Analysing the Global Convergence Capabilities of a Bearing-Based Formation Control Law using ROS

Bachelor Thesis

Nils Meile - S3511863

Supervisors: prof. dr. Bayu Jayawardhan & Mehran Mohebbi

Industrial Engineering and Management
Faculty of Science and Engineering
University of Groningen

June 12, 2020

List of Abbreviations

UAV	Unmanned Aerial Vehicle
UUV	Unmanned Underwater Vehicle
DTPA	Discrete Technology Production Automation
LIDAR	Light Detection and Ranging
ROS	Robotic Operating System
LTS	Long Term Support

Abstract

Formation control offers the possibility to operate without the necessity of a centralized system making it appealing to a myriad of applicable fields. This paper contributes to the current literature by investigating the global convergence capabilities of a bearing-based formation control law using the simulation software ROS. Four agents tried to achieve a rectangular shape in a series of 81 distinct simulations that delivered rigours data about the control law's performance. The results yielded convergence capabilities from any displacement from the desired formation shape although failure rates were high due to a poor data processing performance. Finally, common errors and the error causes were discussed and solutions for future research were suggested.

Contents

1 Introduction **1**

1.1 Problem analysis . . . 2

1.1.1 System description . . . 2

1.1.2 Problem context . . . 3

1.1.3 Problem statement . . . 3

1.1.4 Research objective . . . 3

1.1.5 Research questions . . . 3

2 Literature review **5**

2.1 Preliminaries . . . 5

2.1.1 Graph theory . . . 5

2.1.2 Rigidity theory . . . 5

2.1.3 Local versus global convergence . . . 5

2.2 The control law . . . 5

2.2.1 Notation . . . 6

2.2.2 Angle-constrained formation control problem . . . 6

3 Setup **8**

3.1 Equipment . . . 8

3.1.1 Nexus robots . . . 8

3.1.2 Laser sensor . . . 8

3.2 Software . . . 10

3.3 Data processing node . . . 10

3.4 Controller . . . 11

3.5 Simulation . . . 11

4 Results **14**

4.1 Notation of the result summary table . . . 14

4.2 Formation a=3, b=4 . . . 15

4.3 Formation a=5, b=5 . . . 17

4.4 Formation a=2.5, b=5 . . . 18

4.5 Performance of each zone . . . 20

4.6 Performance of different gains . . . 22

5 Discussion **24**

5.1 Error of 1.7m . . . 24

5.2 Data processing problem . . . 24

5.3 Research questions answered by the thesis . . . 25

6 Future Research **27**

7 Conclusion **28**

1 Introduction

Formation control of a multi-agent system has been a highly researched topic during the past years. It examines the problem of controlling a group of robots in order to arrange them to a specified geometrical shape (Chan et al. 2020). The group is placed randomly in space and needs to allocate its desired location from a suite of on-board sensors and relative position measurements, instead of using the help of a centralized system (de Queiroz et al. 2019). The application of controlling the formation of a multi-agent system ranges from transportation to surveillance and search operations, thus being an extremely relevant research topic (Guo et al. 2010). Previous research has shown that the relative information that an agent requires to achieve a desired formation is based on: (1) position, (2) displacement, (3) distance or (4) bearing measurements (Cao et al. 2019, Ahn 2019, Li et al. 2018). Bearing only measurements have an advantage for situations where exchanging signals is prohibited or the payload of a robot needs to be reduced, so that heavy sensors are undesirable (Trinh et al. 2019, Ahn 2019). These characteristics are of great importance in the industry, for example in the production of unmanned aerial vehicles (UAVs) and unmanned underwater vehicles (UUVs) (Li & Xie 2018).

Extensive research has already been carried out in the field of bearing-based formation control. The research areas focus on a variety of dynamics, namely, single-integrator dynamics, double integrator dynamics, formation control using the absolute position, formation control using the relative position, formation control using a directed network (Li & Xie 2018) and Chan et al. (2020) found a control law guaranteeing *collision avoidance*. However, the control law constructed by Chan et al. (2020) has been proven only theoretically.

However, the performance of the control law is still unexplored, meaning simulations on the system's capabilities need to be performed. Therefore, this research project will contribute to the existing research by analyzing the limitations and merits of the given control law, which is based on the formation control using only bearing measurements.

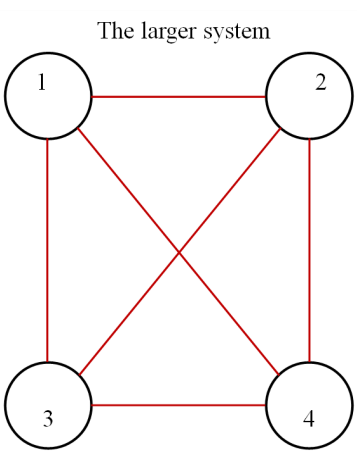
Firstly, the problem analysis is given, which will give a system description that scopes the boundaries of the project. The final problem is elaborated on and stated in the problem analysis followed by the research questions. Secondly, the preliminary literature is studied and the simulation setup is described. Then, the results will give a detailed overview of the outcome of the simulations. Finally, a conclusion is drawn.

1.1 Problem analysis

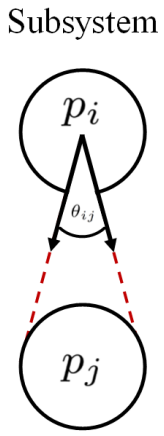
1.1.1 System description

Analyzing the system in terms of a larger system and a subsystem will prove helpful for understanding the critical components involved. Firstly, the larger system is composed of multiple mobile robots that converge to a desired geometrical shape. Each mobile robot is able to allocate their adjacent neighbours by a rotating laser equipped on top of their structure. Therefore, in an ideal case the mobile robots are aware of their neighbours position but do not have access to the data the other robots are sensing because the system is decentralized. Figure 1a illustrates the identification between all four mobile robots in space.

The subsystem serves to visualize the sensing setup of each mobile robot. For that, Figure 1b shows two robots, where robot p_i is the observer and robot p_j is being observed (Guo et al. 2010). Each robot is assumed to be circular with a radius of $r = 1$. The bearing measurement is taken from the outermost corners of the adjacent neighbours and the allocated corners are then used to find the internal angle θ_{ij} (Chan et al. 2020). Using geometrical arguments it is also possible to find the inter center distance (d_{ij}).



(a) The simulation setup with four mobile robots. Edges between each agent are added to illustrate that the robots are able to sense each other (Chan et al. 2020)



(b) Sensing setup with robot 'i' being the observer and robot 'j' being the observed robot (Chan et al. 2020)

Figure 1: Illustration of (a) the whole system and (b) the subsystem

1.1.2 Problem context

Bearing based formation control has been studied extensively considering various dynamics of a multi-agent network. The DTPA laboratory has researched the field of formation control and Chan et al. (2020) has designed a control law providing local exponential convergence for circular mobile robots using relative bearing measurements only. The control law was solved purely theoretically, thus the system has not been tested using a 3D simulation engine. More specifically, the influence of sensors and the data transmission between the robots was not simulated yet.

1.1.3 Problem statement

A research group from the DTPA laboratory has developed a control law for bearing based formation control of a multi-agent system. However, the control law was only proven on its local exponential convergence theoretically and has not been tested on its local and global convergence in a simulation. Therefore, the expected performance, meaning the systems limitations and merits, are unknown.

1.1.4 Research objective

The research objective is to analyze the performance of the given bearing-based formation control law, and explore the control law's limitations within three months by conducting computer simulations. The results will be relevant as the acquired data will contribute to the findings made by the DTPA laboratory and thus supports current literature on the formation control of a sensor reduced multi-agent system

1.1.5 Research questions

The questions for the research project are reduced to one central question. The central question has been formulated in a way, such that the sub questions answer the central question and therefore the stated research objective can be achieved.

Central question

1. What is the performance of the control law for bearing-based formation control with respect to global convergence using computer simulation?

Sub questions

1. What is the convergence behavior of the control law?
2. What are the regions of attraction?
3. What are the control law's limitations in a simulation?

4. What are the reasons for a simulation failure?

5. What is the probability that the system will converge?

2 Literature review

2.1 Preliminaries

The following section intends to elaborate on the preliminary knowledge that was applied in the paper of Chan et al. (2020). As the research paper is based on the preliminaries, it is necessary to understand the essential information to follow through the approach of the control law.

2.1.1 Graph theory

An undirected graph $\mathcal{G}$ is a pair $(\mathcal{V}, \mathcal{E})$ where $\mathcal{V}$ is the set of *vertices* $\mathcal{V} := \{1, 2, ..., n\}$ and $\mathcal{E} := \{\{i, j\} | \, i, j \in \mathcal{V}\}$ is the finite set of vertices. The set of unordered pairs $\{i, j\}$ is called *edges* (Chan et al. 2020). An edge $\{i, j\}$ indicates that i and j are neighbours to each other, and the sets of neighbour vertices of i are denoted as $\mathcal{N}_i := \{j \in \mathcal{V} | \, \{i, j\} \in \mathcal{E}\}$. Furthermore, the graph does not have an edge from vertex i to the same vertex i (Chan et al. 2020).

2.1.2 Rigidity theory

To guarantee that the desired formation shape is realizable, the 'Rigidity theory' is applied. The theory studies the attainability of formations given predetermined constraints between the agents, for example the distance or the inner angle between each agent (Ahn 2019). More generally, rigidity theory describes the phenomenon that it is technically not feasible to move a node of the network without deforming the geometrical formation (Sidman & John 2017).

2.1.3 Local versus global convergence

Chan et al. (2020) demonstrated theoretically that the constructed control law has local exponential convergence, meaning that the agents can converge to the desired formation shape if an agent's individual solution point is located nearby the agent's initial position. Consequently, the control law is said to be globally convergent if the agents converge to their solution point regardless of their initial position on the global coordinate frame.

2.2 The control law

This subsection will briefly mention and explain the crucial expressions and formulas of the given control law. First, the notations are given followed by the actual control law describing the angle-constraint control problem.

2.2.1 Notation

The notation $x^\top$ is the transpose of the vector x and $||x|| = \sqrt{x}$ describes the 2-norm of x (Chan et al. 2020). The notation x^* is used to visualize the desired value of x and hold constant during the simulation as the desired formation shape will be predetermined. Lastly, x_L, x_+ and x_R are used to describe the left, center and right side of the circular robot respectively.

2.2.2 Angle-constrained formation control problem

The multi-agent system is assumed to move with single integrator dynamics,

$$\dot{p}_i(t) = u_i(t),$$

where u_i and p_i are stacked vectors and u_i is the input velocity for the control of the nexus robots inside the ROS environment. In order to derive the control law, a group of n robots, with $\mathcal{V} = \{1, 2, ..., n\}$ being the index set, is assumed.

The sensory system is located in the center of the robot p_i and the laser is able to locate the two outermost corners of its adjacent neighbours denoted as p_{jL_i} and p_{jR_i}. The outer corners are measured in form of *relative bearing measurements* and are given by $g_{ijL} = \frac{z_{ijL}}{||z_{ijL}||}$ and $g_{ijR} = \frac{z_{ijR}}{||z_{ijR}||}$. The internal angle is then obtained using geometry rules:

$$\cos(\theta_{ij}) = 1 - 2\left(\frac{r}{d_{ij}}\right)^2. \tag{1}$$

During the simulation the system attempts to acquire the desired relative angles by reducing the system's error, which is described by the difference of the actual relative angle measurement to the desired relative angle measurement: $e_{ij} = \cos(\theta_{ij}) - \cos(\theta^*_{ij})$.

