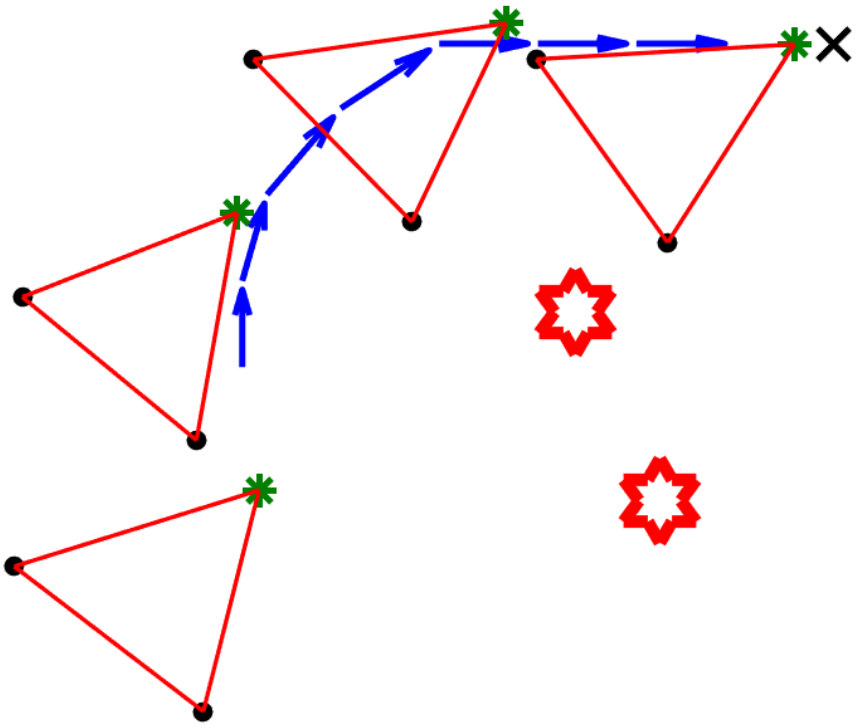
CHALMERS
UNIVERSITY OF TECHNOLOGY



Remote Automatic Control of Robots via Wireless Communication Networks

Formation Control

Bachelor's thesis at the Department of Electrical Engineering

ROBERT BRENICK
NEDA FARHAND
AXEL KÅBERGER
MATTIAS STRID

Supervisor: MOHAMMAD ALI NAZARI, Co-Supervisor: MARKUS FRÖHLE
Examiner: HENK WYMEERSCH

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

Bachelors's thesis 2017:SSYX02-17-82

Remote Automatic Control of Robots via Wireless Communication Networks

Formation Control

Robert Brenick
Neda Farhand
Axel Kåberger
Mattias Strid

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

Remote Automatic Control of Robots via Wireless Communication Networks
Formation Control

Robert Brenick
Neda Farhand
Axel Kåberger
Mattias Strid

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

Bachelors's Thesis 2017: SSYX02-17-82
Department of Electrical Engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Movement of three robots in a formation, along a path around obstacles.

Typeset in LaTeX
Gothenburg, Sweden 2017

Abstract

In this project, we design a controller to remotely move a system of robots from an origin to a target point via an optimal path, while they keep a predetermined formation.

Several pieces of equipment are used during the project. We exploit three robots, a master controller, and a camera serving for positioning the robots. The positioning system is able to extrapolate the position of a number of tags, seen by the camera. The infrastructure also consists of a wireless local area network (WLAN) enabling communication between the controller, the positioning system, and the robots. This communication was handled through a set of open source libraries, going by the common name *Robot Operating System*.

Using PID controllers and a set of movement algorithms, the controller was able to admissibly maintain the robots in formation, as they moved to a target point while avoiding obstacles.

Keywords: Formation, Control, Wireless, PID, Robot, Network, Localisation, GulliView, ROS.

Sammanfattning

I detta projekt utformar vi ett kontrollprogram som trådlöst förflyttar ett system av robotar från deras startpositioner till ett mål längs en distansoptimerad väg, medan de håller en fördefinierad formation.

Utrustningen som används är tre robotar, en central kontrollenhet, och en kamera för att lokalisera robotarna. Lokaliseringssystemet kan extrapolera positionerna hos ett antal etiketter, som i sin tur registreras av kameran. Infrastrukturen består också av ett trådlöst lokalt nätverk (WLAN), vilket möjliggör kommunikation mellan kontrollenheten, lokaliseringssystemet, samt robotarna. Denna kommunikation sköttes av en uppsättning bibliotek, vilka går under det gemensamma namnet *Robot Operating System*.

Genom att använda PID-regulatorer och en uppsättning kontrollalgoritmer, kunde kontrollenheten på ett tillfredsställande sätt upprätthålla robotarnas formation, då de förflyttade sig till ett mål samtidigt som de undvek hinder.

Keywords: Formation, Kontroll, Trådlöst, PID, Robot, Nätverk, Lokalisering, GulliView, ROS.

Acknowledgements

We would like to thank our main supervisor Mohammad Ali Nazari for guiding us, as well as Markus Fröhle for his valuable input throughout the project. Thomas Petig has also given us a lot of help with the camera positioning system which we are very grateful for.

The support from the other Bachelor and Master groups at Communication and Antenna Systems has been vital for this projects success.

And of course, we owe great gratitude to Henk Wymeersch, our examiner.

Robert Brenick, Neda Farhand, Axel Kåberger and Mattias Strid

Gothenburg, May 2017

Contents

List of Figures

1

Introduction

Autonomous robotics has been around since the middle of the 20th century, with the earliest being the robot tortoises Elmer and Elsie who were built to autonomously find their way to the recharge station [1]. Although slow, the robotic reptiles paved the way for the technology to quickly develop for a wide range of applications. After a couple of decades autonomous industrial robots were made available, revolutionising how manufacturing is done. This marked the beginning of an era, in which robots have an ever increasing impact on our daily lives. Since then, much of the development has focused on designing robots to do tasks deemed either too dangerous or tedious for people. One example from the current frontier is the use of drones to clear dust from solar panels, sparing people from working in often dry and inhospitable places [2].

One of the most sought-after goals in autonomous robotics is to create affordable and safe autonomous vehicles. Many prototypes already exist, and several companies aim to have these cars on the market within the next couple of years [3]. The large investments in this area are benefiting autonomous robots in general, as some vital parts can be produced at a fraction of the original cost. One example is LIDAR-systems[1], whose cost has dropped by more than 99% during the last 5 years [4]. These developments and drops in prices have made larger systems viable, consisting of multiple robots cooperating to complete tasks much faster than humans could. Companies like Amazon utilise such systems to sort and send packages with increased rates and decreased costs [5].

We want to add functionality to a system of logistic-robots, similar to that used by Amazon. As of now, mainly small robots are used to transport small packages – however, robots of similar sizes could carry a much larger object if organised in the correct way.

1.1 Aim and scope

The purpose of this project is to design a controller that moves three robots to a point, taking the shortest possible path while keeping a predetermined formation. It is desired that the resulting system shows that a formation of robots could be a viable way of transporting large objects in a limited area. Being in an enclosed environment allows the system to utilise a global coordinate system, where the coordinates of every robot can be sent to a master controller with negligible delay. The tracking of positions is done by a camera connected to the network. Using the camera data, the controller does calculations

[1] Light Detection and Ranging, used to determine distances to objects with pulsed laser beams.

and send control commands to the robots. The system is leader-based, utilising a position-based formation control.

1.2 Problem description

The problem is divided into subproblems. A group of robots are connected to the controller using suitable hardware and software. Then, a system to track the positions of the robots is implemented. Finally a program is developed to have the robots move along an optimised path, keeping a formation that can carry a larger load than the robots can carry individually.

2

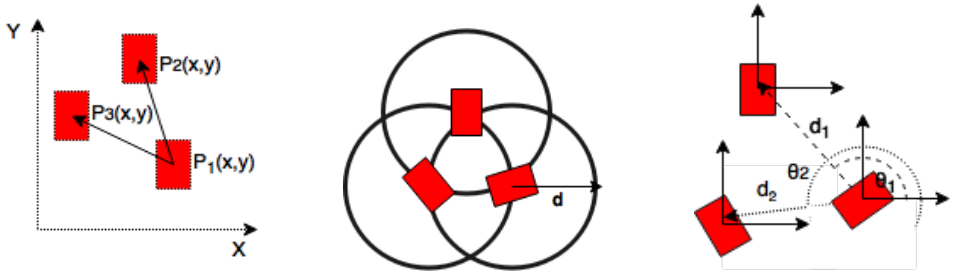
Theory

This section describes the main theory used during this project. This includes how the robots can be organised, how a PID controller functions, and how a path planning algorithm should be exploited to avoid obstacles while optimising the path. Lastly, a movement algorithm that allows a robot to drive and rotate at the same time is described.

2.1 Variations of formation control

Depending on what information the robots receive and how they interact within the system, there are different ways to approach formation control. Interaction is mainly done in one of two ways – namely leader-based control, or swarm control. In a leader-based system, one robot is assigned to be the leader. The leader receives all relevant information on where the formation should move, disregarding the locations of the other robots. Meanwhile, the remaining robots are assigned follower status, with the single objective to keep the formation intact.

In contrast to a leader-based system, a swarm-controlled one can vary more in its implementation, although in general all the robots share a set of behaviours in how they attract and repel each other. These behaviours can be manipulated in order for them to reach consensus on a planned formation through independent calculations. An advantage of doing the calculations independently is that they can be done at the same time, enabling swifter formation. While the swarm can easily mobilise into a formation, the lack of individual commands each robot receives makes it difficult to move the formation without



(a) Position (Global) **(b)** Distance (Inertial) **(c)** Displacement

Figure 2.1: The three different kinds of controllers according to [6], as well as their required coordinate-system.

heavy calculations [7]. In a leader-based system these calculations are made immensely simpler, allowing for more precise control of the system and making it the preferred system for this project.

According to a survey by K.-K. Oh et al. [6], the amount of information available from the system results in three types of controllers. The most stable, but also most taxing on the system, is a *position-based* controller (figure 2.1a) where each robot is aware of its position in a global coordinate system. This allows the robots to receive precise information about their target position and enables them to plot a path with ease. This will, however, require a stable stream of position data, which is often difficult to achieve. On the opposite side of the spectrum is the *distance-based* controller (figure 2.1b), where the robots only know the distances between themselves and the other robots. This gives them a wider range of possible sensing capabilities, but increases the number of required interactions between the robots for proper control, making system wide cooperation more complicated. The middle-ground is a *displacement-based* controller, shown in figure (2.1c), where the robots know not only the distances, but also the angles between itself and the other robots. This can be done either by using a global coordinate system, or by aligning the robots inertial systems, depending on the infrastructure of the system.

