceptable, and seeing as the UWB produces errors at about 2 centimeters, it should not present any impossible scenarios for the controller. This is also what was seen in the real world experiments since no collision was recorded and no path like that from figure 5.13d. If for instance such a path was present, it would definitely show in the measurement from 5.9 as they would not be similar at all. Notice however that the results gathered here are from 20 intersections, excluding the one from 5.13d, and that the pathing would deteriorate with more runs. Nevertheless, the general pattern shown in the simulation results should persist and definitely represent a generous estimate of what would be expected at 50 intersections as well.

Besides noise disrupting the system, there is another source of error that leads to erroneous pathing. That is overshooting and is shown quite accurately in figure 5.14, where an increased starting velocity interval leads to higher velocities and acceleration values on top of those lead to overshooting. Overshooting in turn leads to miscalculated angles which leads to more overshooting and in general a less robust system. A comparison between the two scenarios where the system becomes less robust, that is between this figure and figure 5.13d, shows several similarities. Foremost, neither of them avoid collision for 20 intersections, even if the one with an interval containing higher velocities is still more robust than the noisy model. This can be seen from looking at the figures and comparing their paths or from a more theoretical point of view; in the case with the interval containing higher velocities it

only overshoots if certain conditions are met whilst the noisy model is always noisy.

A very important factor that cannot be neglected when discussing these scenarios, is the guard distance. The guard distance's primary use is to adjust the system for how much deviation the robot has from its ideal path when it travels through the intersection. These simulations were all ran with the same primary settings, meaning that they had a guard distance adjusted for relatively low noise and a lower velocity interval. Changing the constants used for the angular velocity when the robots are aiming too far away from the point they are going to, could also reduce the risk of collision at high velocities. Those constants being too low could result in the robot not having enough time to correct its direction while its traveling to and through the intersection. Adjusting the guard distance for the problem parameters would definitely increase the robustness of the system. However regarding the case with the high noise, the model is very unpredictable which lessens the effect of the guard distance.

A similar mindset should be taken when decreasing the distance from the starting points of the robots to the intersection. The results from doing this is shown in figure 5.15, where the distance was reduced from the standard 4 meters to 2 meters. This lead to an increased probability of collision, mainly due to the same problem as with the high velocities, that the robot does not have enough time to adjust its direction when its travelling to its end point. An adjusted guard distance would also lead to much greater robustness in this case.

6.3 System robustness analysis

From both the simulations and the real world experiments, it was clear that the parameters we give the algorithm are co-dependant. For instance if the system is to have an interval with higher velocities, it is necessary that the guard distance is increased. In general the system will become more robust the greater the guard distance chosen, since the possibility of collision will decrease due to greater distances between the robots. However if this was the sole purpose of the controller, it is easiest to instantaneously give the variable robot zero acceleration and always avoid collision. This obviously is not a desired algorithm and we can add flexibility to the demands of our algorithm. The smaller the guard distance, the more efficient the system and algorithm will appear. Therefore one wants to choose the smallest value for the guard distance that does not result in a collision. It is also in our interests to have the largest possible starting velocity interval since that shows the flexibility of the system in the way that it can handle a greater array of scenarios. In this equation one also needs to consider the fact that the higher the velocities of the robot at the start, the greater the possibility of a collision. Hence it is up to the user to decide its desirable to have a large starting velocity interval and then also large guard distances and less fluidity, or if the opposite of having smaller guard distances and a more fluid system and sacrifice flexibility.

6.4 Further development on the intersection controller

The algorithms and methods deployed can be improved upon in the future given more resources and time. Several of these possible improvements are presented below.

6.4.1 Improved robustness

The system, although meeting the desired robustness, could still be improved upon. The algorithm theoretically guarantees that there will be no collisions in the intersection at the set velocities used in the tests. The robustness, however, was reduced over time due to the poor calculation and correction of the states, specifically the robot's angle. If further tests and optimization were to be made on system, this problem needs to be able to be corrected. This will allow tests to see if the theoretical zero-collision robustness actually is achievable, meaning more consecutive intersections than previously obtained is possible. A larger variety of algorithms could also be tested on the system to compare robustness or various optimization methods. Gathering data from the implementation of the reservation and priority solutions and comparing that to the hybrid algorithms could also further support the superiority of the hybrid algorithm.