In order to guarantee collision avoidance at any given time moment, the robots are not allowed to touch each other during the simulation, meaning the inner-center distance needs to remain larger than both of the agent's radii added together: $||p_j(t) - p_i(t)|| > 2r$, $\forall t \geq 0$. Additionally, the objective is to achieve the desired angle when time goes to infinity. Therefore, convergence is guaranteed if $\theta_{ij}(t) \to \theta^*_{ij}$ as $t \to \infty$ or $e_{ij}(t) \to 0$ as $t \to \infty$.

Given the requirements stated above, Chan et al. (2020) proposed the following angle-based potential function,

$$V_{ij}(e_{ij}) = \frac{1}{2}r\left(\frac{\cos(\theta_{ij}) - \cos(\theta^*_{ij})}{\cos(\theta_{ij}) - \frac{1}{2}}\right)^2 = \frac{1}{2}r\left(\frac{e_{ij}}{e_{ij} + c_{ij}}\right)^2, \tag{2}$$

where $c_{ij} = \cos(\theta^*_{ij}) - \frac{1}{2}$. The first derivative yields $v_{ij}(e_{ij}) := \frac{\partial}{\partial e_{ij}} V_{ij}(e_{ij}) = r\frac{e_{ij}c_{ij}}{(e_{ij}+c_{ij})^3}$.

Furthermore, the control input that will be used during the simulation is given by:

$$u_{ij}^{\top} = -2\hat{v}_{ij}(e_{ij}(1-\cos\theta_{ij})\sin\theta_{ij}\frac{g_{ij+}^{\top}}{||g_{ij+}||^2}, \tag{3}$$

where,

- $\hat{v}_{ij}(e_{ij}) = \frac{v_{ij}(e_{ij})}{r}$
- $\frac{g_{ij+}^{\top}}{||g_{ij+}||^2} = \frac{2\frac{s}{a}z_{ij}}{4\left(\frac{s}{a}\right)^2 d_{ij}^2}$
- $\frac{s}{a} = \frac{1}{2r}sin\theta_{ij}$

The given formulas were enough to formulate a code for the controller that is able to acquire a velocity input navigating the agents into the desired direction. The lines 205 to 231 of the Python code in Listing 2 show how the control law was realized in code.

3 Setup

This section will give information on the equipment and code used to execute the simulation. Additionally, the structure, that was used for the simulations so that rigorous data can be obtained, will be explained.

3.1 Equipment

3.1.1 Nexus robots

The mobile robots used in the simulation are manufactured in reality by the company 'Nexus robotics' (see Figure 2) and they can be controlled by an external program running on the programming language "Python". Each robot is equipped with a battery, a computer, a DC motor, a rotating laser and a low quality camera. However, since the research is based on bearing measurements, the rotating laser on the top center of the robot will be the only activated sensor during the simulation. The simulation will also disregard the rotation of the robot, thus it is assumed the robot can move into any two dimensional direction without rotating around its own axis. Lastly, the minimum speed of the robot is limited to 0.002 $\frac{m}{s}$ and the maximum speed is limited to 0.2 $\frac{m}{s}$. This restriction was necessary to optimize the tracking of other robots during the simulation. Otherwise, the rapid movement would decrease the measurement quality and produce unreliable results.

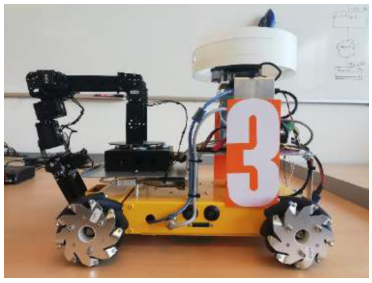


Figure 2: Picture of one of the Nexus robots used

3.1.2 Laser sensor

The laser scanner mounted on the top center of each robot is a RPlidar scanner. Understanding the functioning of the RPlidar scanner is crucial in order to understand the reaction of the system to certain measurement inputs. Thus, this section will give a brief overview of the way the RPlidar scanner operates.

The RPlidar scanner is a 360° rotating laser sensor produced by the company 'Slamtec' (Slamtec 2018). At the start of each scan, the sensor will begin by shooting an infrared laser ray into its starting direction followed by new laser ray shots, clockwise around its midpoint, till a whole revolution is completed (Rasshofer & Gresser 2005). This process will be repeated until the simulation is stopped. Some laser rays will collide with an object and then bounce back so that they are sensed by a photo sensor equipped to the laser. The time the laser ray takes to return to its origin is called the time-of-flight and is used to determine the distance between the laser scanner and the object (Shan & Toth 2018).

The RPlidar sensor stores the information inside an array and publishes the array to the intended subscriber.

It is important to recognize that each of the array's entries represent one laser ray and that the array is ordered in a way that the first laser ray will be the first entry of the array. If a laser ray does not collide with an object, the array will show "Infinity". For every object that is recognized, the sensor will give the distance to the object in meters. Figure 3 shows a schematic drawing of the process.

Depending on the user demands, various sensor settings can be adjusted. For example, it is possible for the user to vary the field of view from 0° - 360° to for example 0° - 180°, which will increase the computational capacity as a decreased number of rays will reduce the distance calculations of the laser's computer. However, the simulation performed for this research will require a 360° view thus the robots will be identifiable from any position in the simulation environment.

The standard setting also includes a resolution with the unit 'ray per degree', which is equal to 1. This proved disadvantageous during the simulation as a higher density of rays was needed for the agents, located far from each other, to remain visible for the sensor (see Figure 3). Accordingly, the resolution was increased to 10 $\frac{rays}{degree}$.

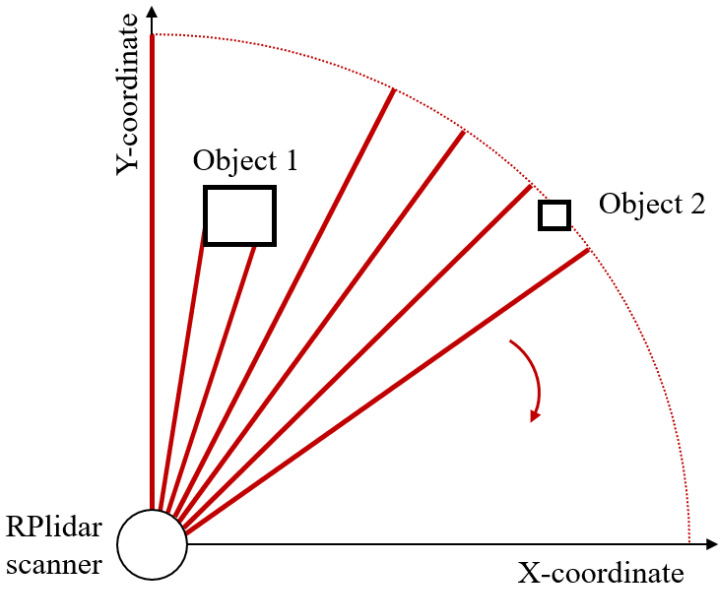


Figure 3: Illustration of the functioning of the RPlidar sensor. Recognize that object 1 is identified and object 2 is not.

3.2 Software

The software program chosen to execute the simulation is ROS (Robotic Operating System). The software runs on the operating system Ubuntu 18.04 LTS (Bionic Beaver) and is updated to its currently most recent version called "ROS Melodic". Additionally, the software is open source and offers message passing between processes based on a subscriber-publisher principle. A subscriber is a node that 'subscribes' or acquires data from another node and a publisher is a node that 'publishes' or sends data to another node. This methodology allows the existence of decentralized controllers inside ROS because each robot will be able to perform their own calculations.

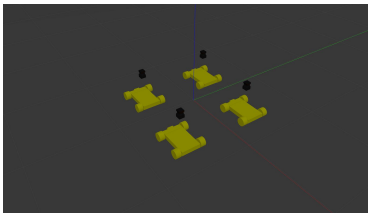


Figure 4: Picture of four Nexus robots spawned inside of Gazebo

At the beginning of each simulation, four nexus robots are spawned on individual coordinates in the physics engine tool "Gazebo". Then, two algorithms (Data processing node, Controller), written in the programming language "Python", are executed repeatedly. The functioning of each code will be elaborated in sections later on. However, as each mobile robot does its individual computations on board, a sum of eight codes needs to be running simultaneously (see Figure 4).

Once everything is running correctly, each agent will use the velocity input of the controller to calculate the direction and speed the mobile robot needs to maneuver in order to acquire the desired geometrical shape. The measurements that have been collected by each robot during the simulation (Input velocity of each robot and Inter-agent distance error) are then formatted into a diagram and published to the user's desktop.

3.3 Data processing node

The data processing node's function is to structure the information received from the scanner and publish only what is needed to the controller. The process of structuring the data will be elaborated in the following section.

Firstly, the data processing node subscribes to the published data of the individual robot measured by the laser. The data is then analyzed for clusters or more specifically groups of values that show numbers and do not show infinite values. As the robots are the only objects that have collision enabled inside of the simulation environment, it can be concluded, that each cluster is representative for a robot. Therefore, the number of adjacent neighbours is determined by counting the number of clusters.

Secondly, the infinite values inside of the array are filtered out by the data processing node and a separate array for each data cluster is constructed. If everything worked as intended, there should be three separated and ordered arrays. Next, the distance to each robot is calculated by finding the average value of each cluster. The relative angle is determined by counting the number of rays that have been shot out since the starting position of the laser scan. Knowing that the resolution is 10 $\frac{rays}{degree}$ will result in the relative angle towards each robot.

Lastly, the relative distance d_{ij} and position vector z_{ij} of each robot are summarized into a vector called "Z-values", which is published to the respective controller. The code for the data processing node is represented in the Appendix Listing 1.

3.4 Controller

The controller's code used during this research is based on a Matlab code formulated by Nelson P.K. Chan which is transferred to a Python code and implemented into the ROS environment by the researcher. The objective of the controller is to use the Z-value vector, published by the data processing node, to calculate a control input that will steer the robot into its desired position. The way the controller operates is explained in the following section.

Before the Z-value vector can be used to calculate the velocity input for the nexus robots, the desired angle towards the robots needs to be determined. For convenience, the desired angle is calculated inside the code and not entered manually. This procedure has the advantage that the user has to define only the height (a) and width (b) of the desired formation shape before the simulation, and the code will compute the remaining values needed for the control law.

Thereafter, the respective data for each nexus robot is attained from the Z-value vector. The data is then used to compute the control input which is published to nexus robot. The code for the controller is represented in the Appendix Listing 2.

3.5 Simulation

Before a series of simulations is conducted, a structured plan is made that will guarantee comprehensive data acquisition. It is ensured that every simulation will test distinctive system properties thus collecting rigorous data that will pinpoint the control law's limitations.

Overall, three formation shapes will be tested. The first formation will have a height of 4m and a width of 3m. This specific formation shape was also tested by Chan et al. (2020), thus a successful simulation will demonstrate the general functionality of the control law. The second formation will be a square of 5m. This formation intends to test the system's behavior if Nexus 1 and Nexus 4 have two identical inner-angles towards the robots Nexus 2 and Nexus 3. The last formation will be of height 2.5m and width of 5m. This scenario will test the collision avoidance feature of the control law as the robots (Nexus 1 Nexus 2 + Nexus 3 Nexus 4) will need to maneuver closely to the minimal inner distance of $2r$ without crashing into each other.