2.2 PID controller

A classic type of controller that is widely used for different types of systems is the PID controller [8]. A single-input, single-output (SISO) PID controller has one reference signal, denoted by r, and one system signal, denoted by s. Having fed the input data, the proportional part P, the integral part I, and the derivative part D, give an output signal to be used by the system to reach the reference. The P part takes care of the difference between r and s at the moment, the I part handles differences over time and the D part deals with the predicted future.

The contribution of each part is scaled by a constant, which we denote by K_P, K_I and

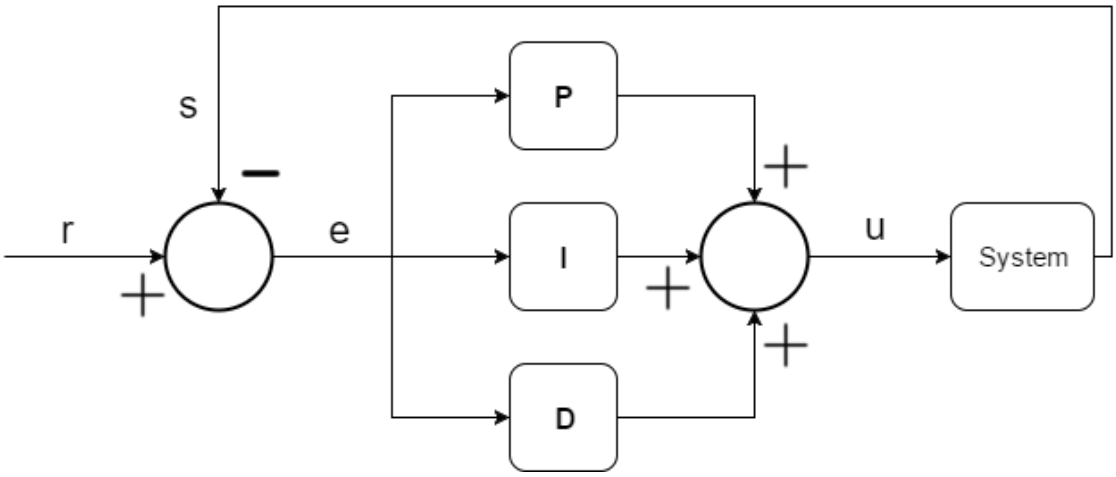


Figure 2.2: Flowchart for a single-input, single-output closed-loop PID controller. The inputs to the controller are the current state of the system and the desired state as the reference. The output of the PID controller is the control signal.

K_D where the indices stand for the corresponding part. By adjusting these values, the behaviour of the system can be modified.

In figure 2.2 a flowchart of a closed-loop control system is shown. The error, e, at time t is defined as the difference between the reference and system signals:

$$e(t) = r(t) - s(t) \tag{2.1}$$

Using this error, the output $u(t)$ is described as:

$$u(t) = K_P e(t) + K_I \int_0^t e(\tau)d\tau + K_D \frac{de(t)}{dt}. \tag{2.2}$$

In terms of tuning the PID controller to behave in a specific manner, incrementing the K_P value gives in general a quicker response, but also increases overshooting. To reduce this overshoot K_D can be increased, while increasing K_I gives a more stable output and allows the follower to catch up with the leader if it is constantly trailing behind with a larger distance than desired.

2.3 Path finding and shortest-path algorithm

Algorithms can be used to determine the shortest path from one node to another in a graph, albeit varying in efficiency depending on the scenario[9]. Supposing there is a graph representing a map, on which an optimal path from one point to another is desired. Where there is nothing blocking a straight line between the starting point and the goal, one algorithm may seem better than another. However, in the presence of obstacles it is no longer a matter of simply drawing a straight line, and the same algorithm may no longer be preferable.

Two kinds of algorithms, *Djikstra's algorithm* and a *greedy algorithm*, fulfil different requirements of those that are needed to determine the shortest possible path from a starting point to a specified goal[9]. For more information regarding these algorithms, see appendix C. A third algorithm, the *A*-algorithm*, combines the reliability of Djikstra's algorithm with the narrowness of a greedy algorithm.

2.3.1 A*-algorithm

The A*-algorithm, first described in [10], aims to expand[1] the least possible number of nodes, while finding the shortest possible path from one point to another. This means that it needs to determine which node has the highest or least potential in leading to the optimal path.

The algorithm uses a function that estimates what total cost travelling via a certain node will lead to. The actual cost of travelling by an optimal path via a node n is denoted

[1] "Expanding" a node, in this context, means visiting its neighbours, but not necessarily adding them to the path.

$f(n)$, and is the sum of $g(n)$; the actual cost of travelling by an optimal path to n from the node of origin, and $h(n)$; the actual cost of travelling by an optimal path from n to a goal node.

Estimation of $f(n)$ is done by an evaluation function, $\hat{f}(n)$, when expanding any node, n, such that:

$$\hat{f}(n) = \hat{h}(n) + \hat{g}(n), \tag{2.3}$$

where $\hat{g}(n)$ and $\hat{h}(n)$ are the estimates of $g(n)$ and $h(n)$ for n.

The algorithm works in such a way that it favours nodes with the smallest $\hat{f}$ value as potential successors to a prior node. Generally described, the algorithm works as follows:

Beginning with the starting node as the "current" node n, it is expanded, and the algorithm proceeds to go through each of the node's neighbours – estimating their $\hat{f}$ values. Throughout the process, it is kept track of which of the neighbouring nodes offers the least estimated cost, as well as which nodes have already been visited. The neighbour with the smallest $\hat{f}$ value amongst all neighbours of the current node is then marked as the successor to the current node. Following this, the currently expanded node is closed, and the node designated as its successor is marked as the current. This process is repeated until the current node is the goal node.

The A*-algorithm applies the dynamic choice process where a greedy algorithm does not, and resembles Djikstra's algorithm in that they both dynamically choose which node to expand next. While Djikstra's algorithm finds the shortest paths to all other nodes in a graph from a certain starting node, the A*-algorithm uses only the resources needed in finding the path to one specific node.

2.3.2 Graph model and representation of the map

A graph is a potent way of representing a map. Since each node can store desirable properties, they can hold information of what set of coordinates they represent. That way, the node can also give some indication of whether it is an obstacle or not. Allowing each node to know which of the adjacent nodes within the map are accessible fulfils the same purpose as edges. The direction of the edges becomes inconsequential, since there are no limits to the direction in which movement can take place without consideration of obstacles and physical boundaries.

Using a binary matrix as a model simplifies passing information through the algorithm about where the obstacles are and which nodes are neighbours. The matrix acts as both a graph and a map of the room in two dimensions holding some number of nodes, where the density of the nodes varies depending on the level of accuracy desired. The nodes are temporarily assigned indices representing their coordinates, which is done by shifting and scaling the coordinates into their corresponding places in the matrix. For instance, a room of a certain size $a \times b\,\mathrm{m}^2$ with a density of n nodes per metre would be represented by the

following matrix:

$$R = \begin{bmatrix} x_{0,0} & \cdot\cdot & x_{0,n(b-1)} \\ \cdot & \cdot & \cdot \\ x_{n(a-1),0} & \cdot\cdot & x_{n(a-1),n(b-1)} \end{bmatrix} \tag{2.4}$$

The node $x_{i,j} \in \{0, 1\}$ is on the i^{th} row and in the j^{th} column of the matrix. A value of 1 represents an obstacle being on the corresponding coordinates in the room, while a value of 0 represents an accessible position.

Thus, the neighbouring nodes to which there exist legitimate edges for any node are the nodes directly adjacent to it in the matrix – both horizontally, vertically, and diagonally, for all such nodes that are accessible and within range of the matrix.

Supposing $\mathbf{A}$ is a set of nodes that are accessible and within range of the matrix, and the neighbouring nodes of the node x_{ij} is denoted by $\mathbf{B}_{x_{ij}}$, such that:

$$\mathbf{B}_{x_{i,j}} = \{x_{i-1,j}, x_{i-1,j-1}, x_{i,j-1}, x_{i+1,j}, x_{i+1,j+1}, x_{i,j+1}\} \tag{2.5}$$

It is possible, using a third set of nodes, $\mathbf{C}$, to state that legitimate edges exist only between x_{ij} and nodes in $\mathbf{C} = \mathbf{A} \cap \mathbf{B}_{x_{ij}}$.

Since all nodes and edges can be defined, the model translates into a graph that is compatible with the algorithm described in 2.3.1.

2.4 Smooth movement

In order for a robot to move to a designated target in two dimensions, it has to align itself in the correct orientation and move towards the target. There are two general methods for moving to the target; one where the robot is first rotated into the correct orientation and then moves forward, and the other where the robot rotates and moves forward simultaneously.

The second method is what we will call a *smooth movement*, which results in a continuous, smooth curve for the robot to follow and allows it to move forward immediately. According to O. Beronius et al. [11], who did a study on the same robots used in this project, the smooth movement they developed is favourable over the alternative, hence this is the algorithm that will be used in this study. The maths for calculating the angular and linear velocities for achieving the smooth movement to the target is described below.