To create a more real-world applicable scenario, the system can be equipped with more sensors that are of a higher accuracy. This includes cameras, radars, lasers and other equipment that are currently being used in the development of autonomous vehicles. With this extra data, the system would be able to determine its state with greater certainty. This would allow for opportunities such as emergency brakes, obstacle avoidance, lane detection and other functions that will help navigate the vehicles through an intersection, as well as further prevent any accidents from occurring. Additionally, this increased robustness could allow for greater liberties being taken in regards to optimization and thus efficiency.

6.4.2 Further solution possibilities

In a real scenario, an intersection will have to handle more than two vehicles trying to cross at the same time. An intersection will most likely also contain additional lanes and more intersecting roads. It would therefore be reasonable to improve the system to be able to handle this. The current algorithm used would not be directly applicable for such scenarios, since it assumes that there are two vehicles going into the two way intersection and only one robot needs to adjust its speed to the robot with priority.

Previous studies have been made on more appropriate approaches for handling these various scenarios. One approach is creating a grid of sectors in the intersection that will be reserved by vehicles according to their lane or route, if they need to turn for

example [3].

The final solution used in this report is as described a hybrid of a optimized and rule based solution. Pure optimization solutions are however possible [1]. These solutions would not only optimize the control actions of one vehicle but for the whole system, finding the most efficient solution possible. The complexity of these solutions compared to the ones in this report are significantly increased however.

7

Conclusion

In conclusion, the purpose of the project was to look at ways that fully automated vehicles can deal with the problems of road intersections. The demands were a robust system that prevents collision and also optimizes the movement of the vehicle through the intersection. The limitation of the problem is that it only applies to a two-vehicle intersection with only one lane on each road.

The system developed consists of the vehicles, where two programmable robots were used, as well as a location system and computers running the ROS code. The robots were then programmed to travel in a self intersecting continuous path. An algorithm was developed which provides priority to one of the robots that travels into the intersection, where the other robot's velocity would be optimized to minimize the control actions and to avoid collision.

The resulting system allowed for the robots to achieve at least 50 consecutive intersections in each trial. Furthermore, proof of the optimization of the movement could be recorded to prove that the algorithm worked as expected.

Bibliography

[1] R. Hult, G. R. Campos, E. Steinmetz, L. Hammarstrand, P. Falcone, and H. Wymeersch, "Coordination of cooperative autonomous vehicles: Toward safer and more efficient road transportation", *IEEE Signal Processing Magazine*, vol. 33, no. 6, pp. 74–84, Nov. 2016, ISSN: 1053-5888. DOI: 10.1109/MSP.2016.2602005.

[2] C. Thompson. (Mar. 2017). The 3 biggest ways self-driving cars will improve our lives, [Online]. Available: http://www.businessinsider.com/advantages-of-driverless-cars-2016-6?r=US&IR=T&IR=T/#roads-will-be-safer-1 (visited on 03/11/2017).

[3] P. Dai, K. Liu, Q. Zhuge, E. H. M. Sha, V. C. S. Lee, and S. H. Son, "A convex optimization based autonomous intersection control strategy in vehicular cyber-physical systems", in *2016 Intl IEEE Conferences on Ubiquitous Intelligence Computing, Advanced and Trusted Computing, Scalable Computing and Communications, Cloud and Big Data Computing, Internet of People, and Smart World Congress (UIC/ATC/ScalCom/CBDCom/IoP/SmartWorld)*, Jul. 2016, pp. 203–210. DOI: 10.1109/UIC-ATC-ScalCom-CBDCom-IoP-SmartWorld.2016.0050.

[4] P. Dai, K. Liu, Q. Zhuge, E. H.-M. Sha, V. C. S. Lee, and S. H. Son, "A convex optimization based autonomous intersection control strategy in vehicular cyber-physical systems", in *Ubiquitous Intelligence & Computing, Advanced and Trusted Computing, Scalable Computing and Communications, Cloud and Big Data Computing, Internet of People, and Smart World Congress (UIC/ATC/ScalCom/CBDCom/IoP/SmartWorld), 2016 Intl IEEE Conferences*, IEEE, 2016, pp. 203–210.

[5] O. Beronius, E. Lundén, M. Malmquist, and A. Rohlin, "Coordination of robots via a wireless network", unpublished.

[6] (n.d.). Trilateration exercise. Accessed March, 2017, U.S. Air Force, [Online]. Available: http://www.gps.gov/multimedia/tutorials/trilateration/.