Secondly, each of the three formation shapes will be explored regarding their regions of attraction. Therefore, each robot will be positioned at an increasing distance from their initial point of solution. All together, three zones are created around each of the four points of convergence. The first zone will test the local convergence of the formation shape and covers an radius of 1m from the initial point of convergence. The second zone reaches from 1m to 2m from the initial point of convergence and the third zone can be any where from 2m and higher. The intention is to create an increasingly challenging situation in order to identify reasons of failure.

Thirdly, within each zone the robots will be placed at three different positions. The first position tested should be the most basic and easiest placement to converge from, so that it can be proven that the displacement from the solution point is not the reason for failure. Consequently, the first position will be a scaled version of the desired formation shape. The second and third position will be chosen randomly within each zone. However, it should be recognized, that the second positioning is an 'easier' shape to converge from than the third positioning. Meaning, the third positioning tries to test limitations by:

- placing robots in between the sight of view of other robots
- switching the position of two robots to see if they can be identified correctly by their adjacent neighbours
- creating an initial formation shape that is contrasting to the desired formation shape

Finally, each positioning will be tested three times. Nonetheless, the gain of each controller will be increased after each attempt. The gain was chosen from the following values: 10, 20, 50, 100, 200, 500. Higher gain values are used when the robots are placed further away from each other. This decision was made to increase the converging time significantly as the inner angle decreases considerably if long-distance measurements are taken. Consequently, the control input is multiplied with a value much smaller than 1, resulting in an extremely low velocity input that sometimes is lower than 0.002 $\frac{m}{s}$, which

stops the robot's movement (see Section 3.1.1).

To summarize, a total of $3 \cdot 3 \cdot 3 \cdot 3 = 81$ simulations will be conducted. The results of each simulation will be captured on a summary table and saved in diagrams. Figure 17, 18 and 19 show a representation of the spawning positions of each robot on a coordinate system. Each zone is marked by a ticked red circle that is placed around each of the four red solution points. The dots coloured in green, grey and blue represent the positions where each robot was spawned. The green dot is the scaled version of the desired formation shape. The grey and blue dots are randomly placed in each zone, whereby convergence from the blue dots is more contrasting to the desired formation shape than than convergence from the grey dots.

4 Results

This section will give a detailed description of the results obtained during the simulations. First, an introduction to the notation of the summary table is given. Then, the results of formation 1, 2 and 3 are described in Section 4.2, 4.3 and 4.4. Lastly, the performance with respect to each zone will be analyzed (see Section 4.5) and the performance with respect to the gains (see Section 4.6).

4.1 Notation of the result summary table

The summary tables (Figure 6, 8 and 11) include information for each simulation executed. Each table shows data from 27 simulations that have been conducted for one specific formation shape. The initial position shows the x and y coordinates of each robot's spawning position. The convergence column gives information on whether the system converged ("Yes") or not ("No"). For the purpose of simplicity, the system is stated to have converged if the final error range remains below 0.5m for each robot. Thereafter, the next column shows the time it took for the system to converge. If the system did not converge the time was not stated.

Then, the gain is given followed by the final error range. The column of the final error range shows the error of the inner-center distance to each robot and uses a variety of different notations. The notation "$0.2 >$ " means that the final error did fluctuate due to the robots readjusting progress. However, the absolute inner-center distance error never got higher than 0.2m after convergence. The notation "$1.7 =$ " shows that the robot's final error fluctuations were unusually small and remained at 1.7m. The intention for this implementation, was to identify error causes more easily and to have another qualifier for the quality of the systems final position. The notation "$0.4 \uparrow$ " illustrates that the system converged until a final error of 0.4m, but then the final error started to increase indefinitely. Lastly, a "-" sign was used if the system did not converge or did not show extraordinary behavior that could have been used for interpretation of error causes.

The "Error causes" aim to give a brief insight into the reason why an error occurred. It can be broken down to "Wrong measurement", "View blockade", "Wrongly identified", "Not identified", "Instable" and "?". A wrong measurement can be traced back to the data processing node. Even though the controller receives a value for each robot, the measurement that is published is actually wrong. A similar but slightly different error type is "wrongly identified". Each robot also receives a value for this in conjunction. However, there is now a confusion of measurements between two robots, for example the distance that should be published for the robot Nexus 2 is actually the distance from the

robot Nexus 4. This error occurs as the data processing node identifies the robots by their relative angle and not something more distinguishable, making tracking their neighbours more difficult. The error "Not identified" occurs if one or multiple robots are not sensed by the sensor. Therefore, no data is published to the controller, and thus the robot is not moving. Next, "View blockade" has the same consequences as "Not identified". Only this time the error can be further specified, meaning that one robot blocks the view between two other robots, by standing in between two robots which results in fewer robots being identified. Therefore, the two other robots cannot identify each other which leads to the incomplete transmission of the Z-value vector. Lastly, "?" means an error occurred but the causes are unknown.

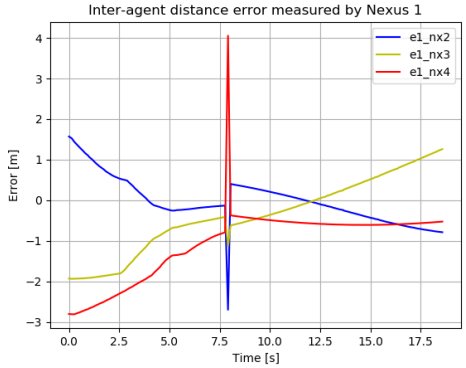
The final column shows the success percentage of each starting position, each convergence zone and each formation shape.

4.2 Formation a=3, b=4

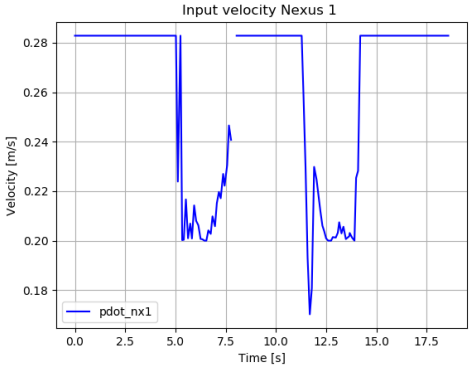
An individual description of each simulation would become extremely extensive and go beyond the scope of the research. Therefore, it was decided to choose only specific simulations that are representative for the general system behavior.

The formation a=3 b=4 demonstrated an overall success rate of 85.1% making it the highest performing formation. The average time spent converging was 26.07s. In general the system did not have noticeable difficulties with converging to the desired formation shape, considering the fact that robots' starting positions were not switched (see Figure 6, Zone 3, Position 3, Try 1-3, where each simulation failed due to a wrong identification at the beginning) or the robots did not block each others' sight of view (see Figure 6, Zone 2, Position 3, Try 3, where the view of sight between Nexus 2 and 3 was blocked by Nexus 4).

Nonetheless, one specific error did occur, which is noteworthy, as it happened during later simulations too. During the second try of position 3 in zone 2 (see Figure 5), wrong measurement values got published to the robot Nexus 1 which resulted in a crash of the controller. The exact cause of the wrong transmission cannot be diagnosed, but the interruption in the graphical plot of the input velocity (see Figure 5b) and the peaks from seconds 7.5 - 9 in the graphical plot of the inter-agent distance error measurement (see Figure 5a) are common identifications for the incident of this specific error. When reading up on the measurements inside of the data processing node, it can be seen that an inter-agent distance of below 2 got transmitted. Accordingly, it is logical that the controller would crash because it contradicts the constraint of collision avoidance (see Section 2.2.2).



(a) The inter-agent distance error measurement of Nexus 1 during the second try of position 3 in zone 2 for formation 1 (a = 3, b = 4)



(b) The input velocity measurement of Nexus 1 during the second try of position 3 in zone 2 for formation 1 (a = 3, b = 4)

Figure 5: Illustration of the inter-agent distance error measurement and input velocity of Nexus 1 during the simulation

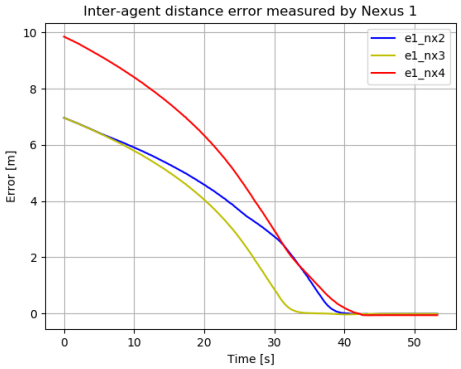
Moreover, Figure 20a shows a typical example of a successful convergence to formation 1. Note that exponential convergence is not observable as the speed of the Nexus robots got limited to a maximum of $0.2\frac{m}{s}$ (see Section 3.1.1).

Desired formation		Initial position								Try	Convergence	Time [s]	Gain	Final error range [m]	Error cause	Success [%]		
a	b	Nexus 1		Nexus 2		Nexus 3		Nexus 4										
		x	y	x	y	x	y	x	y									
3	4	0	-1	4	-1	0	4	4	4	1	Yes	7	10	0.2 >	-	100	100	85.1
										2	Yes	5	20	0.2 >	-			
										3	Yes	3	50	0.2 >	-			
		0	0	2.5	0	0	3.5	3	3	1	Yes	20	10	0.25 >	-			
										2	Yes	18	20	0.2 >	-	100		
										3	Yes	33	50	0.2 >	-			
		0	0	3.65	-0.25	1	3	4.25	3	1	Yes	82	10	0.2 >	-			
										2	Yes	86	20	0.25 >	-	100		
										3	Yes	60	50	0.25 >	-			
3	4	-0.9	-1.9	4.9	-1.9	-0.9	4.9	4.9	4.9	1	Yes	28	10	0.1 >	-			
										2	Yes	12	20	0.05 >	-	100		
										3	Yes	8	50	0.05 >	-			
		2	-1.5	4.5	1	-0.9	4.9	5	3	1	Yes	30	10	0.3 >	-			
										2	Yes	20	20	0.5 >	-	100	88.7	
										3	Yes	15	50	0.3 >	-			
		0.5	1	3.5	-2.5	-1	2.5	2.5	2	1	Yes	20	10	0.3 >	-			
										2	No	-	20	-	Wrong measurement	66		
										3	No	-	50	-	View blockade			
3	4	-3	-4	7	-4	-3	7	7	7	1	Yes	22	100	0.2 >	-			
										2	Yes	23	200	0.3 >	-	100		
										3	Yes	20	500	0.2 >	-			
		1	-6	6.5	-0.5	-3	7	7	6	1	Yes	27	100	0.2 >	-			
										2	Yes	27	200	0.2 >	-	100	66.7	
										3	Yes	27	500	0.2 >	-			
		0	-4	8.5	3	-1	7.5	7.5	-0.5	1	No	-	100	-	Wrongly indentified			
										2	No	-	200	-	Wrongly indentified	0		
										3	No	-	500	-	Wrongly indentified			

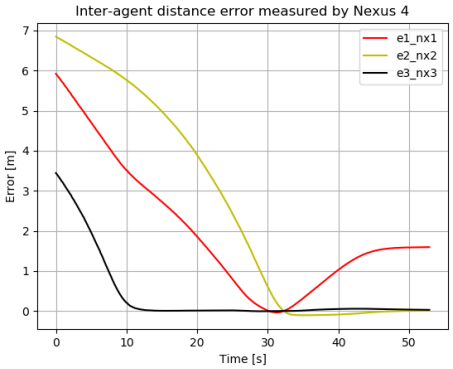
Figure 6: Table that summarizes important data from the 27 simulations done on the Formation 1 (a=3, b=4)

4.3 Formation a=5, b=5

The formation a = 5 b = 5 had a success rate of 51.85% with an average convergence time of 22.69s. The average final error range of the formation is 0.16m. If the system did converge, it happened relatively smoothly and faster compared to the simulations of other formations. Figure 7a shows a good example for a stable convergence.