First of all, a two dimensional Cartesian coordinate system has to be set up. The robot's position and the target position as well as the robot's orientation in this space has to be known. The distance between the robot and target is denoted by d. The angle ϕ is defined as the angle to the target position from an axis parallel to the positive x-axis, originating from the robot's position. The angle θ is the orientation of the robot, calculated from the same axis. The angles are shown in figure 2.3. The resulting goal angle for the robot to align itself according to from its current orientation is given by:

$$\alpha \equiv \phi - \theta. \tag{2.6}$$

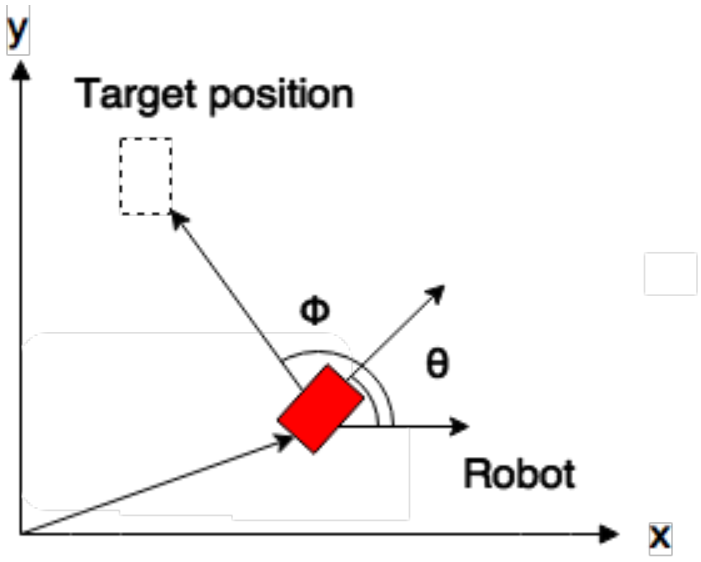


Figure 2.3: The angles used to calculate how the robot has to rotate in order to align itself with the target. The angle ϕ is the angle to the target from the robot and θ is the orientation of the robot, both given from an axis parallel to the positive x-axis, originating from the robot's position.

The angle α is shifted by 2π in order to take a value in $[-\pi, \pi]$. Since, for example, a rotation by π gives the same final orientation as a rotation by 3π, the robot does minimal rotation to align itself. Then α can be used to achieve the smooth movement. The forward movement scale factor:

$$\chi \equiv \frac{\pi - |\alpha|}{\pi} \tag{2.7}$$

takes values between 0 and 1. The smaller the magnitude of α is, the robot is almost in the correct orientation, the bigger χ gets. When the robot is almost in the opposite direction, χ tends to zero.

The linear and angular speeds are then given as:

$$v_{lin} = d \cdot \chi$$
$$v_{ang} = \alpha,$$

which are used by the robot when moving to the target position. These values have to be continuously recalculated in order to achieve the smoothest possible movement.

3

Hardware and Software

There are two primary hardware components. The first one consists of three mobile robots from Omron Adept tech. being able to receive control actions from a laptop. The laptops are running the Robot Operating System (ROS), which is used as the primary software platform for control and communication between the robots. The second component is the positioning system, which is using a web camera to read visual markers that are analysed in a computer vision program called GulliView. The code produced during the project has been written in Python, except for some minor changes to GulliView, which is written in C++.

3.1 Hardware

Presented below is a list of the main hardware components required to set up the project.

- **Omron Adept MobileRobots Pioneer P3-DX**
  Although the P3-DX robots [12] can support a wide range of optional accessories such as laser sensors for navigation and robotic arm manipulators, they are only used as mobile agents in this project. Any other similar robot with a ROS interface could be interchangeable with the P3-DX, with only minor modifications to the code base. Each robot has two differential-drive motors with 500-tick encoders and can reach a top speed of 1.2 m/s. The dimensions of the robots are $455 \times 381 \times 237$ [mm].
- **Logitech Webcam C930e**
  When using the GulliView vision-based localisation interface, at least one camera is required for the video stream. The Logitech Webcam C930e is well suited for tracking due to its wide field of view (90°), sufficiently low distortion, and a resolution of 1920×1080.
- **Thomson TG784 Wi-Fi router**
  The router used to set up a local Wi-Fi network, that enables communication between the controller and the robot computers.
- **Ubuntu laptops**
  In this project five different laptops running Ubuntu 16.04 LTS *(Xenial Xerus)* were used. One for each robot, connecting it to the network and providing control commands, one for running the GulliView tracking software and server, and one running the master control algorithm.

3.2 ROS and ROSARIA

ROS [13] is a wide-spread software platform used in many research projects involving robotic systems. It is used to implement low-level device control, hardware abstraction, and as a way to handle communication between different processes running in the system.

ROSARIA [14] is a node running in ROS, utilising the Advanced Robot Interface for Applications (ARIA) from MobileRobots, to create a ROS interface with the P3-DX robots. It supports various control actions such as setting the robot's linear velocity, angular velocity, and other motion parameters. It is also used to receive sensor information from the robot platform.

3.2.1 ROS fundamentals

In this section, a breakdown of the different fundamental building blocks of a robot system implemented with ROS is provided. The information here is a summary of the extensive information found on the ROS wiki[1].

- **Core**
  A ROS core is the foundation on which the system runs. It provides naming services, a parameter server and system-wide logging sent to the topic */rosout*. A core is the first thing that has to be run when starting a ROS-based system and only one core can run at the same time.
- **Nodes**
  Nodes are small isolated computational services, that are linked together similar to a graph, where the edges in the graph are topics that carry messages between nodes. They can be seen as individual programs with a ROS outer layer that exposes an API[2]. This means that new nodes can be easily embedded into the system. It also means that a ROS system is not limited to a specific programming language, so that nodes can run Python or C++ scripts.
- **Messages and topics**
  Messages sent over topics is the method used for communication between nodes. Nodes subscribe to the topics which are unidirectional, i.e. nodes have the ability to both subscribe and publish to a topic. Messages are type defined, either as a ROS standard message or a user defined message. They are set up as data structures containing primitive data types, or arrays of primitive types.
- **ROSARIA**
  ROSARIA is the node that handles communication between ROS and the robots' ARIA interface. Each robot has its own instance of a ROSARIA node, running with unique parameters such as name and IP address in the network. The ROSARIA node is able to subscribe to the *cmd_vel* topic, which contains a message called *Twist*. The *Twist* message contains two different types of parameters for controlling

[1] wiki.ros.org
[2] Application Programming Interface.

the robot; *linear* and *angular*. The *linear* parameter sets the forward speed in meters per second, and the *angular* sets the rotational speed in radians per second.

3.3 GulliView

GulliView [15] is a system used for visual localisation developed at Chalmers University of Technology as a cost efficient alternative for autonomous vehicle control and positioning on small scales. The system consists of a camera mounted over the test area, and a computer running the GulliView program. A ROS node on the same computer is publishing the position data as a topic on the local Wi-Fi network with an update rate of 30 ± 2 Hz. The markers GulliView are able to track are called AprilTags [16]. The accuracy was well with in the requirements but never thoroughly measured in this project. An extensive test of the GulliView system were carried out by another bachelor's project [17] in 2016. Each robot has two AprilTags on top of themselves – one used for the position of the robot, and the other giving the heading of the robot relative to the first tag.

To set up the GulliView system, four AprilTags are placed on the driving surface in a quadrilateral square used as a base for a transformation matrix. This transformation matrix allows GulliView to see tags from an angle and still be able to correctly localise them in the driving plane. The distance between the base tags represent the size of the unit GulliView uses, where each axis is set to be 1000 units long. Strategical placement of the tags one meter apart allows the software to publish the position data where one unit equals one millimetre.

4

Method

The formation movement required several different sub-systems to work together. The communication between the robots had to be set up, and their positions had to be found. Using the position data, a controller could make decisions on the path for the leader and the follower movement to keep the formation. The methods used to build these systems are described in this section.

4.1 Communication

Several communication methods and protocols were applied in this project. RS-232 serial communication was used between the physical robots and their respective computer running ROSARIA. The ROSARIA computers, master controller and GulliView computer were connected to a Wi-Fi network. They communicated with each other using messages sent over ROS topics, as described in section 3.2.

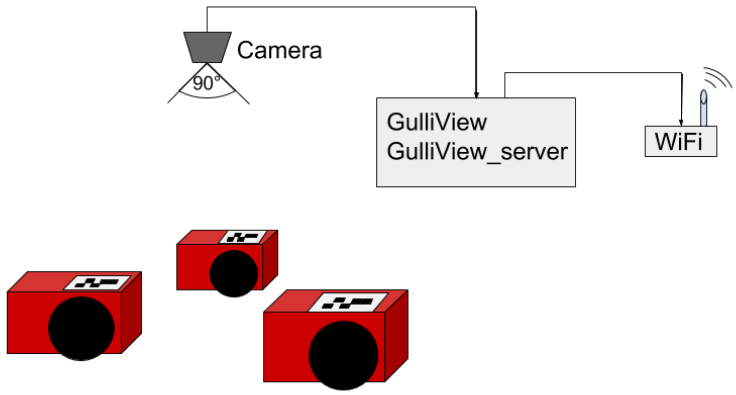


Figure 4.1: GulliView system set up with one camera, seeing three robots with AprilTags on top. The tags are tracked by the GulliView program and their positions are then sent from the GulliView server to the controller.

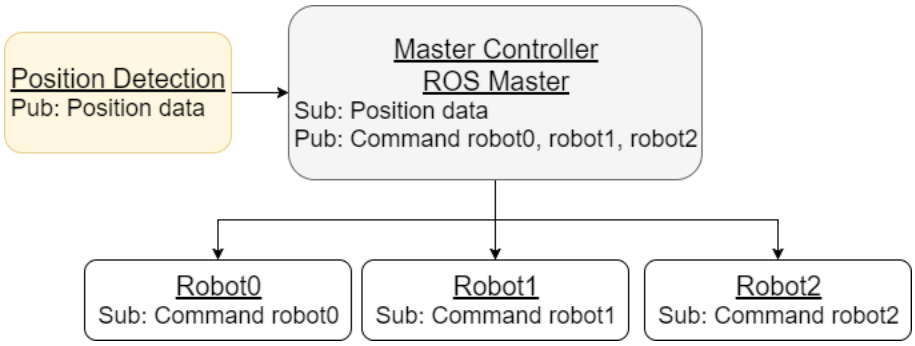


Figure 4.2: Visualisation of how the ROS-nodes interact within a local network.

4.1.1 System setup

Figure 4.1 presents how the network and camera is set up. The figure also shows three robots with AprilTags on top, used for collecting position data with GulliView. Figure 4.2 provides an overview of how the nodes communicate as publishers and subscribers. The master controller subscribes to a topic published by the camera node, which gives the AprilTags' positions. This data can then be used to determine the robot's positions and orientations. The controller does the necessary movement calculations and publishes the resulting control commands to three topics, each intended for one specific robot.

4.2 Formation control

Among the different variations of controllers described in section 2.1, a leader- and position-based controller was selected. It is well suited due to the stability of the system network, allowing access to a constant and precise stream of position data in a global coordinate system.