[7] Y.-x. Yuan, "Step-sizes for the gradient method", Chinese Academy of Sciences, P.O.Box 2719, Beijing, P.R. China, Tech. Rep.

[8] A. Croeze, L. Pittman, and W. Reynolds, "Solving nonlinear least-squares problems with the gauss-newton and levenberg-marquardt methods", LSU, Baton Rouge, LA, USA, Tech. Rep.

[9] C. Keatmanee, J. Baber, and M. Bakhtyar, *Simple example of applying extended kalman filter*, 1st International Electrical Engineering Congress, Chiangmai city, Thailand, 2013.

[10] *Understanding the basis of the kalman filter via a simple and intuitive derivation*, Lecture notes, United kingdom: IEEE, September 2012.

[11] B. Lennartson, "Tillståndsmodeller", in *Reglerteknikens grunder*, B. Lennartson, Ed., vol. 4, Lund, Sweden: Studentlitteratur, 2002, pp. 95–98.

[12] "Rangenet user guide pulson 400 series", Time Domain, Huntsville, AL USA, Tech. Rep. 320-0320C, May 2016.

[13] M. Yavari and B. G. Nickerson. (Mar. 2014). Ultra wideband wireless positioning systems, [Online]. Available: `http://thetoolchain.com/mirror/dw1000/uwb_wireless_positioning_systems_technical_report.pdf`.

[14] *Pulson ranging and communications*, Time Domain, Huntsville, Alabama, 2012, pp. 7–8.

[15] (n.d.). Pulson 440. Accessed Feb 3,2017, Time Domain, [Online]. Available: `http://www.timedomain.com/products/pulson-440/`.

[16] (n.d.). Wifi alliance. Accessed May 11, 2017, Wifi, [Online]. Available: `http://www.wi-fi.org/`.

[17] (n.d.). Pioneer 3dx. Accessed May. 12, 2017, Adept Mobile Robots, [Online]. Available: `http://www.mobilerobots.com/Libraries/Downloads/Pioneer3DX-P3DX-RevA.sflb.ashx`.

[18] (n.d.). Ubuntu wiki. Accessed Feb 7, 2017, Ubuntu, [Online]. Available: `https://wiki.ubuntu.com/`.

[19] *Robotic operating system*, www.ros.org, Accessed May 12, 2017, Open Source Robotics Foundation, May 2016.

[20] (Feb. 2017). Terms of use, Creative Commons, [Online]. Available: `https://creativecommons.org/terms/`.

[21] *Rosaria*, http://wiki.ros.org/ROSARIA, Accessed May 12, 2017, Open Source Robotics Foundation, Mar. 2017.

[22] A. Technology, *Mobilesim*, http://robots.mobilerobots.com/wiki/MobileSim, Accessed May 12, 2017, 2016.

[23] "Global positioning system (gps) standard positioning service (sps) performance analysis report", William J. Hughes Technical Center WAAS T&E Team, Atlantic City International Airport, NJ, Tech. Rep. 96, Jan. 2017.

A

Appendix

A.1 Start-up code and instructions

In this section, the necessary steps needed to run the intersection problem are described.

First three computers are needed with ROS and ROSARIA installed, which means they need to run or emulate Ubuntu. In order to install these, look at the respective wikis. On these computers, open the terminal and change directory to *catkin_ws/src/*. Next use the command *git clone* `https://github.com/Olodus/SSYX02Group2.git` and then go to the *catkin_ws* directory and use *catkin_make*.

In the real world place the anchors on appropriate positions and add the measurements to the *Measure.py* file in the directory *controller/last_year_code/*. Connect two computers to the robots that are going to cross the intersections. Place a UWB on each robot and connect them to the computers. Connect all the computers to the same Wi-Fi network and make sure that they all have the correct bash-files in regards to the IP-numbers. Then connect the computers that are linked to the robots to the central computer by changing the bash-file and writing

```
export ROS_MASTER_URI=http://#IP of central Computer#:11311
```

Start a *roscore* on the centralized computer and then start the servers and RosAria nodes on the computers linked to the robots. Start the RosAria node through

```
rosrun rosaria RosAria __name:=RosAriaN
```

and to start the server, write

```
rosrun controller get_coord_server.py __name:=get_coord_serverN
   get_coord:=get_coordN _ip_of_uwb:=#IP of UWB#
```

The N should be set to 1 for the robot starting in the center of the coordinate

problem and 2 for the one starting displaced.