(a) The inter-agent distance error measurement of Nexus 1 during the first try of position 1 in zone 3 for formation 2 (a = 5, b = 5)



(b) The inter-agent distance error measurement of Nexus 4 during the first try of position 2 in zone 3 for formation 2 (a = 5, b = 5)

Figure 7: Illustration of two inter-agent distance error measurements of Nexus 1 and Nexus 4 during the simulation

The first errors that occured in zone 1, position 3, try 1-3 were foreseeable as the positioning of the robots triggered a view blockade between Nexus 2 and Nexus 3 caused by Nexus 4 (see Figure 18 the blue dots in zone 2). This had the consequence that Nexus 2 and 3 only recognized 2 robots which shortened the Z-value vector, thus the controller used wrong or no values for its calculation. In most of the cases the agents started drifting away from each other as wrong velocity inputs are received.

However, in zone 3 another error occurred that cannot be pinpointed. Essentially, as soon as the robots began converging to their desired formation shape everything seemed to work accordingly. Nevertheless, once all the agents reached the minimum inter-agent distance error, Nexus 1 and Nexus 4 started drifting away until they have reached an error of 1.7m, while Nexus 2 and Nexus 3 kept their desired inter-agent distance of 5m. Figure 7b and Figure 20b,c show the same behavior during other simulations. This behavior together with a possible solution will be further discussed in Section 5.1.

Desired formation		Initial position								Try	Convergence	Time [s]	Gain	Final error range [m]	Error cause	Success [%]		
a	b	Nexus 1		Nexus 2		Nexus 3		Nexus 4										
		x	y	x	y	x	y	x	y									
5	5	0.5	0.5	4.5	0.5	0.5	4.5	4.5	4.5	1	Yes	17.5	10	0.2 >	-	100	88.9	51.8
										2	Yes	10	20	0.15 >	-			
										3	Yes	6	50	0.2 >	-			
		0	-1	5	1	-1	5	6	5	1	No	20	100	0.6 >	Instable	66.6		
										2	Yes	10	200	0.1 >	-			
										3	Yes	7	500	0.3 >	-			
		0	1	5	1	0	4	5	4	1	Yes	8	100	0.3 >	-	100		
										2	Yes	25	200	0.4 >	-			
										3	Yes	14	500	0.2 >	-			
5	5	-1.4	-1.4	6.4	-1.4	-1.4	6.4	6.4	6.4	1	Yes	10	100	0.3 >	-	100	44.4	
										2	Yes	9	200	0.2 >	-			
										3	Yes	8	500	0.1 >	-			
		-1	1.5	5	1	-1	5	6	5	1	Yes	45	100	0.4 >	-	33.3		
										2	No	-	200	0.5 =	?			
										3	No	-	500	0.4 ↑	Wrong measurement			
		1	1.5	7	0	0	7	3.5	4	1	No	-	100	-	Wrongly indentified	0		
										2	No	-	200	-	Wrongly indentified			
										3	No	-	500	-	Not identified			
5	5	-3.5	-3.5	8.5	-3.5	-3.5	8.5	8.5	8.5	1	Yes	45	100	0.1 >	-	66.6	22.2	
										2	Yes	29	200	0.1 >	-			
										3	No	-	500	-	Measurement error			
		-3.5	-1	3	-5	0	10	7.5	6	1	No	-	100	1.7 =	Measurement error	0		
										2	No	-	200	1.7 =	Measurement error			
										3	No	-	500	-	Measurement error			
		2	1.5	7	0	0	4	5	4	1	No	-	100	-	Wrong measurement	0		
										2	No	-	200	0.15 >	Wrongly indentified			
										3	No	-	500	2.5 =	?			

Figure 8: Table that summarizes important data from the 27 simulations done on the Formation 2 (a=5, b=5)

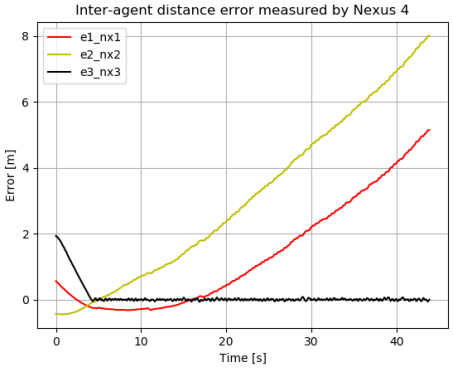
4.4 Formation a=2.5, b=5

The formation a = 2.5 and b = 5 did have the same success percentage as formation 2 of 51.85%. The average convergence time of formation 3 is 69.7s and the average final error range is 0.167m.

The formation shape is the most unstable due to an uncertainty of the data processing node. The simulations of zone 2 have an relatively low success rate of 11.1%. This is caused by a wrong measurement transmission of Nexus 3. As it can be seen from Figure 9 the Nexus 3 robot did have a constant distance to Nexus 1 and Nexus 4 even though Nexus 4 continuously drifted away from Nexus 1. This type of system behavior is not possible. In the simulation environment it was observed that Nexus 3 actually kept the intended inter-agent distance of 2.5m to Nexus 4, but drifted away from Nexus 1. Therefore, it can be concluded that the Z-value vector, transmitted from the data processing node to the controller of Nexus 1, was wrong. however, the reason for the wrong measurement is unknown. Further examples of the same behavior are given in Figure 20d, e in the Appendix.

Furthermore, another error was observed inside of zone 3. During each simulation within zone 3, the system started converging accordingly until Nexus 4 had reached a specific relative position to the other robots. As soon as this position was reached, Nexus 4 received frequently data from four robots instead of three robots. Consequently, the graph

(a) The inter-agent distance error measurement of Nexus 1 during the third try of position 3 in zone 2 for formation 3 (a = 2.5, b = 5)

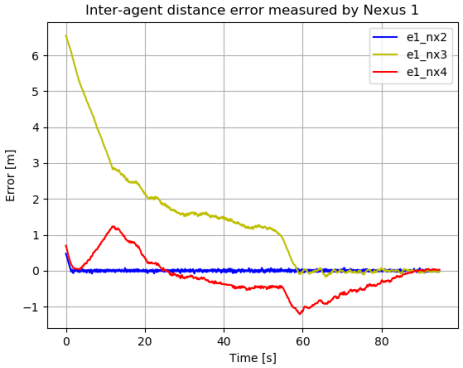


(b) The inter-agent distance error measurement of Nexus 4 during the third try of position 3 in zone 2 for formation 3 (a = 2.5, b = 5)

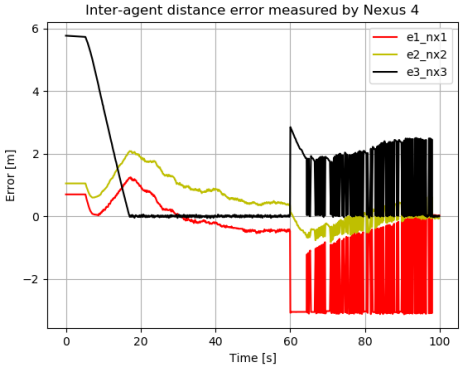
Figure 9: Comparison of the measurements of the Nexus robots 1 4 during the same simulation

jumps back and fourth between two values which caused the fluctuations in the graphical plot (see Figure 10b). However, the other three agents continued to receive measurements of only three robots (see Figure 10a). The ramifications of the wrong measurements were that it was arbitrary if the system did converge or not. If the correct measurements outweigh the wrong measurements, the system would converge (see Figure 20f and Figure 21a-c). However, there are also multiple examples where the system did not converge anymore (see Figure 21d,e).

When analyzing the graph accurately, it can be observed that the upper and lower values of Nexus 1 copy the same trend as the upper and lower values of Nexus 3. This symmetry of both trends might hint a confusion of the data processing node between the measurements of Nexus 1 and 3. A possible cause and solution to fix this problem will be discussed in Section 5.2.



(a) The inter-agent distance error measurement of Nexus 1 during the second try of position 3 in zone 3 for formation 3 (a = 2.5, b = 5)



(b) The inter-agent distance error measurement of Nexus 4 during the second try of position 3 in zone 3 for formation 3 (a = 2.5, b = 5)

Figure 10: Comparison of the measurements of the Nexus robots 1 & 4 during the same simulation

Desired formation		Initial position								Try	Convergence	Time [s]	Gain	Final error range [m]	Error cause	Success [%]		
a	b	Nexus 1		Nexus 2		Nexus 3		Nexus 4										
		x	y	x	y	x	y	x	y									
2.5	5	-0.5	-0.5	3	-0.5	-0.5	5.5	3	5.5	1	Yes	40	10	0.2 >	-	100	100	51.8
										2	Yes	20	20	0.1 >	-			
										3	Yes	17	50	0.4 >	-			
		0	-1	2.5	1	0	6	2.5	4	1	Yes	100	10	0.4 >	-	100		
										2	Yes	70	20	0.3 >	-			
										3	Yes	60	50	0.1 >	-			
		-0.5	-0.5	2	0.5	0	6	3.5	5	1	Yes	150	10	0.4 >	-	100		
										2	Yes	120	20	0.4 >	-			
										3	Yes	90	50	0.2 >	-			
2.5	5	-1.4	-1.4	3.9	-1.4	-1.4	6.4	3.9	6.4	1	Yes	60	100	0.1 >	-	33.3	11.1	
										2	No	-	200	-	Measurement error			
										3	No	-	500	-	Measurement error			
		0	-1.5	3.5	1	-0.5	3.5	1	5	1	No	-	100	-	Measurement error	0		
										2	No	-	200	-	Measurement error			
										3	No	-	500	-	Measurement error			
		0.5	1	-2	2.5	0	6	4	4	1	No	-	100	-	Measurement error	0		
										2	No	-	200	-	Measurement error			
										3	No	-	500	-	Measurement error			
2.5	5	-3.5	-3.5	6	-3.5	-3.5	9.5	6	9.5	1	No	-	100	-	Measurement error	0	44.4	
										2	No	-	200	-	Measurement error			
										3	No	-	500	-	Measurement error			
		-3.5	-3.5	3	-5	-3	5	3	2.5	1	Yes	80	100	0.1 >	-	66.6		
										2	Yes	50	200	0.2 >	-			
										3	No	-	500	-	Measurement error			
		2	-3	5	-3	7.5	-3	4	3	1	Yes	80	100	0.2 >	-	66.6		
										2	Yes	90	200	0.2 >	-			
										3	No	-	500	-	Measurement error			

Figure 11: Table that summarizes important data from the 27 simulations done on the Formation 3 (a=2.5, b=5)

4.5 Performance of each zone

The results of the three previous formation shapes have been restructured into a table (see Figure 13). The new table focuses on the behavior of the three zones of the three formation shapes with respect to the success percentage, convergence time and average

final error. The table is also visualized in three bar charts (see Figure 12a), 12b) and 12c)).