The formation control algorithm is divided into several parts. First a leader is determined based on the robots' positions and the goal. Then the shortest path to the goal for the leader, in regard to obstacles that the formation has to avoid, is calculated. The path is represented by a list of interim target positions that the leader goes to using the smooth movement algorithm. From the information about the positions of all robots in the system and the orientation of the leader, the followers are using two PID controllers each to adjust their relative distances to both the leader and the other follower. The process is described in detail below.

4.2.1 Setting the leader

As mentioned above, the first thing that has to be done is setting the leader. This is not necessary for the system to work, but when choosing the robot that is the closest to the final target position, it will move ahead of the other robots in the system. In this way the inverse pendulum problem, which easily can lead to unstable systems, is avoided. Since all of the robots in this study are of the same type, there are no new problems arising when

changing the leader. The algorithm to set the leader is simple in its design – the only things considered are the distances of each robot to the goal, which are calculated using position data from the camera positioning system. The robot that is the closest to the target position is assigned to be the leader, while the others are assigned to be followers.

4.2.2 Path finding

When the role of the leader had been assigned, a representation of the room was given through a binary matrix, as described in 2.3.2. This matrix was generated using input from the camera. The observed system of coordinates was divided into a grid where each square was represented by an element in the matrix. Obstacles were represented by designated AprilTags.

A practical example using a $4 \times 5\,\mathrm{m}^2$ map and setting a node density of 1 node per metre would generate the following matrix:

$$R_{4\times5} = \begin{bmatrix} x_{0,0} & x_{0,1} & x_{0,2} & x_{0,3} & x_{0,4} \\ x_{1,0} & x_{1,1} & x_{1,2} & x_{1,3} & x_{1,4} \\ x_{2,0} & x_{2,1} & x_{2,2} & x_{2,3} & x_{2,4} \\ x_{3,0} & x_{3,1} & x_{3,2} & x_{3,3} & x_{3,4} \end{bmatrix}. \tag{4.1}$$

While $R_{4\times5}$ is generally described as in matrix 4.1, it would in a specific case depict the values of all nodes according to what the camera has perceived. For instance, if there are AprilTags representing obstacles placed across nodes x_{11}, x_{21}, x_{31}, x_{12}, x_{22}, and x_{32}, R_a would appear as in the matrix below:

$$R_{4\times5} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \end{bmatrix}. \tag{4.2}$$

Following the matrix generation, and when given a goal, a path was plotted using an implementation of the A*-algorithm. The A* implementation used was written in Python 2.7 by C. Careaga [18]. The principles of the algorithm are described in section 2.3.1. This found and reconstructed the shortest path from the starting point to the goal while avoiding all obstacles. The path was presented as a list containing coordinates, which were along the path leading to the goal as calculated by the A*-algorithm.

In the case of $R_{4\times5}$, when stating for instance $x_{3,0}$ as the node of origin and $x_{0,4}$ as the goal node, the optimal path in the form of indices returned by the algorithm would be:

$$path_{R_{4\times5}} = \{(3,0),(2,0),(1,0),(0,1),(0,2),(0,3),(0,4)\}. \tag{4.3}$$

$$R_{4\times5_{path}} = \begin{bmatrix} 0 & + & + & + & G \\ + & 1 & 1 & 0 & 0 \\ + & 1 & 1 & 0 & 0 \\ S & 1 & 1 & 0 & 0 \end{bmatrix} \tag{4.4}$$

The matrix 4.4 illustrates the plotted path, with S marking the starting node, G marking the goal, and the remaining nodes belonging to the path marked with +.

4.2.3 Leader movement

The smooth movement algorithm (sec. 2.4), was applied to the movement of the leader as it sequentially travelled to the coordinates provided by the path finding algorithm. The leader received a linear speed to go with, which was scaled with the forward movement scale factor χ (eq. 2.6) for all target points except the final one. Going to the final target point, the distance left was also considered, allowing a smooth stop. The movement of the leader was done in disregard of the followers positions, making it desirable to move the followers into formation before moving the leader.

4.2.4 Follower movement

Making sure that the formation is kept without knowledge of the future gave rise to many problems. If the actual path of the leader was known to the followers it would be trivial to calculate a path for the followers to maintain the formation, but this was not the case. Instead the information available was the data from the camera positioning system which are positions and orientations of the robots.

By using two PID controllers; one for the distance to the leader and one for the distance to the other follower, the follower's position could be controlled. The inputs to the PID were the desired distance to the other robot, and the actual distance between the robots. From this, a control output was calculated as described in the PID controller theory section, see 2.2. In general the idea is that the output reflects how far away from the correct distance to the other robot it is, has been and will be in the future. These control outputs are denoted by u_l and u_f for a follower to a leader and another follower respectively.

The robots were only allowed to move forward, which means it is desired to have a system with no overshoot. Overshooting can be minimised by tuning the constants of the PID controller. Still, overshooting occurred in some cases, which were handled by slowing down the follower until it fell out of formation again. Hence, turning around was not allowed when the followers were getting too close to the leader.

The control outputs u_l and u_f were used to decide in which direction to go. To illustrate the idea, some examples will be given before a more rigorous mathematical approach is introduced. If u_f has a large value compared to u_l, it is desirable to move towards the other follower. If instead both u_l and u_f are small, the follower is more or less at the correct distance from both the other robots. Therefore, it is desirable to orientate itself in the orientation of the leader and go in this direction, since we know that the leader is moving forward.

The ideas above are realised by building a direction vector for the follower, denoted by $\vec{v}_\mathrm{dir}$. To find $\vec{v}_\mathrm{dir}$, three vectors have to be used; the vector from the follower to the leader, $\vec{v}_\mathrm{l}$, the vector from the follower to the other follower, $\vec{v}_\mathrm{f}$, and a vector representing the orientation of the leader, $\vec{v}_{\mathrm{ori\_l}}$. All of these vectors are normalised so that they are of equal magnitude, before scaling them with u_l and u_f. The scaled vectors are then added together, which gives:

$$\vec{v}_\mathrm{dir} = \vec{v}_\mathrm{l} \cdot u_\mathrm{l} \cdot u_\mathrm{f} + \vec{v}_\mathrm{f} \cdot u_\mathrm{f} + \vec{v}_{\mathrm{ori\_l}} \cdot \exp\Big(- \sqrt{u_\mathrm{l}^2 + u_\mathrm{f}^2}\Big). \tag{4.5}$$

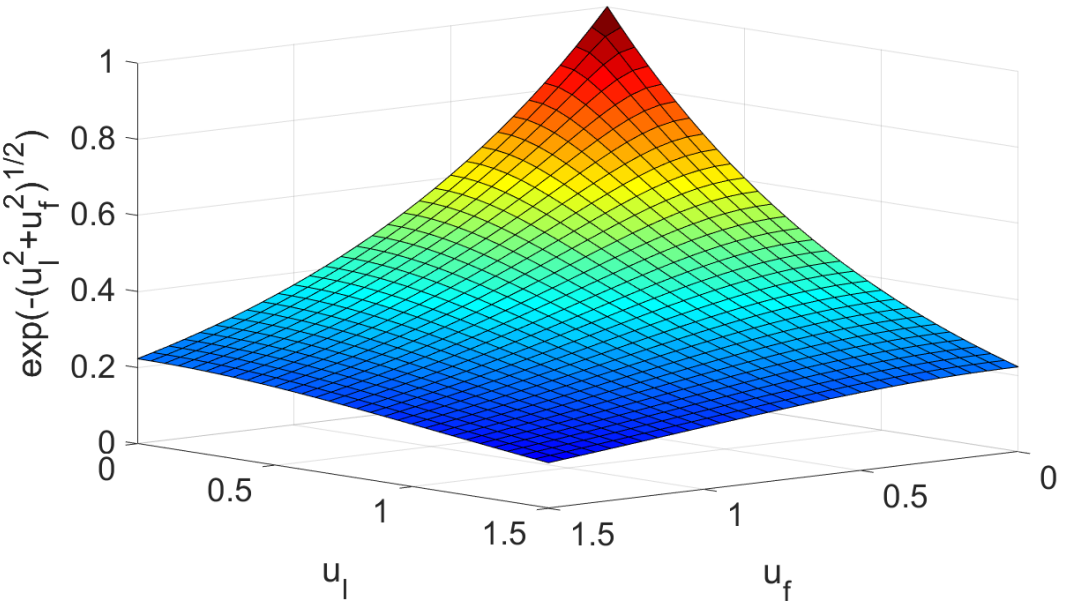


Figure 4.3: Visualisation of the scaling factor for going in the leader's orientation in regard to the control commands u_l and u_f received from the PID controllers.

The exponential term scaling the orientation vector of the leader takes values between 0 and 1. It tends quickly to 0 as soon as this follower is getting far away from one of the other robots. It is only when both u_l and u_f are small this scale factor is close to 1 – a graphical visualisation of the exponential term is shown in figure 4.3.

When the direction is calculated, the robot gets the *total control command*

$$u \equiv \sqrt{u_\mathrm{f}^2 + u_\mathrm{f}^2}, \tag{4.6}$$

which multiplied by the forward movement scale factor χ (eq. 2.6) gives the linear speed in meters per second the robot should go with.

4.3 Testing

This section describes the methods used to test the developed algorithms and produce the results latter presented in this report. The testing was done as an iterative process; firstly the PID controllers were tuned, then the speed of the leader was examined, and finally the formation movement around obstacles was studied. For all tests, the formation was set as an equilateral triangle with a side length of 1 m.