Now the bash-files can be ran on the central computer to initiate the algorithms and programs. To get the priority problem running, simply use the command

```
bash robot_run.bash -priority
```

A.2 Images of robot setup and UWB

(a) Image of the robot setup

(b) Image of a UWB sensor

Figure A.1: The figure shows two images, one of the robot setup and one of the UWB.

A.3 API and Code documentation

In these sections there will be instructions for how the implemented code is used and what it does.

A.3.1 RobotHandler

The RobotHandler is an abstraction adding additional and more complex commands over the ones already offered by RosAria.

RobotHandler provides a couple of Services representing the different missions.

A.3.1.1 Subscribed topics

Filter/measurements

Topic outputting *nav_msgs/Odometry* messages describing the state of the system. Uses this data for maneuvering.

A.3.1.2 Provided services

/get_state

Returns the current state of the robot as a *Odometry* message.

/is_ready

Returns whether the robot is currently executing a command or not. Is used to poll if a new mission can be sent.

/stop

Ends the mission currently executed by the robot and makes the robot stop any movement.

/set_speed

Sets the linear speed of the robot. Provides free movement paired together with */set_ang_vel* and */set_acc*.

Input arguments:

std_msgs/Float

Returns:

Returns an error message if the robot is already executing a mission (except */set_ang_vel* and */set_acc*). Can be called while executing the */steer_towards* mission A.3.1.2 to control movement along given path.

/set_acc

Sets the acceleration/deceleration of the robot. Provides free movement paired together with */set_ang_vel* and */set_speed*.

Input arguments:

std_msgs/Float

Returns:

Returns an error message if the robot is already executing a mission (except */set_ang_vel* and */set_speed*). Can be called while executing the */steer_towards* mission A.3.1.2 to control movement along given path.

/set_ang_vel

Sets the angular speed of the robot. Provides free movement paired together with */set_speed* and */set_acc*.

Input arguments:

std_msgs/Float

Returns: Returns an error message if the robot is already executing a mission (except */set_speed* and */set_acc*).

/aim_at_point

Aims the robot towards a given point.

Input arguments:

geometry_msgs/Point

Returns:

Returns an error message if the robot is already executing a mission.

/go_to_point

Moves the robot towards to a given point.

Input arguments: *geometry_msgs/Point*

Returns:

Returns an error message if the robot is already executing a mission.

/follow_path

Moves the robot following to a given path.

Input arguments:

nav_msgs/Path

Returns:

Returns an error message if the robot is already executing a mission.

/steer_towards

Puts the robot into "steer"-mode. During this mode, the robot will continuously aim itself as to follow the path formed between the location of the robot when the mission was initiated and the point given as argument. User can call *set_speed* and *set_acc* to freely control the speed of the robot along the path.

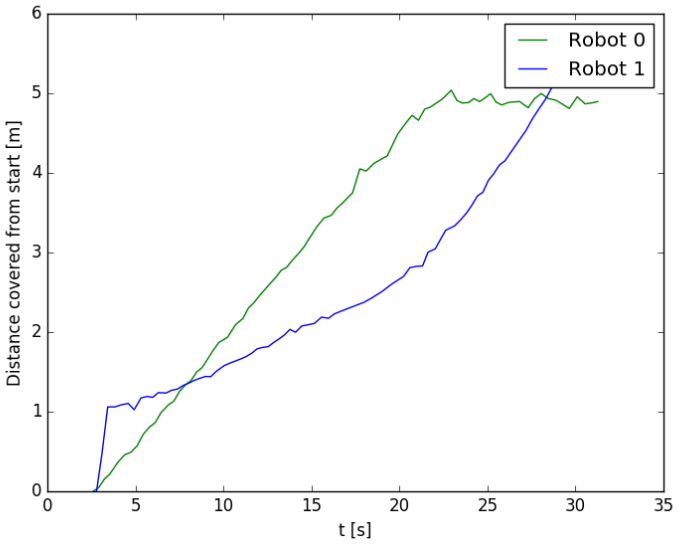
Input arguments:

geometry_msgs/Point

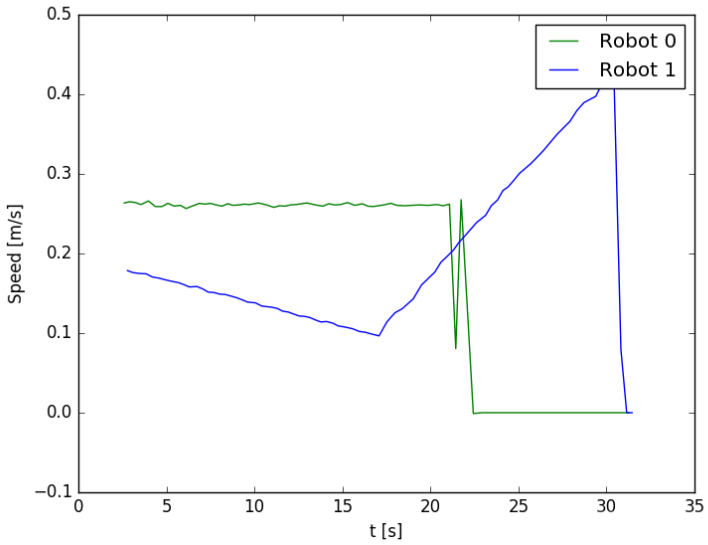
Returns:

Returns an error message if the robot is already executing a mission.

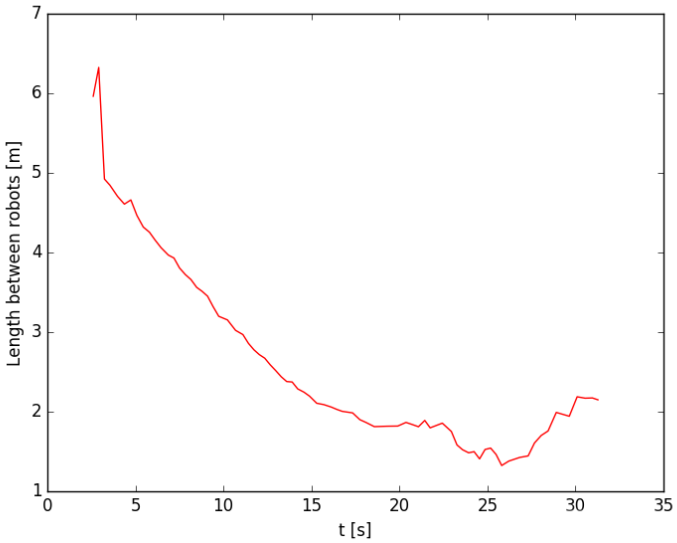
A.4 Additional measurements



(a) The graph shows the total travelled distance of the robots as a function of time. The blue line represents Robot 1 which is the one that is controlled.



(b) The graph shows the different speeds of the robots as a function of time.



(c) The graph shows the length between the robots as they traverse the path.

Figure A.2: The figures A.2a, A.2b and A.2c show different data collected from the same measurement. Figure A.2a plots the distance covered, A.2b is velocity of the robots and A.2c is the distance between them. In this data set, the controllable robot decelerates and reaches the end of the path after the uncontrollable robot.

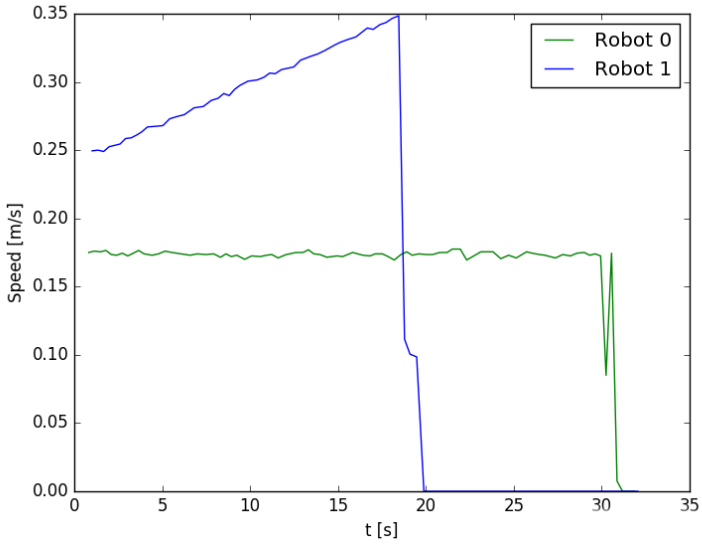


Figure A.3: The figure shows the speeds of the robots as they travel to and through the intersection.