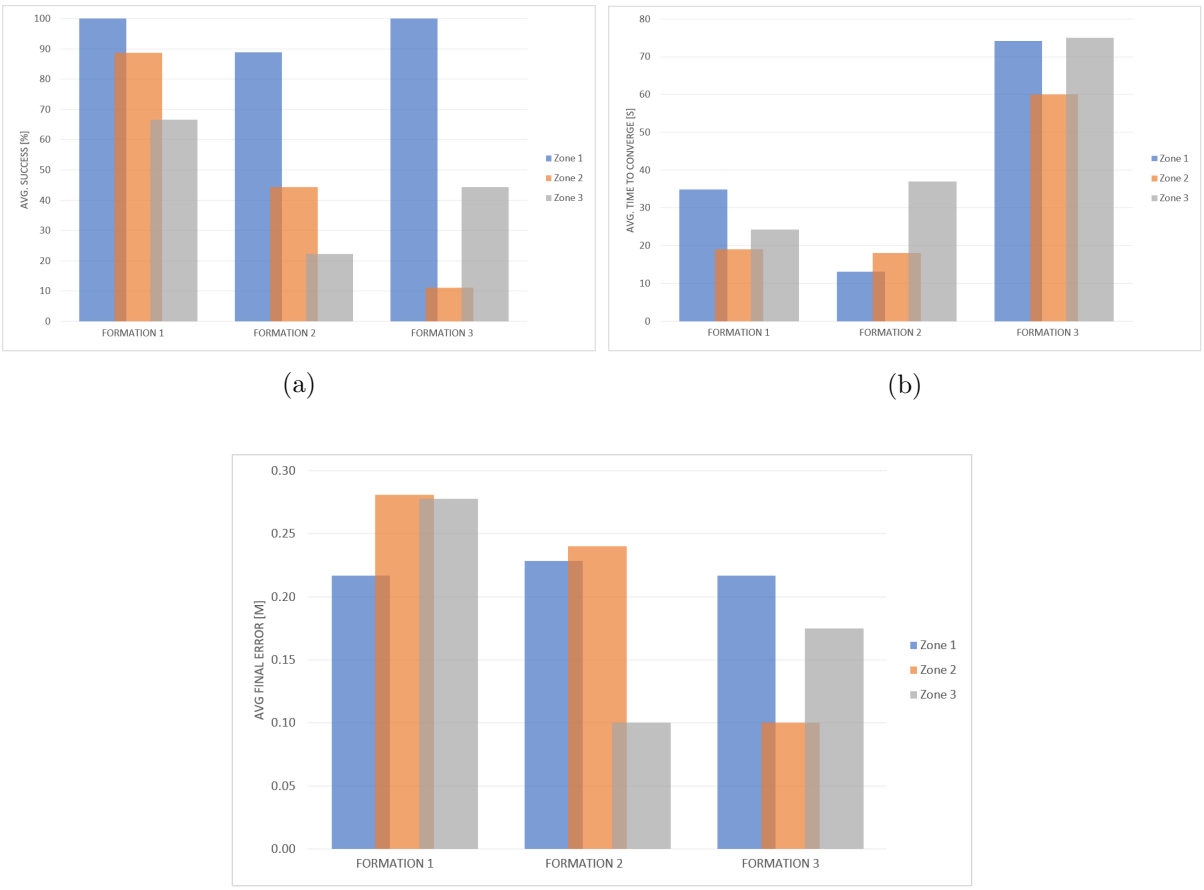


Figure 12: The three diagrams illustrate the performance each formation inside of the three zones with respect to the success percentage, the convergence time and final error range

Figure 12a shows the success percentage with respect to each formation shape and the zones 1 - 3 separated. Overall, the formation a=3 b=4 has the highest success rate of all zones combined, followed by formation 2 and 3 which actually perform equally high. Formation 1 and 2 show the trend that the success rate decreases with increasing zones or in other words, with an increasing distances from the solution point. However, zone 2 is performing the lowest in formation 3.

Secondly, Figure 12b illustrates the convergence time with respect to each formation shape and the zones 1 - 3 separated. Formation 3 has the highest convergence time by far while Formation 1 and 2 needed approximately the same overall time to converge. There is no obvious trend between the zones.

Finally, Figure 12c demonstrates the final error range with respect to each formation shape and the zones 1 - 3 separated. Even though there is no evident trend between the zones, there is a trend between the three formation shapes. Generally, the overall final error did reduce slightly from formation 1 to formation 2 to formation 3.

Criteria	Zone 1			Zone 2			Zone 3		
	Formation 1	Formation 2	Formation 3	Formation 1	Formation 2	Formation 3	Formation 1	Formation 2	Formation 3
Success	100	88.9	100	88.7	44.4	11.1	66.7	22.2	44.4
Average Time [s]	34.89	13.06	74.11	19.00	18.00	60.00	24.33	37.00	75.00
Time per Zone [s]	40.69			32.33			45.44		
Average final error [m]	0.22	0.23	0.22	0.28	0.24	0.10	0.28	0.10	0.18
Final error per Zone [m]	0.22			0.21			0.18		

Figure 13: The table analyzes the performance of each formation with respect to the three different zones in terms of its convergence time, final error range and success percentage

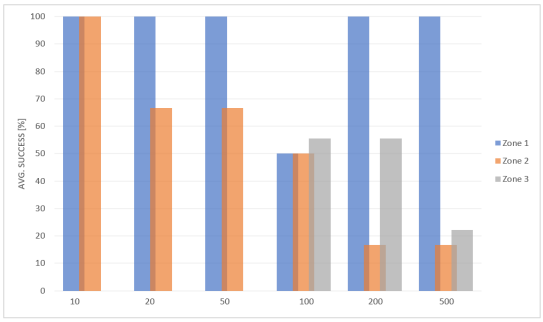
4.6 Performance of different gains

In order to analyze the influence of the gains on the success percentage, convergence time and final error another table has been created, summarizing the necessary information (see Figure 15). The table has been visualized in three different bar charts (see Figure 14a, 14b and 14c) which will be described in this section.

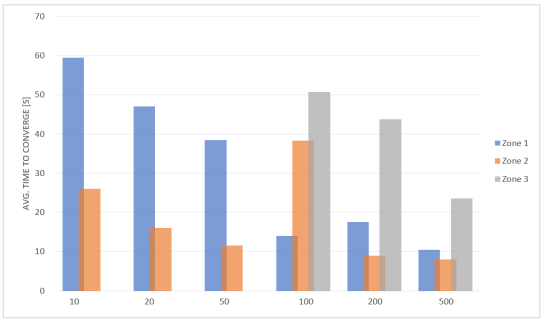
The first chart (Figure 14a) shows the success percentage with respect to each gain and the zone 1-3 separated. It is easily visible that zone 1 is performing the highest for all gains used except for the gain 100, where zone 3 performs slightly better. Additionally, it can be said that there is a correlation between an increasing success percentage and decreasing gain.

Secondly, Figure 14b illustrates the convergence time with respect to each gain and the zones 1 - 3 separated. The most obvious trend is that the convergence time decreases if the gain increases inside the same zone. However, zone 2 at gain 100 represents an outlier as the convergence time jumps up to 38.33s. Furthermore, the larger zones most commonly have the highest convergence times within the same gain compared to the lower zones.

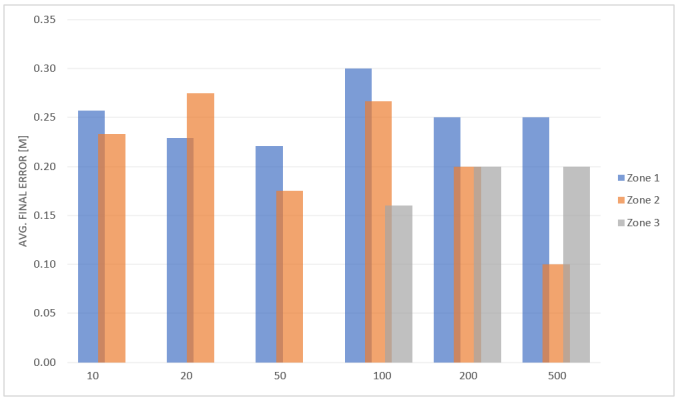
Finally, Figure 14c demonstrates the final error range with respect to each gain and the zones 1 - 3 separated. Zone 1 gain 100 shows the maximum final error range while zone 2 gain 500 has the smallest final error range. Besides that, there are no obvious trends within the data.



(a)



(b)



(c)

Figure 14: The three diagrams illustrate the performance each gain (10, 20, 50, 100, 200, 500) inside of the three zones with respect to the success percentage, the convergence time and final error range

	Zone 1						Zone 2						Zone 3		
Gains	10	20	50	100	200	500	10	20	50	100	200	500	100	200	500
Avg. Time [s]	59.50	47	38.43	14	17.50	10.50	26	16	11.50	38.33	9	8	50.80	43.80	23.50
Avg. Success [%]	100	100	100	50	100	100	100	66.67	66.67	50	16.67	16.67	55.56	55.56	22.22
Avg. Final error [m]	0.257	0.229	0.221	0.300	0.250	0.250	0.233	0.275	0.175	0.267	0.200	0.100	0.160	0.200	0.200

Figure 15: The table analyzes the performance of each formation with respect to the used gains (10, 20, 50, 100, 200, 500) in terms of its convergence time, final error and success percentage

5 Discussion

The research intends to explore the global convergence capabilities of the control law made by Chan et al. (2020) using computer simulation. The simulation series that was executed to test the control law's limitations revealed discrepancies which will be elaborated on in this section. Finally, whether the research questions, formulated in Section 1.1.5 and 1.1.5, are answered by the results will be discussed.

5.1 Error of 1.7m

The cause for a displacement of 1.7m of the robots Nexus 1 and Nexus 4 is unknown, yet a certain intervention of the system might offer a solution to the problem.

During the simulation of the formation described in Section 4.3, it was observed that the connections of the robots Nexus 2 and Nexus 3 remained at a stable distance. It gave the impression to the researcher that the connections (1↔2, 1↔3, 2↔4, 3↔4) are more prioritized by the system than the connection between robot 1 and 4. In other words, if Nexus 2 and Nexus 3 approached each other, it would have been possible for them to keep the intended distance to Nexus 1 and Nexus 4 by pushing the robots more outward. However, Nexus 2 and Nexus 3 are not able to sense that the connection 1↔4 becomes increasingly incorrect due to the fact that each robot is decentralized and can only access its own information. This means that Nexus 4 will be displaced by the 'power' of two robots while trying to keep the desired distance to Nexus 1 on its own.

A possible solution therefore might be to implement another connection between robot 2 and robot 3 so that the formation's rigidity increases.

5.2 Data processing problem

In order to find a potential solution, that is able to fix the data processing problem described in Section 4.4, the cause of the fluctuation needs to be traced back to its origin.

Firstly, fluctuations are caused by a confusion between the robots 1 and 3, which is caused by the Z-vector. The Z-vector varies in size when the fluctuations occur because the Z-vector was designed in such a way that it can store the information of any number of robots. Therefore, the vector has three entries reserved for each robot that is recognized by the laser. If the laser suddenly detects more than three robots, new entries will be added into the Z-vector accordingly. The new vector might then move information from its initial position and replace it with data of the additional robot. As a result, the controller will call misleading information.