4.3.1 Tracking setup

During the development phase, a test setup to quickly run new code on a working system was built. In this prototyping setup, a camera was mounted on the ceiling 305 cm from

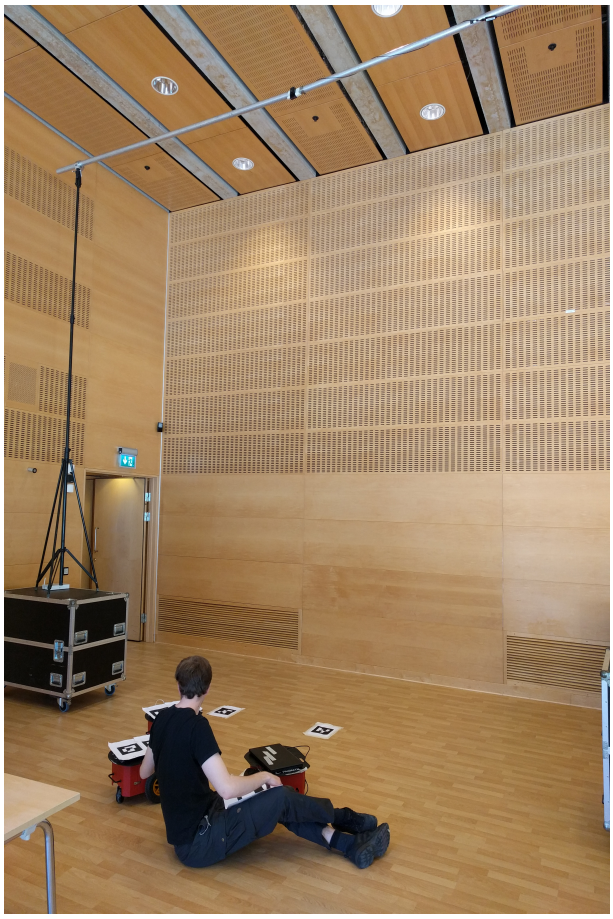


Figure 4.4: Overview of the camera tracking system. The camera is mounted on an aluminium pipe suspended between two tripods standing upon two cases. The estimated height of the camera is $\approx 5.3\,\mathrm{m}$.

the floor which resulted in a tracking area of $2.8 \times 2\,\mathrm{m}^2$. While this was too small to run any large scale test, it was enough to validate that the concept of the code worked on a physical system.

Latter on, a larger test area was necessary. While one way of expanding the tracked area was by adding more cameras to the system, another approach was chosen. This way only one camera was used, but in return a room with a high ceiling was required. This resulted in a usable test area of $4.8 \times 4\,\mathrm{m}^2 = 19.2\,\mathrm{m}^2$ seen in figure 4.4. The reason behind using only one camera is discussed in section 6.1.2.

4.3.2 PID tuning

First of all, the optimal values for the PID controller had to be found. The definition for being the optimal values in this study, are those that give a short time for the followers to get into position and a minimal overshoot of the distances in between the robots. For investigating these things, a program was written to collect data from the camera positioning system.

To decide the values of K_P, K_I and K_D an experiment was set up where the robots always started in the same position and direction. The leader drove with the same speed, 0.1 m/s, to the same point. Only the PID constants were changing in between two measurements. Since the size of the robots is about $40 \times 40\,\text{cm}^2$ (sec. 3.1), transporting things on top of the robots on a movable surface allows the object to stay in place – even if the robots fall out of formation within a range. An acceptable range for still being considered in formation was set to ± 10 cm, since this margin is well within the dimensions of the robots.

By iterating through different values of the PID constants, and studying the resulting behaviour, the values could be tuned. To determine how to change the values, the ideas presented in section 2.2 were used.

4.3.3 Speed of the leader

Another thing that could change the conditions on the system is the leader's speed. Since the followers are using PID controllers that are constantly counteracted by the moving leader, the magnitude of this movement might have an impact. By setting up an experiment with all things kept the same, apart from the speed of the leader, the effects of this can be examined.

4.3.4 Formation control

When the PID values were set and the speed of the leader had been examined, the formation algorithm could be tested. From the position data, the distances in between the robots could be calculated for every update from the camera. Using the acceptable range for being in formation of ± 10 cm defined above in 4.3.2, the time spent in formation could be displayed as a percentage of the total time. Obviously these results will depend on the initial conditions of the system, the path the leader takes and other parameters not controlled by the followers.

5

Results

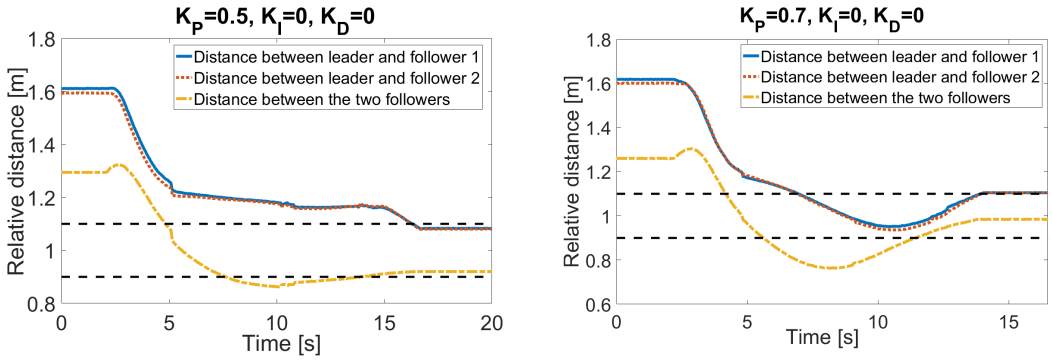
This section shows the tuning process of the PID controller and the results of the experiments outlined in the previous chapter.

5.1 PID tuning

The full list of tests can be found in table 5.1 along with the measured settling time and information about overshooting. Those that give a representative image of the tuning procedure can also be found in figures in this section. The column "Max overshoot" in the table gives the biggest overshoot registered for any distance between two robots. Note that values on the PID constants that gave an unstable system, for example robots almost colliding, uncontrollably oscillating, or similar are not included. As mentioned in the testing chapter (sec. 4.3), the desired distances in between the robots were all set to 1 m, and the acceptable margin to ±10 cm for being considered in formation.

Table 5.1: The behaviour of the system in relation to different values for the PID constants. For each set of values the time it takes to achieve a formation within the admissible margins, and maximum overshoot in the distance between any of the robots are shown.

K_P	K_I	K_D	PID scale to leader	Time to formation	Max overshoot
0.5	0	0	1	-	14 cm
0.6	0	0	1	11 s	21 cm
0.7	0	0	1	9 s	24 cm
0.8	0	0	1	-	39 cm
0.4	0	0.0002	1	-	50 cm
0.6	0.8	0.0002	1	-	11 cm
0.6	0.8	0.0003	1	-	31 cm
0.4	1	0.0002	1	-	11 cm
0.6	1	0.0001	1	-	10 cm
0.6	1	0.0004	1	-	3 cm
0.6	1.4	0.0004	1	9 s	2 cm
0.6	1.2	0.0003	1	-	4 cm
0.6	1.4	0.0003	1	-	6 cm
0.7	1.3	0.0003	1	12 s	-
0.7	1.3	0.0003	$K_P \cdot 1.2$, $K_D \cdot 1.05$	5 s	2 cm



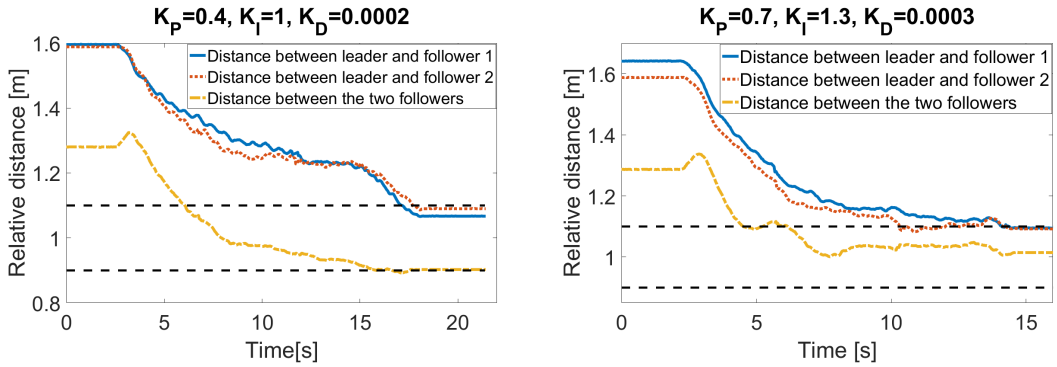
(a) $K_P = 0.5$, $K_I = 0$, $K_D = 0$ **(b)** $K_P = 0.7$, $K_I = 0$, $K_D = 0$

Figure 5.1: The distances in between the robots over time for the two different sets of PID values. The dotted black lines show the acceptable margin for being in formation.

In figure 5.1a the relative distances between the robots over time are shown, using PID constants with values of $K_P = 0.5$, $K_I = 0$, and $K_D = 0$. The followers cannot get into formation until the leader stops around 15 seconds in. There is also a small overshoot in the distance between the followers, meaning that they get too close to each other. In figure 5.1b the K_P value is increased to 0.7, while the other constants are kept at 0. With this alteration, the robots get into formation at the expense of the distance in between them overshooting more than before.

If the other PID constants, i.e. K_I and K_D are set to non zero values the overshooting can be controlled. In figure 5.2a the values are $K_P = 0.4$, $K_I = 1$ and $K_D = 0.0002$ which more or less removes the overshoot. In return, the robots no longer get into formation during the time the leader is moving. This is partly corrected by increasing the values to $K_P = 0.7$, $K_I = 1.3$, and $K_D = 0.0003$ which can be seen in figure 5.2b. The robots are still not quite able to get into formation while the leader is moving, and increasing the values more only resulted in unstable systems.

Since the general problem was the followers never catching up with the leader, the values of the PID constants for the PID controlling the distance between the leader and follower were increased. However, the values for the PID controlling the distance in between the



(a) $K_P = 0.4$, $K_I = 1$, $K_D = 0.0002$ **(b)** $K_P = 0.7$, $K_I = 1.3$, $K_D = 0.0003$

Figure 5.2: The distances in between the robots over time for the two different sets of PID values. The dotted black lines show the acceptable margin for being in formation.

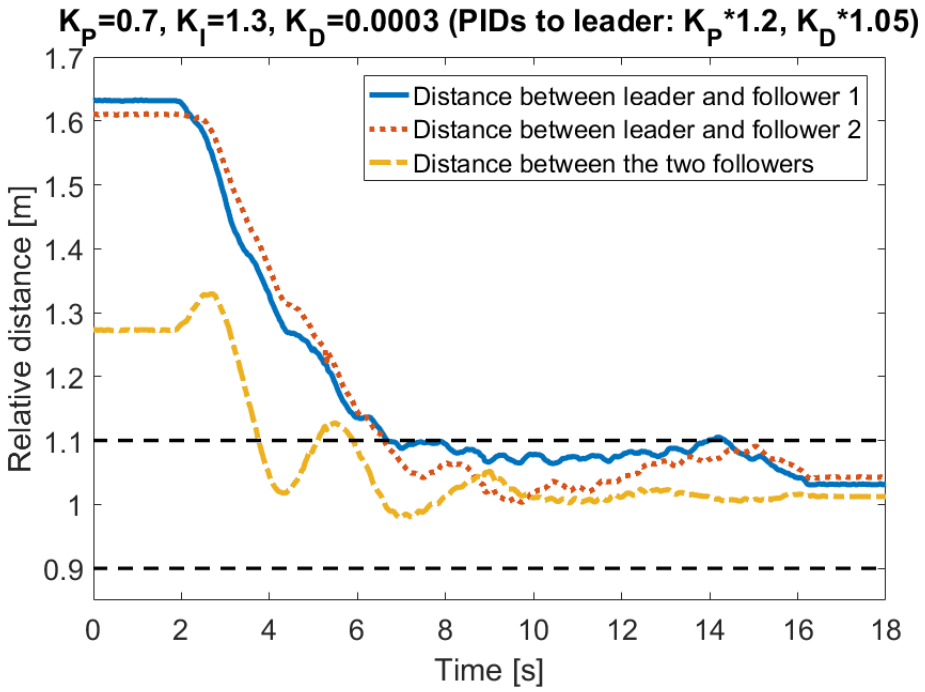
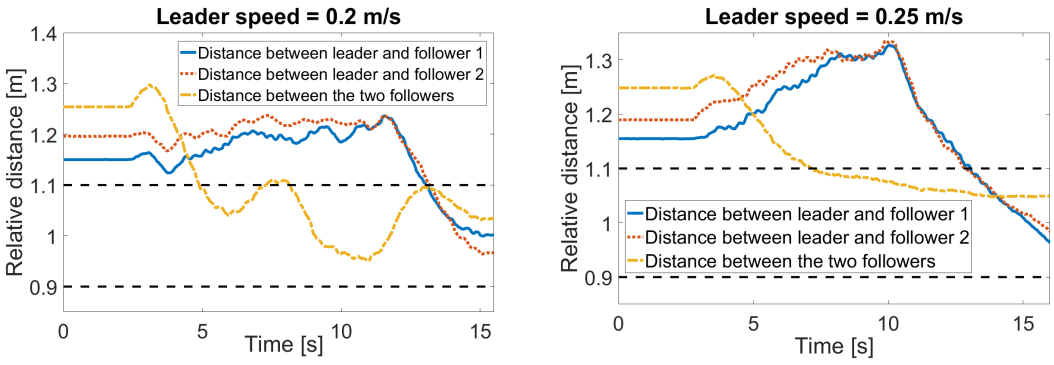


Figure 5.3: The distances in between the robots over time for the PID tuning process with $K_P = 0.7$, $K_I = 1.3$, and $K_D = 0.0003$. For the PID controlling the distance to the leader the K_P value was scaled by 1.2, and the K_D value with 1.05. The leader is moving with a speed of 0.10 m/s. The dotted black lines show the acceptable margin for being in formation.

followers were kept the same. Scaling K_P up for the PID controlling the distance between a leader and follower by 1.2 and K_D by 1.05 gave results according to figure 5.3, which shows a time for getting into formation of just under 5 s and an overshoot in the distance between the followers of about 2 cm. Those are the PID values used from now on.

5.2 Speed of the leader

In figure 5.4a, the relative distances between the robots when the leader is going by a speed of 0.20 m/s can be seen. If that is compared to figure 5.3, in which the leader is moving with 0.10 m/s, the speed's impact on the formation can be studied. When the leader is going with a higher speed, the followers settle around 20 cm from the leader instead of 10 cm when going with the lower speed. If the speed is additionally increased, to 0.25 m/s, the distance in between them becomes larger than before, up to 35 cm, which can be seen in figure 5.4b. The figures may imply that the correct formation is achieved. However, this is due to the fact that the leader halts after reaching the goal, allowing the followers to catch up. Increasing the speed of the leader to values higher than 0.25 m/s resulted in the followers not being able to find the final formation.



(a) Leader speed = 0.20 m/s.

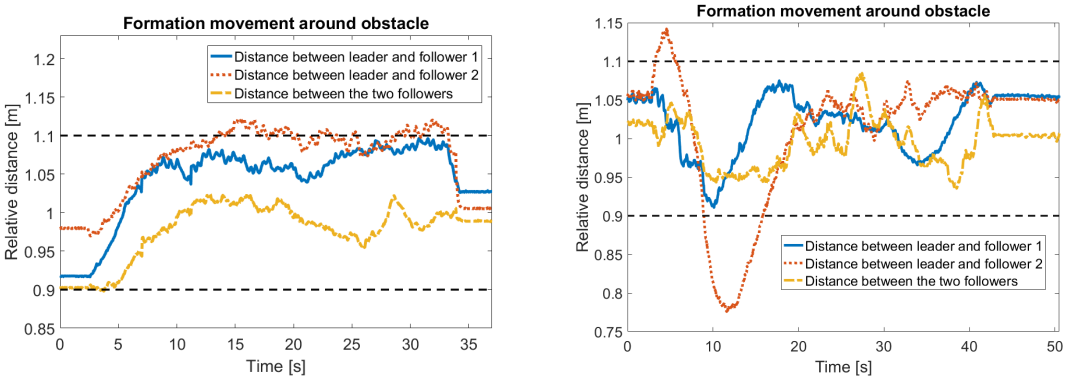
(b) Leader speed = 0.25 m/s.

Figure 5.4: The distances in between the robots over time when the leader is going with two different speeds. The dotted black lines show the acceptable margin for being in formation.

5.3 Formation movement

The main objective was to move three robots in a formation along the shortest path to a target position while avoiding obstacles. Using the final PID values from the tuning process in section 5.1 and a speed of the leader of 0.1 m/s, the distances in between the robots for the entire movement are plotted for two different tests in figures 5.5a and 5.5b. In figures 5.6a and 5.6b the corresponding movements of the robots are shown. The leader is represented by a star and is meant to follow the shortest path calculated by the A*-algorithm to the target, allowing the entire formation to avoid obstacles. The distances in between the robots are also shown. In general it is visible that the follower that travels outside of the leader's trajectory tends to have more trouble staying in formation than the one travelling on the inside.

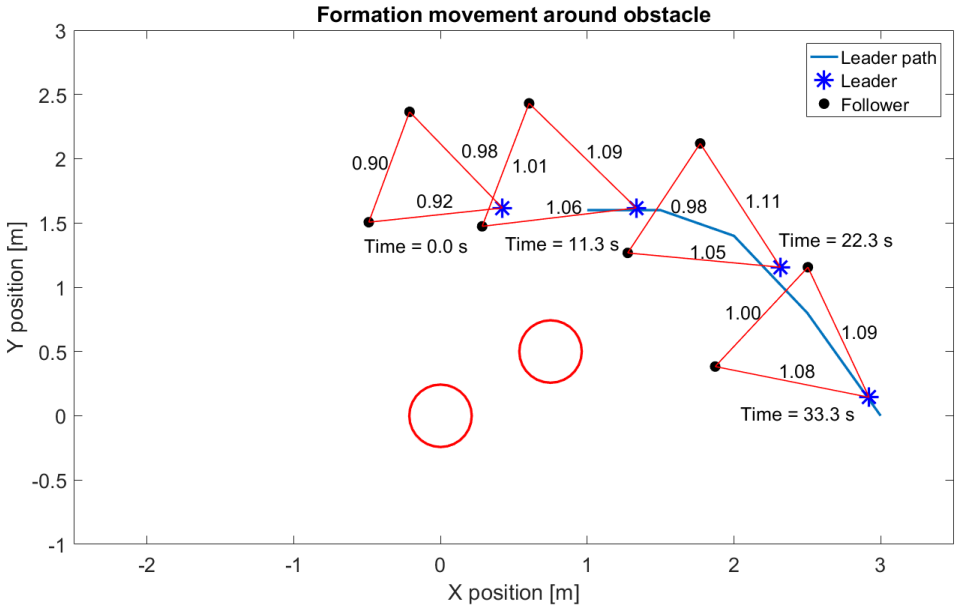
In the first test (fig. 5.5a) the percentage of the total time spent in formation within the accepted margins of ±10 cm is 64%, which is calculated from when the robots first



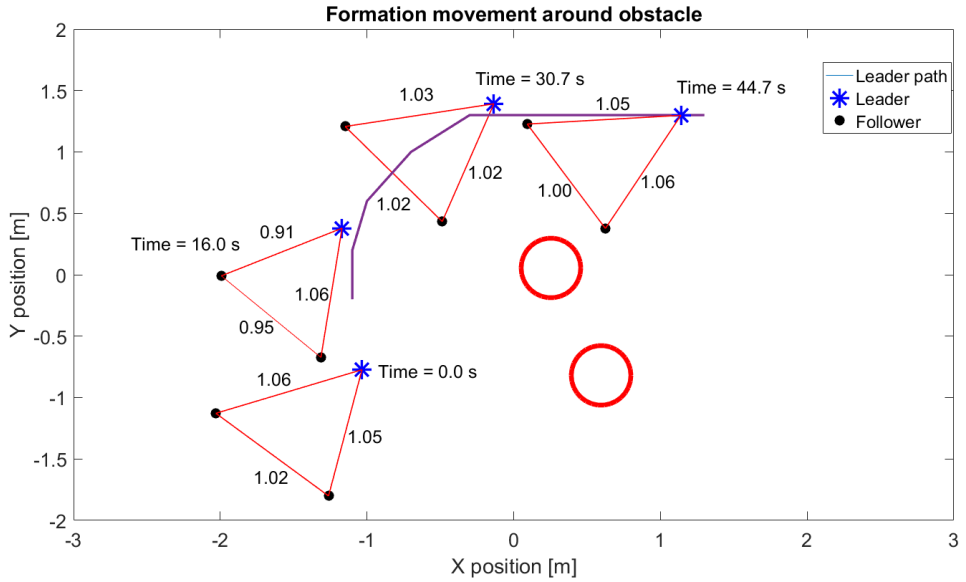
(a) The corresponding positions of the robots during the movement can be seen in figure 5.6a. The formation spends 64% of the time within this margin.

(b) The corresponding positions of the robots during the movement can be seen in figure 5.6b. The formation spends 77% of the time within this margin.

Figure 5.5: The distances in between the robots over time when the formation is travelling around an obstacle. The dotted black lines show the acceptable range for being in formation.



(a) Positions corresponding to the test with the relative distances in between the robots shown in figure 5.5a.