Nonetheless, the vector varies in size due to the wrong identification of the data processing node. As described in Section 3.1.2, the data published by the laser scanner is a vector with ideally three clusters because three robots are visible. However, in this scenario the laser scanner will publish four clusters to the data processing node which then concludes that the robot Nexus 4 is surrounded by four robots. A clearer understanding can be obtained by looking at the following scenario. For example, a situation is created where only the robots Nexus 4 and Nexus 2 are considered (see Figure 16). Nexus 4 starts to scan for Nexus 2 with its rotating laser sensor, although this time, the starting position of the laser scanner will be pointed on the center of Nexus 2. Then the scan will continue clockwise around Nexus 4. However, as soon as the laser scan reaches its final position, Nexus 2 will be measured a second time. This is extremely crucial as now the published vector includes two clusters which are placed at the end and beginning of the array. The data processing node will then interpret the two clusters as two robots even though only one robot is recognized.

It is assumed that a similar situation occurs within the simulation of formation 3. The agent starts to fluctuate once changing velocity inputs are received by the controller. An improved code for the data processing node, that is able to combine data clusters from the same robot, should be implemented into the controller.

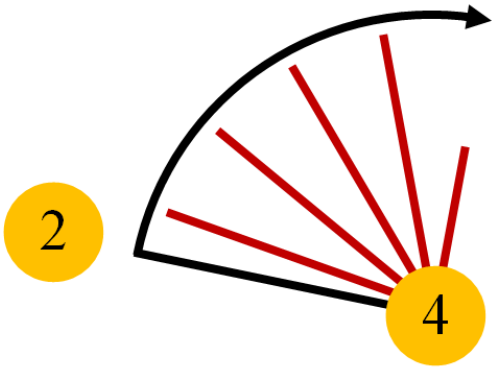


Figure 16: Illustration of the assumed scenario in Section 5.2

5.3 Research questions answered by the thesis

The research aimed to find an answer to the following central question: "What is the performance of the control law for bearing-based formation control with respect to global

convergence using computer simulation?". The question was then broken down into several sub questions which are stated in Section 1.1.5. This subsection will discuss the answers to the research question.

The results have found that the control law does converge with a success percentage of 62.9% to its desired formation shape. Additionally, provided that the gain is chosen appropriately, the control law does not show a boundary of the robot's distance away from its point of solution. Nonetheless, the robots most probably do not converge if:

1. the robots rotational order is changed.
2. the robots are blocking each others sight of view.
3. the robots receive wrong measurement inputs by the data processing node.

The reasons for a simulation failure can , except for a few unknown cases, all be tracked down to the data processing node or laser scanner. Not a single simulation failure could be identified that failed due to the wrong computations of the control law. Therefore, it can be assumed that the control law is globally convergent. However, global convergence cannot be guaranteed as the chance of failure during a simulation was simply too high hence a well-founded statement about the global convergence behavior of the control law cannot be made.

First, the measurement and tracking inside of the simulation needs to be improved significantly by rewriting the code for the data processing node. A new code will make the simulation more stable guaranteeing a more accurate analysis of the control law. However, revision of the data processing node goes beyond the scope of the research and therefore it is suggested as future research.

6 Future Research

The simulation results showed that there is a high chance of failure due to the unreliability of the data processing node. Frequently, inaccurate and partially wrong data was published by the data processing node to the controller of the robots and especially to the robot 'Nexus 4'.

Although, the results of the research did diagnose the limitations of the control law, the gathered data proved to be too unreliable because the simulation is highly error prone. Therefore, it is suggested to improve the code of the data processing node further and repeat a series of simulations. Comparing the gathered data to the results of this research will identify similarities and dissimilarities, thus supporting the credibility of this research's findings. Moreover, new results can be found that will add on the knowledge acquired in this research.

Another implementation into the system that is worth analyzing, is the addition of a sixth edge between Nexus 2 and Nexus 3. Section 5.1 already mentioned that a new edge might strengthen the formation's stability. However, simulations need to be performed in order to see if a sixth edge would remove fluctuations and stop Nexus 1 and Nexus 4 from drifting outside of the formation shape.

7 Conclusion

In this thesis, a theoretically proven bearing-based formation control law was analyzed on the control law's performance with respect to its global convergence capabilities. In order to test the control law, a series of simulations were executed on the software ROS Melodic. Additionally, each robot used local information acquired only by a RPlidar laser scanner mounted on to the robot, thus the control law did not make use of a centralized system. The aim of the research was to identify the region of attraction, the success percentage, the control law's limitations and the reasons of failure. Therefore, 81 distinct simulations tested a variety of scenarios, thus diagnosing the flaws of the simulation and control law.

The results have shown that the overall success percentage is 62.9%. Generally, it was observed that an increasing displacement of the agents, away from the desired formation shape, correlates with an increasing failure rate and convergence time. However, the final error range did not correspond to the robot's displacement towards its desired location. Nonetheless, there was no region of attraction identified. Furthermore, the simulations mainly failed due to a false transmission of distance measurements which also was the main cause for a system failure. The wrong measurements were caused by the data processing node that performed poorly in the interpretation of the laser scanner's data. More specifically, it is suggested to rewrite the code of the data processing node, so that tracking and identification features will be improved.

In conclusion, it is assumed that the control law is globally convergent. However, the system is currently unstable, making the gathered data unreliable. Therefore, it is suggested to improve the conversion of laser scanner measurement into usable data by rewriting the code of the data processing node. Consequently, the new simulation will be more stable, and the new findings will support the assumptions of the research.

References

Ahn, H.-S. (2019), *Formation Control*, Vol. 205, Springer, Cham.
URL: *https://ebookcentral.proquest.com/lib/[SITE_ID]/detail.action?docID* = 5742866

Cao, K., Li, D. & Xie, L. (2019), 'Bearing-ratio-of-distance rigidity theory with application to directly similar formation control', *Automatica* **109**.

Chan, N. P., Jayawardhana, B. & de Marina, H. G. (2020), 'Angle-constrained formation control for circular mobile robots'.

de Queiroz, M., Cai, X. & Feemster, M. (2019), *ormation Control of Multi-Agent Systems*, Wiley, CPI Group (UK) Ltd, Croydon, CR0 4YY.

Guo, J., Lin, Z., Cao, M. & Yan, G. (2010), 'Adaptive leader -follower formation control for autonomous mobile robots'.

Li, X., Luo, X., Wang, J., Zhu, Y. & Guan, X. (2018), 'Bearing-based formation control of networked robotic systems with parametric uncertainties', *Neurocomputing* pp. 234–235.

Li, X. & Xie, L. (2018), 'Dynamic formation control over directed networks using graphical laplacian approach', *TRANSACTIONS ON AUTOMATIC CONTROL,* (11), 3761–3774.
URL: *https://ieeexplore-ieee-org.proxy-ub.rug.nl/stamp/stamp.jsp?tp=arnumber=8270712*

Rasshofer, R. H. & Gresser, K. (2005), 'Automotive radar and lidar systems for next generation driver assistance functions'.

Shan, J. & Toth, C. K. (2018), *Topographic Laser Ranging and Scanning: Principles and Processing*, second edn, CRC Press.
URL: *https://books.google.nl/books?id=dGpQDwAAQBAJlr=source=gbs_navlinks_s*

Sidman, J. & John, A. S. (2017), 'The rigidity of frameworks:theory and applications'.

Slamtec (2018), 'Prlidar a1'.
URL: *https://www.slamtec.com/en/Lidar/A1*

Trinh, M. H., Zhao, S., Sun, Z., Zelazo, D., Anderson, B. D. O., Fellow, L. & Ahn, H.-S. (2019), 'Bearing-based formation control of a group of agents with leader-first follower structure', *IEEE Transactions on Automatic Control* (2), 598–613.

Appendix

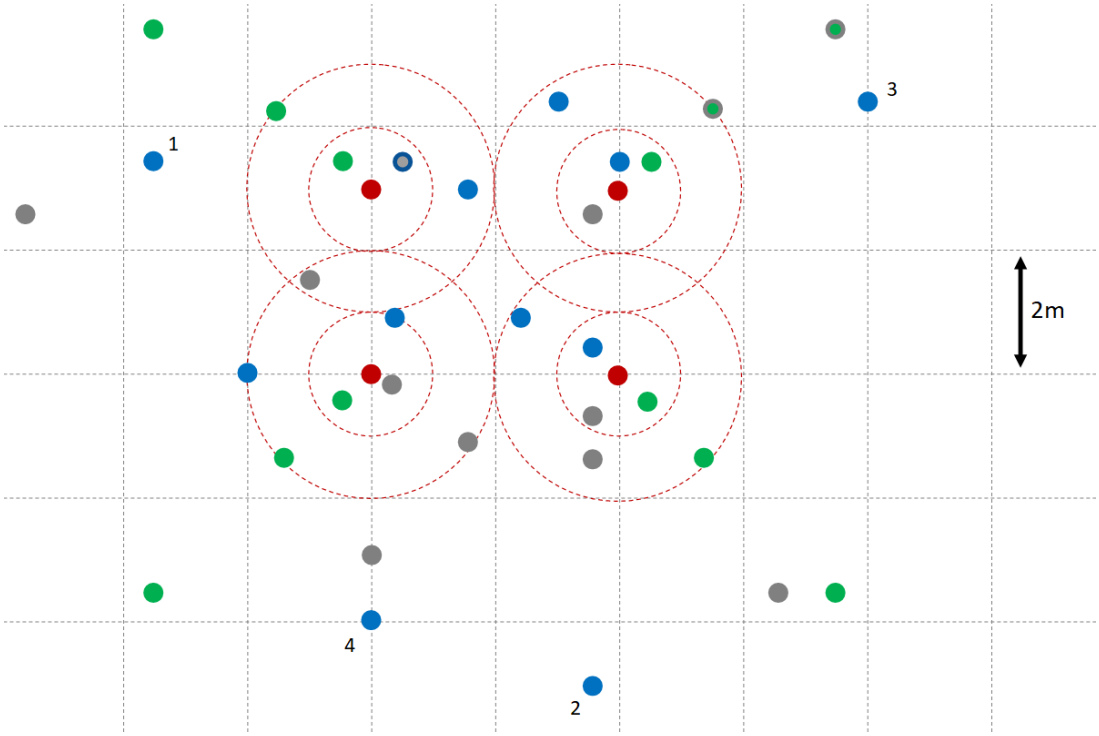


Figure 17: Visual illustration of the spawning coordinates of each nexus robot for the formation a=3 b=4