(b) Positions corresponding to the test with the relative distances in between the robots shown in figure 5.5b.

Figure 5.6: The positions of the robots in the formation at four discrete times for two different tests. The leader, represented by a star, moves along a path to a target point while avoiding obstacles represented by red circles. The time that has elapsed since the beginning of the measurement is shown next to the formation. The respective distances in between the followers, marked by a black dot, and the leader, are printed next to the line connecting them.

started moving until they were all in formation at the goal. It can be seen in the figure that the time being out of formation, is spent by only one follower a few centimetres from the acceptable margin. For the second test (fig. 5.5b) the percentage of the time spent in formation is 77%, but while being out of formation the distance error in between one of the followers and the leader is considerably larger than for the first test.

For both tests it can be seen that the formation is avoiding the obstacles as desired. Hence the area the camera sees translates into the desired graph representation, allowing the A*-algorithm to calculate the path for the leader.

5.4 Code

The final Python code for the control algorithms, and other programs used can be found in the public GitHub repository found at github.com/Stridma/SSYX02-17-82.git.

6

Discussion

In this section, we will discuss the results of our project in relation to the project aim and problem definition from sections 1.1 and 1.2. Additional matters of discussion are hardware and software tools used in this project, to which degree they had the required performance, as well as thoughts on further development of the project.

6.1 Hardware and software

The size of the robots and limitations on the camera system will be highlighted in this section.

6.1.1 Size of the robots

The robotic platform used in this project, explained briefly in section 3.1, had a characteristic that somewhat affected the project's outcome. This was the relationship between the size of the robots and the coverage area of the GulliView camera system. Driving three large robots on an area of about $2.8 \times 2\,\mathrm{m}^2$ meant that there were some limitations on the tests possible to perform. The only tests that were able to be executed, without setting up a larger test area, was either getting the robots to drive to a formation from scattered positions, or beginning in formation and driving 2 meters in a straight line. Smaller and more agile robots could have accelerated the development and supported more complex test scenarios, like driving around several corners.

6.1.2 GulliView as tracking system

GulliView as a tracking system presented challenges, with one of them being the code base that is lacking in its documentation. This meant that a lot of time and effort was spent on getting the system up and running, rather than core aspects of the project. Some modifications were made to the GulliView_server code[1], which was necessary for it to send out information on all three robots and obstacles.

[1]The code is found in the public GitHub repository at github.com/Stridma/SSYX02-17-82.git

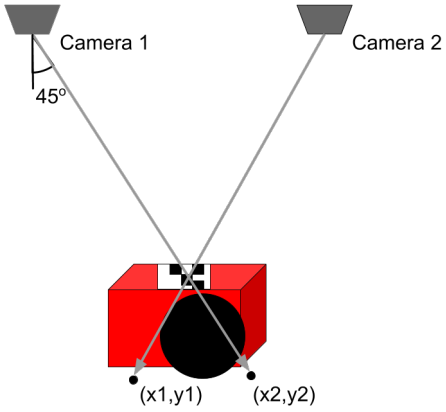


Figure 6.1: Illustration of tracking problem when driving from one field of view of one camera to another.

While GulliView has support for using multiple cameras in the system, the overlapping view between multiple cameras – while seeing the same four AprilTags used for setting up the room coordinate system – has some discrepancies. The main reason for this is that those four tags are taped to the floor and GulliView is tracking all other tags as if they were on the same plane. Errors then occur when two cameras track a robot tag (fig. 6.1) since the robot has its tags on top of it, about 30 cm above the floor. This difference in height causes GulliView to send out conflicting position data from the two cameras.

Due to the camera having a specified field of view of 90° the robot will have an angle of 45° from the camera when leaving the field of view. Knowing that the robot is 237 mm high, you can apply some basic trigonometry and calculate that the camera will position the robot about as far off as the robot tag is above the floor, since:

$$\tan 45^\circ = 1 = \frac{\text{height of robot}}{\text{position error}}.$$

Experiencing this during testing, the first solution tested to counter this error was to raise the four tags on the floor, to the same level as the robots' tags. While this fixed the majority of the issue, some positional error was still present when driving from one camera to another.

Another approach on avoiding this issue without using a filter or other changes to the software, is to instead use one camera but mount it higher over the floor plane allowing for a larger field of view.

6.2 Movement algorithm

Building upon the results listed in section 5, this section describes and analyses the process of testing the controller.

6.2.1 PID controller

Finding the optimal set of PID constants for the system was crucial for the project. First, the behaviour of the system was investigated in simulations, in order to give an idea of what to do with the real robots. Initial testing revealed that the usage of a leader-based system meant that the requirements on the two PID controllers used for each follower were a bit different. Since the leader was moving without consideration of the followers and the distance between them increased continuously, the desire for a follower to keep up with the leader had to be increased. The idea was that the integral term would accumulate the error of a follower trailing behind, but any further increments in the K_I value resulted in unstable systems. Instead, the PID controlling the distance between a follower and the leader was given larger values than the PID controlling the distance between the followers. This allowed for a somewhat better formation control algorithm, since the time to get into formation was reduced.

6.2.2 Follower direction

A substantial part of the efforts regarding the follower movement algorithm was deciding in which direction to go. The different ways of deciding the direction that were examined during the study are not described in the report, but only the final solution that was used. From the beginning, the follower used only one PID controller – first determining a goal based on its own and the other robots' positions, and the desired distance in between them. Then, controlling its movement to this point. This resulted in an unstable system since the determined goal was always changing.

The big breakthrough came when two PID controllers were implemented, allowing the follower to individually control its distance to each robot. The success was not immediate. In the first iteration with two PID controllers, the decision on which direction to go in was only based on the vectors from the follower to the leader and the other follower respectively, both scaled with their respective control output from the PID controllers. A plot of the distances between the robots for such a system can be seen in figure B.1 in the appendix. In the next iteration, the leader's orientation was considered, allowing the followers to align their movement with the leader when they were approaching being in formation.

The final step was realised by scaling the movement towards the leader with the product of the two control outputs from both PID controllers, allowing the follower to only move in the direction of the leader's orientation when the correct distance to the other follower was kept. This might give rise to some errors; in case the followers come into the right distance from each other but are far away from the leader they will not aim for the leader. This was never found to be a real problem, since the control outputs are never perfectly zero meaning the followers will always tend to go to the desired formation to some extent.

The improvements on the direction algorithm did not come without a cost – the system now had trouble when a follower was getting too close to the leader. Since the system behaved well in the majority of situations, no additional changes were made to the follower direction. Instead, it was decided to slow down the follower as soon as the distance between itself and the leader was smaller than desired. Since the robots used in this project are

relatively heavy, their inertia meant that going to a full stop and then speeding up again resulted in jerky movement – hence the decision to only slow down the follower. When falling behind the leader again the normal control commands could be used to stay in formation.

6.3 Future improvements

While the resulting controller fulfilled the initial goal, there is still room for improvements. Some of these are discussed in this section.

6.3.1 Alternative positioning systems

An obvious issue in using GulliView as the positioning system is that the tags on top of the robots cannot be covered, making it impossible to transport anything on top of them. If the system were to utilise other positioning equipment, which is not occupying the top of the robots, the transportation of objects would be possible. An example of these positioning systems is LIDAR, which measures distances within the formation by pulsing laser light. A global coordinate system would still be needed for the leader, in order to plot a path to the target. Using LIDAR could also allow the system to avoid collisions, as obstacles without tags would also be detected.

One major advantage of using GulliView is that it gives very accurate position data with a fast update frequency as explained in section 3.3. If another positioning system is evaluated as a replacement, you would have to compare its performance with GulliView. Using a Kalman filter to handle unreliable data was first planned for, but later omitted due to the reliability and accuracy of the GulliView system.

6.3.2 Dynamic leader reassignment

One issue with the current system is the inability to do sharp turns. Manoeuvres such as u-turns, seen in figure B.2 in the appendix, risk putting the system in a state similar to that of an inverse pendulum, compromising the stability of the system. One way to deal with this is to implement the algorithm that reassigns the leader-status from one robot to another (sec. 4.2.1) dynamically, in contrast to assigning the leader in the beginning of the movement. This allows the system to adopt traits from a swarm-based system, such as decreasing the importance of one single robot. For example, if the leader robot would be incapacitated, or moved by external forces, another could quickly take its place, allowing the system to keep active.

7

Conclusion

During the project, a controller was designed and implemented that was successful in moving a formation of three robots from their point of origin to a goal point via an optimal path. Throughout different iterations of the robots' movement along the path, they were able to admissibly maintain their formation – meaning that the formation control can be assumed to have succeeded, and thus that the aim has been achieved.

While it was not shown that the developed system is a viable way of transporting large objects, it holds great promise in the area. A controller able to manage and maintain a moving formation of robots has many functions and possible expansions, especially given the manageable nature of a central controller for a multi-robot system. The project paves the way for further research, as it provides an efficient and reliable way of maintaining a formation of robots throughout movement.

Should the principles of the method used in the project be applied to a system with an even greater number of robots, large scale transport via dynamic autonomous couriers may be implemented. Not only that, but potentially also leading to reliable platooning of autonomous vehicles. Formation control is, by all means, a field of vast applications.

Bibliography

[1] M. Boden. (2006) Grey walter's anticipatory tortoises. [Online]. Available: http://www.rutherfordjournal.org/article020101.html

[2] (2017) The world's first fully automated uav for cleaning solar panels. Aerial Power Ltd. [Online]. Available: https://www.aerialpower.com/solarbrush/

[3] (2017) Forcasts. [Online]. Available: http://www.driverless-future.com/?page_id=384

[4] T.Seba, "Clean distribution: Why conventional energy and transportation will be obsolete by 2030," in *Presentation to: Swedbank Nordic Energy Summit.* 2016 ©Tony Seba, 2016.

[5] E. Kim. (2016) Kiva robots saves money for amazon. Bussines Insider. [Online]. Available: http://www.businessinsider.com/kiva-robots-save-money-for-amazon-2016-6?r=US&IR=T&IR=T

[6] K.-K. Oh, M.-C. Park, and H.-S. Ahn, "A survey of multi-agent formation-control," Survey Paper, Korea Institute of Industrial Technology, 2014.

[7] V. Gazi and K. Passino, "Stability analysis of swarms," *IEEE Transactions on Automatic Control*, vol. 48, 2003.