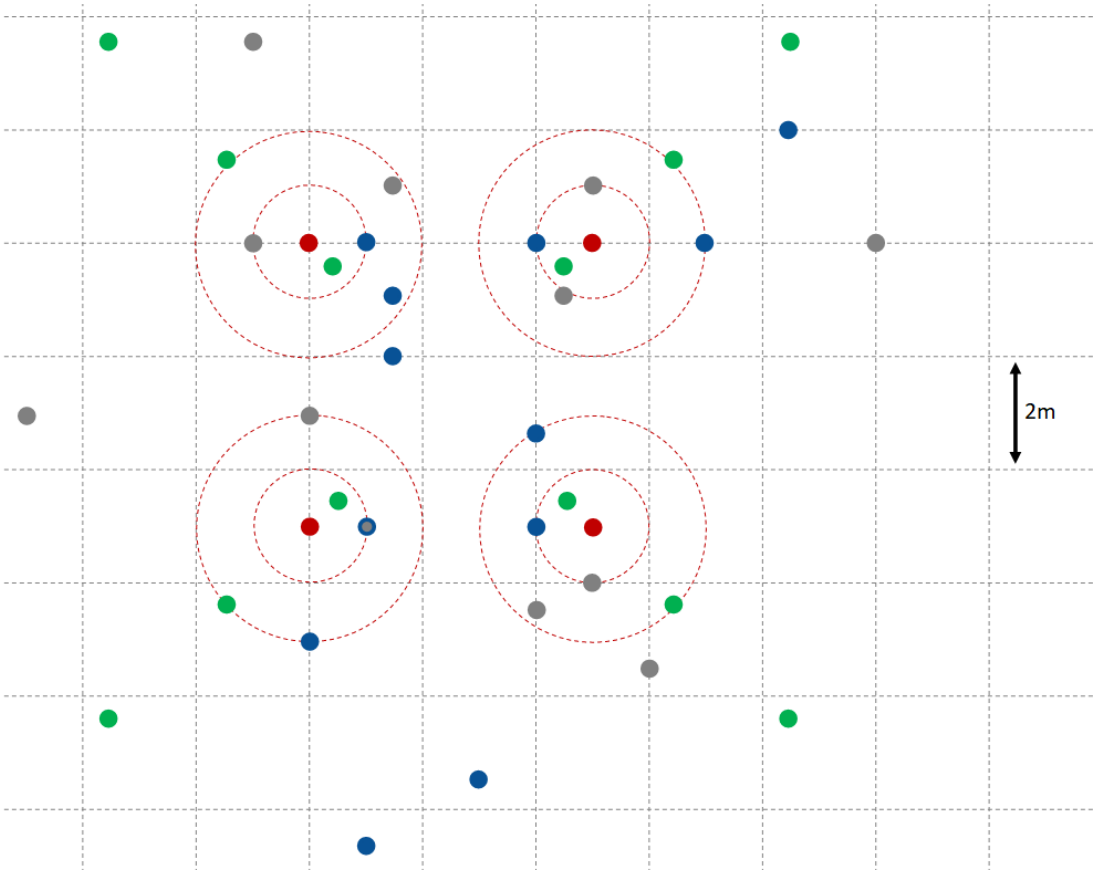


Figure 18: Visual illustration of the spawning coordinates of each nexus robot for the formation a=5 b=5

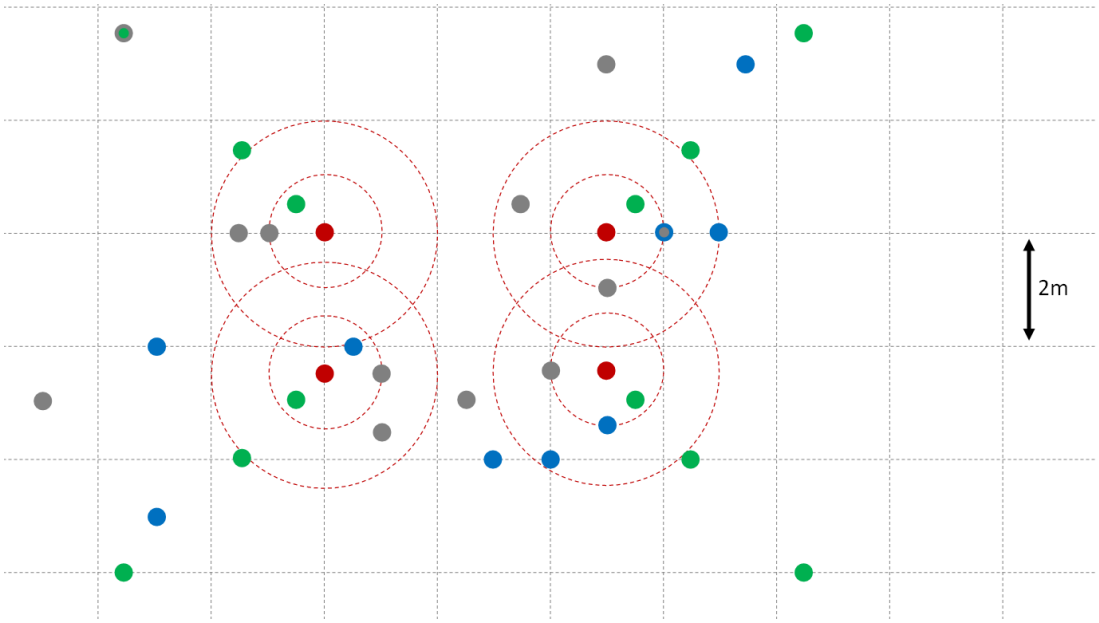
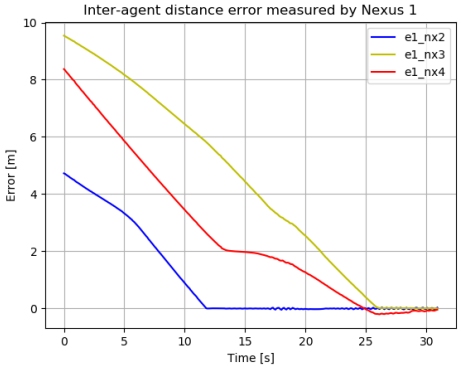
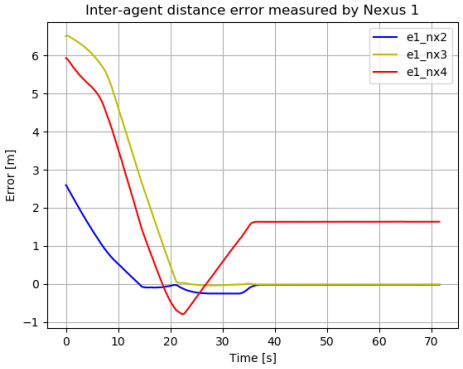


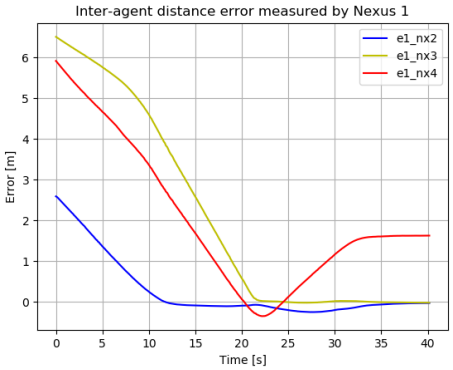
Figure 19: Visual illustration of the spawning coordinates of each nexus robot for the formation a=2.5 b=5



(a) The inter-agent distance error measurement of Nexus 1 during the first try of position 2 in zone 3 for formation 1 (a = 3, b = 4)



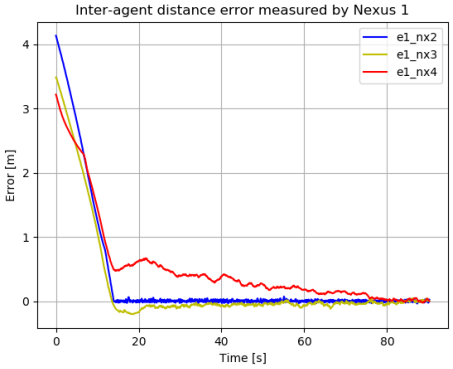
(b) The inter-agent distance error measurement of Nexus 1 during the third try of position 2 in zone 3 for formation 2 (a = 5, b = 5)



(c) The inter-agent distance error measurement of Nexus 1 during the second try of position 2 in zone 3 for formation 2 (a = 5, b = 5)

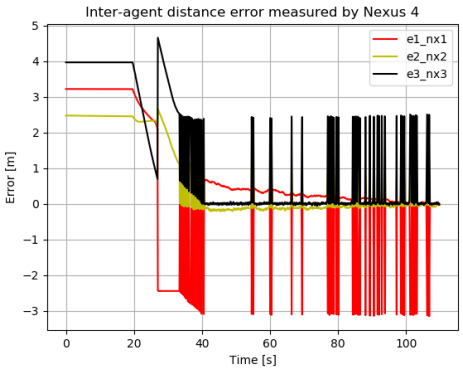
(d) The inter-agent distance error measurement of Nexus 1 during the first try of position 3 in zone 2 for formation 3 (a = 2.5, b = 5)

(e) The inter-agent distance error measurement of Nexus 1 during the second try of position 3 in zone 2 for formation 3 (a = 2.5, b = 5)

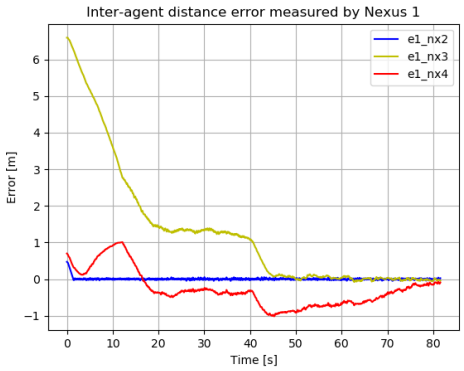


(f) The inter-agent distance error measurement of Nexus 1 during the first try of position 1 in zone 3 for formation 3 (a = 2.5, b = 5)

Figure 20: Collection of diagrams from ROS for illustration purposes Part 1



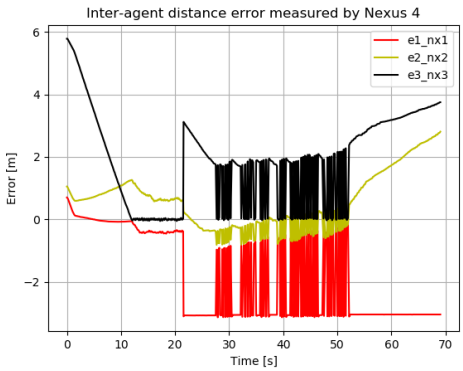
(a) The inter-agent distance error measurement of Nexus 4 during the first try of position 1 in zone 3 for formation 3 (a = 2.5, b = 5)



(b) The inter-agent distance error measurement of Nexus 1 during the first try of position 3 in zone 3 for formation 3 (a = 2.5, b = 5)

(c) The inter-agent distance error measurement of Nexus 4 during the first try of position 3 in zone 3 for formation 3 (a = 2.5, b = 5)

(d) The inter-agent distance error measurement of Nexus 1 during the third try of position 3 in zone 3 for formation 3 (a = 2.5, b = 5)



(e) The inter-agent distance error measurement of Nexus 4 during the third try of position 3 in zone 3 for formation 3 (a = 2.5, b = 5)

Figure 21: Collection of diagrams from ROS for illustration purposes Part 2

```
#!/usr/bin/env python

from __future__ import division

import sys
from math import *
import rospy
import numpy as np
from rospy_tutorials.msg import Floats
from std_msgs.msg import Int32
from rospy.numpy_msg import numpy_msg
from sensor_msgs.msg import LaserScan


class determine_z_values:
    ''' Determines the inter-agent distance values and publishes it to z_values topic '''

    ''' NOTE: this script requires the simulation with LIDAR to be running first, as well as an input
        argument, an example of how to properly run this code in the terminal is as following:

        rosrun lasmulticontrol3 dataprocessingnode_N.py "1"

    Laser scanner angle zero is in the backward direction wrt the robot as used here.
    We use angles (0:360) '''

    def __init__(self):
        ''' Initiate self and subscribe to /scan topic '''

        # which nexus?
        self.name = 'n_1'

        # Desired distance - used for sending if no z is found or if the dataprocessingnode is shutdown:
        # robot not influenced if one z not found
        self.d = np.float32(3)
        self.dd = np.float32(np.sqrt(np.square(self.d)+np.square(self.d)))

        # set min and max values for filtering ranges in meter during initiation
        self.min_range = 0.25
        self.max_range = 50