[8] T. Glad and L. Ljung, *Reglerteknik: Grundläggande Teori.* Studentlitteratur AB, 2006.

[9] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms.* The MIT Press, 2009.

[10] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, pp. 100–107, July 1968.

[11] O. Beronius, E. Lundén, M. Malmquist, and A. Rohlin, "Coordination of robots via a wireless network," Bachelor's thesis, Chalmers University of Technology, 2016.

[12] Pioneer p3-dx. [Online]. Available: http://www.mobilerobots.com/ResearchRobots/PioneerP3DX.aspx

[13] (2017) About ros. [Online]. Available: http://www.ros.org/about-ros/

[14] (2017-03-01) Rosaria. [Online]. Available: http://wiki.ros.org/ROSARIA

[15] E. Olson. (2016) Gulliview. [Online]. Available: https://bitbucket.org/thpe/visionlocalization

[16] (2016) Apriltag. April Lab, University of Michigan. [Online]. Available: https://april.eecs.umich.edu/software/apriltag/

[17] A. Arkheden, R. Zaragatzky, A. Lindhé, and R. Gustafsson, "Ett prisvärt alternativ för global visuell lokalisering och styrning av autonoma fordon," Bachelor's thesis, Chalmers University of Technology, 2016.

[18] C. Careaga. (2014) Python a* path finding (with binary heap) (python recipe). [Online]. Available: http://code.activestate.com/recipes/578919-python-a-pathfinding-with-binary-heap/

A

Assignment of nodes

Depending on the initial placement of an arbitrary number of robots on an $n \times m$ sized map, the assignment of a robot to a certain node in the desired formation on a decided location on the map could be costly. Minimizing the distance travelled, as well as other potential cost variables, for the overall system saves energy and resources.

Preserving the resources as an aim would lay the foundation for a decision reached through consensus, since simply driving the robots to the point closest to themselves could lead to more than one robot going to the same node, thus leaving other nodes in the formation unmanned. The assignment of robots to nodes can be seen as a linear-programming problem as follows:

Let us denote by the matrix M, each robot's distance to each node in the formation. In this example, there are three robots and three nodes:

$$M = \begin{bmatrix} D_{00} & D_{01} & D_{02} \\ D_{10} & D_{11} & D_{12} \\ D_{20} & D_{21} & D_{22} \end{bmatrix},$$

where the element D_{ij} is the i^{th} robot's distance to the j^{th} node in the formation. For each element D, there is a corresponding binary variable X that represents whether or not that path is taken by the robot in question. Only one X for each row can be set to 1, meaning the robot can only go to one node. In a similar fashion, only one X for each column can be set to 1, meaning that no two robots can go to the same node.

D is a constant in the minimising function, and X a variable. By defining a set of constraints as described above, we have the following linear-programming problem:

Minimize f(x) subject to:

$$X_{ij} \in \{0, 1\}$$

$$\begin{aligned} X_{00} + X_{01} + X_{02} &= 1 \\ X_{10} + X_{11} + X_{12} &= 1 \\ X_{20} + X_{21} + X_{22} &= 1 \end{aligned}$$

$$\begin{aligned} X_{00} + X_{10} + X_{20} &= 1 \\ X_{01} + X_{11} + X_{21} &= 1 \\ X_{02} + X_{12} + X_{22} &= 1 \end{aligned}$$

where

$$f(x) = X_{00}D_{00} + X_{01}D_{01} + .. + X_{22}D_{22}$$

The final result will be each X having either the value 0 or 1, such that there is only one 1 in each row and only one 1 in each column. This means that each robot-node pair is exclusive, and if X for the i^{th} row in the j^{th} column is equal to 1, the i^{th} robot goes to the j^{th} node.

This could be applied to n number of nodes and robots by setting the matrix M to an $n \times m$ matrix.

$$M = \begin{bmatrix} D_{00} & .. & D_{0m} \\ . & . & . \\ D_{n0} & .. & D_{nm} \end{bmatrix}$$

B

Additional results

B.1 Discarding leader alignment

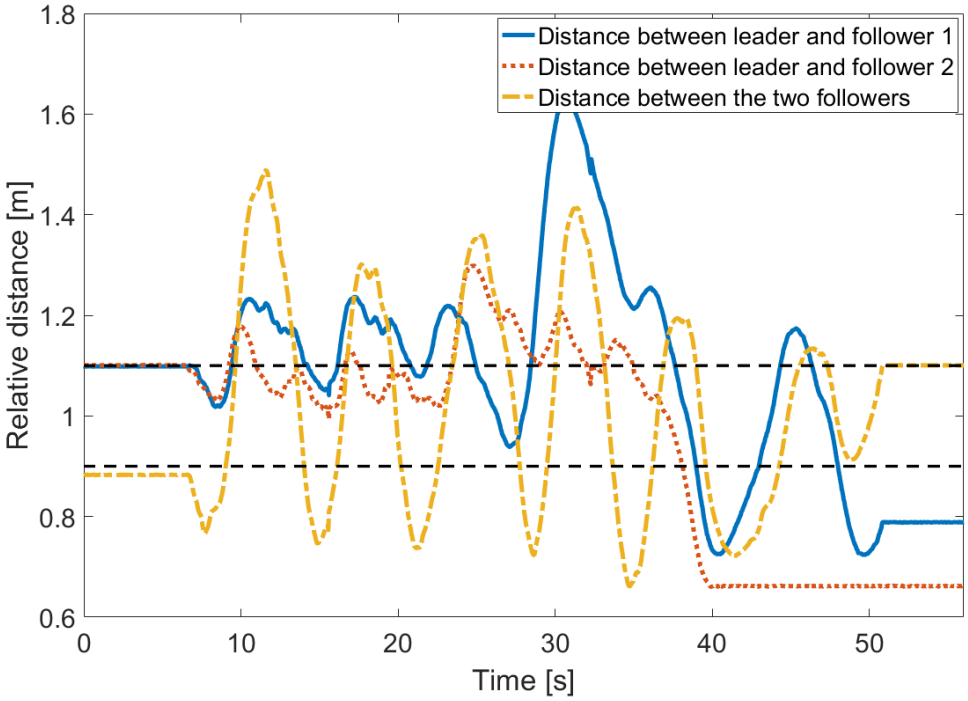


Figure B.1: When the followers are not aligned with the leader when they get into formation, the system remains unstable.

B.2 Sharp turns

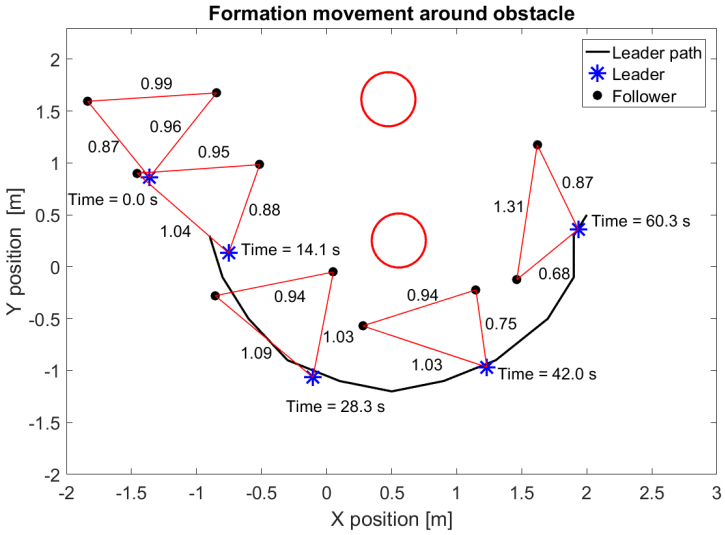


Figure B.2: Sharp turns like this one pushes the system beyond what it can handle. The distances in between the robots during the entire movement is seen in figure B.3.

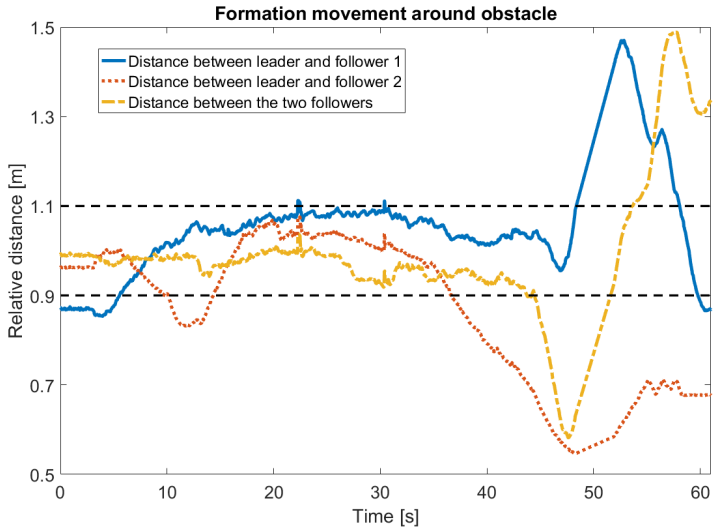


Figure B.3: During the U-turn seen in figure B.2, the system is only in formation 47% of the time until the robots are stopped manually. The formation is doing fine until around 35 seconds in, when the leader is turned towards follower 2.

C

Algorithms

Djikstra's algorithm is a single-source shortest-path algorithm. Supposing that there is a weighted, directed graph with exclusively positive weights, the algorithm finds the shortest path from the starting node to every other node in the graph. It achieves this through keeping a set of nodes for which the final shortest-path weight has already been determined, and repeatedly selecting nodes in the graph that are not in said set based on the minimum shortest-path estimate until there are no remaining [9, pp. 658-659]. While being efficient when wanting to know the shortest path from one node to all other nodes in a graph, when interest only lies in finding the shortest path to a certain node, the broadness of Djikstra's algorithm becomes redundant.

A greedy algorithm is an algorithm that, rather than dynamically determining which choice is better, bases its decision on what seems best at that moment – meaning it will solve the overall optimal task, and not dynamically solve subtasks [9, pp. 414-418]. While this may be highly useful in some cases, in the case of wanting to find the shortest path, it will seek to follow the route that is shortest independent of the obstacle. Largely depending on the characteristics of the obstacle, whether or not a greedy algorithm returns the actual shortest path may vary, making it unreliable in the case of wanting to find the shortest path. That is to say, the algorithm may expand an unnecessarily high number of nodes, since it will not begin avoiding the obstacle until it encounters it.