```

```
41
42          # prepare shutdown
43          self.running = True
44          rospy.on_shutdown(self.shutdown)
45
46          # prepare publisher
47          self.pub = rospy.Publisher(self.name +'/z_values', numpy_msg(
    Floats), queue_size=1) #Publish the distances and angles to agents in
     range
48          self.PUB = rospy.Publisher(self.name +'/agents', Int32 ,
    queue_size=1) #Publish the amount of surrounding agents
49
50  # subscribe to /scan topic with calculate_z as callback
51          rospy.Subscriber('/n_1hokuyo_points', LaserScan, self.
    calculate_z)
52
53          np.set_printoptions(precision=2)
54
55      def calculate_z(self, msg):
56          ''' Calculate the z_values from the scan data '''
57          # Check if not shutdown
58
59          self.ranges= np.asarray(msg.ranges)
60          # print self.ranges
61
62          if self.running:
63
64
65                  # Save the angles (hits) of the robots in seperate
    arrays
66                  z_a = np.where((self.ranges >= self.min_range) & (self.
    ranges <= self.max_range))[0] #The zero at the end is to access the
    first value of the tuple created by "np.where", so z_a is just an
    array
67
68                  n = 1
69                  # print 'z_a = ',z_a
70   for i in range(len(z_a)-1):  # Calculates the number of robots
71
72                      if (z_a[i] - z_a[i + 1] >= -10):
73
74                          continue
75
76                      elif (-10 <= z_a[i] - z_a[i-1] <= 10):
77
78                          n = n + 1
79
```

```
R=[]
            r=np.array([])
Zval=np.array([])

            P = 0
# Compares difference between angles in z_a to decide if it's a new
 robot, and if it is it creates and array r, which is then added to
 the list R
for i in range(P,len(z_a)-1):


                    if (z_a[i] - z_a[i + 1] >= -10) and i!=(len((z_a))
 -2):

                        r=np.append(r,z_a[i])
    elif (-10 <= z_a[i] - z_a[i - 1] <= 10):
                        r=np.append(r,z_a[i])
                        R.append(r)
                        P = 1+i
                        r=np.array([])

    elif (z_a[i] - z_a[i + 1] >= -10) and i==(len((z_a))-2):
                        r=np.append(r,z_a[i])
                        R.append(r)


for i in range(n):  #transform list R to array of integers
                    R[i]=R[i].astype(int)
                    #print R

                if z_a[0]==0 and z_a[-1]==7199:  #This loop makes sure
 that a robot is not read twice if it is located at the scan starting
 point (Doesn't work perfectly)
                    R[0]= np.append(R[0],R[-1])
                    del R[-1]
    n=n-1

                self.PUB.publish(n)
                k = 2


                #print(R)
                self.z_aX=np.zeros([len(R)])
                self.zn_X=np.zeros([len(R)])
self.z_a_min=np.zeros([len(R)])
        self.z_a_max=np.zeros([len(R)])
        self.zn_min=np.zeros([len(R)])
```

```
122         self.zn_max=np.zeros([len(R)])
123 
124                 for j in range(0,n): #This section makes matrices for
    the distances to each neighbour
125 
126                     if R[j] != np.array([]):
127 
128                         self.z_aX[j] = int((np.round((R[j]).mean()))
    /(200))  #Matrix of mean distance from the read values to each
    neighbour
129                         # /200 is needed for a resolution of 10 and
    samples of 7200 10*20=200 (see hokuyo_utm30lx.urdf.xacro)
130                         self.zn_X[j] = np.float32(np.min(self.ranges[(R[
    j][0:(len(R[j]))])]))
131 
132                         # Tracking variables
133 
134                         self.z_a_min[j] = np.min(np.int_(R[j][0:len(R[j
    ])]))
135 
136                         self.z_a_max[j] = np.max(np.int_(R[j][0:len(R[j
    ])]))
137 
138                         self.zn_min[j] = np.min(np.float32(self.ranges[
    np.int_(R[j][0:len(R[j])])]))
139 
140                         self.zn_max[j] = np.max(np.float32(self.ranges[
    np.int_(R[j][0:len(R[j])])]))
141 
142                     else:
143 
144                         self.z_aX[j] = 0
145 
146                         self.zn_X[j] = self.d
147 
148 
149                 self.z_aX=self.z_aX.astype(int)  #transform z_aX to an
    array of integers
150                 print'distance (2, 4, 3)= ',self.zn_X
151                 print'angle= (2, 4, 3)= ',self.z_aX
152                 Zval=[]
153                 self.zx=np.zeros([len(R)])
154                 self.zy=np.zeros([len(R)])
155                 self.z_values=[] #np.zeros([3*len(R)])
156 
157                 for i in range(0,n): #Loop to calculate x and y distance
     to each neighbour
```

```
158
159                     self.zx[i] = np.float32(np.cos((self.z_aX[i]-np.int_
    (180))*2*np.pi/360)*self.zn_X[i]) #2*np.pi/360: converts to radians
160
161                     self.zy[i] = np.float32(np.sin((self.z_aX[i]-np.int_
    (180))*2*np.pi/360)*self.zn_X[i])
162
163                     Zval.append([self.zn_X[i], self.zx[i], self.zy[i]])
164
165                 print ("data from Nexus 1 (2, 4, 3):")
166                 for i in range(0,n):  #Loop to print the x and y
    distances of detected agents
167
168                     print 'zx_[',i,'] = ', self.zx[i]
169                     print 'zy_[',i,'] = ', self.zy[i]
170                     print '--'
171
172                 print '--------------------'
173
174
175                 for i in range(0,n): #This loop puts all the values into
     the z_values matrix
176
177                     self.z_values= np.concatenate(Zval[:][:])
178
179                 self.z_values=np.asarray(self.z_values, dtype=np.float32
    )  #this line ensures that all numbers are type np_float (without it
    the publisher doesn't publish the proper values)
180
181  print 'Z values = ',self.z_values
182
183
184                 # publish z_values to send to Controller Node
185                 self.pub.publish(self.z_values)
186
187
188     def shutdown(self):
189         rospy.loginfo("Stopping dataprocessingnode_1")
190         self.running = False
191         self.z_values = np.array([self.d, self.d, 0, \
192                                   self.dd, self.dd, 0, \
193                                   self.d, self.d, 0], dtype=np.float32)
194         self.pub.publish(self.z_values)
195         print ('Shutting Down')
196  rospy.sleep(1)
197
198 if __name__ == '__main__':
```

```
199     rospy.init_node('dataprocessingnode_1', anonymous=False)
200     determine_z_values()
201     rospy.spin()
```

Listing 1: Data processing node of robot 1

```
#!/usr/bin/env python

#Import important packages
from __future__ import division
from math import *
import rospy
import matplotlib.pyplot as pl
import numpy as np
from rospy_tutorials.msg import Floats
from rospy.numpy_msg import numpy_msg
from geometry_msgs.msg import Twist

# Number of robots = 4
# They are linked to each other in the following way
#   Robot $ 1 $ has a link with Robots $ 2 $, $ 3 $, and $ 4 $
#   Robot $ 2 $ has a link with Robots $ 1 $, and $ 4 $
#   Robot $ 3 $ has a link with Robots $ 1 $, and $ 4 $
#   Robot $ 4 $ has a link with Robots $ 1 $, $ 2 $, and $ 3 $

######################################################################
# Dimensions
N = 4                               # Number of circular robots
M = 5                               # Number of edges
d = 2                               # Ambiance dimension
r = 1                               # Radius of each robot

######################################################################
# Desired formation shape
# Rectangle shape
a = 2.5                               # a = Width
b = 5                               # b = Height
                                    # Positioned in clockwise direction

#  Robot positions BEFORE translation + rotation
p1r = np.array([[0], [0]])
p2r = np.array([[a], [0]])
p3r = np.array([[0], [b]])
p4r = np.array([[a], [b]])

# Translate + rotate the desired formation shape
#TVector = np.array([[2], [2]])  # Translation vector
#RotAngle = 30                    # Rotation angle
#RotMatrix = np.array([[cos(degrees(RotAngle)), -sin(degrees(RotAngle))
   ], [degrees(sin(RotAngle)), cos(degrees(RotAngle))]])
                                    # Rotational matrix

# Robot positions AFTER translation + rotation
```

```
#p1d = np.dot(RotMatrix, p1r) + TVector
#p2d = np.dot(RotMatrix, p2r) + TVector
#p3d = np.dot(RotMatrix, p3r) + TVector
#p4d = np.dot(RotMatrix, p4r) + TVector

# Stacked position vector
#pd = np.array([[p1d], [p2d], [p3d], [p4d]])

# Relative state variables
# Relative position
z12d = p2r - p1r
z13d = p3r - p1r
z14d = p4r - p1r
z24d = p4r - p2r
z34d = p4r - p3r

# Relative distance
d12d = np.linalg.norm(z12d)
d13d = np.linalg.norm(z13d)
d14d = np.linalg.norm(z14d)
d24d = np.linalg.norm(z24d)
d34d = np.linalg.norm(z34d)

# Cosine of internal angle (Article Formula (5))
ctheta12d = 1 - 2 * (r / d12d)**2
ctheta13d = 1 - 2 * (r / d13d)**2
ctheta14d = 1 - 2 * (r / d14d)**2
ctheta24d = 1 - 2 * (r / d24d)**2
ctheta34d = 1 - 2 * (r / d34d)**2

# Stacked cosine vector
cthetad = np.array([[ctheta12d], [ctheta13d], [ctheta14d], [ctheta24d],
    [ctheta34d]])

# Desired Internal angle
theta12d = np.arccos(ctheta12d)
theta13d = np.arccos(ctheta13d)
theta14d = np.arccos(ctheta14d)
theta24d = np.arccos(ctheta24d)
theta34d = np.arccos(ctheta34d)

# Stacked internal vector
thetad = np.array([[theta12d], [theta13d], [theta14d], [theta24d], [
    theta34d]])

# Fixed constants / Interval for the error vector (e = ctheta - cthetad)
cd = cthetad - 0.5
```

```
92 fd = 1 - cthetad
93
94 # Minimum value
95 bb = np.minimum(0.5 * cd, fd)
96
97 ################################################################
98 class controller:
99     ''' The controller uses the interagent distances to determine the
       desired velocity of the Nexus '''
100
101     def __init__(self):
102         ''' Initiate self and subscribe to /z_values topic '''
103         # controller variables
104         self.running = np.float32(1)
105
106         self.U_old = np.array([0, 0])
107         self.U_oldd = np.array([0, 0])
108
109         # Motion parameters
110
111         # prepare Log arrays
112         self.einndis12_log = np.array([])
113         self.einndis13_log = np.array([])
114         self.einndis14_log = np.array([])
115         self.Un = np.float32([])
116         self.U_log = np.array([])
117         self.time = np.float64([])
118         self.time_log = np.array([])
119         self.now = np.float64([rospy.get_time()])
120         self.begin = np.float64([rospy.get_time()])
121         self.k = 0
122
123         # prepare shutdown
124         rospy.on_shutdown(self.shutdown)
125
126         # prepare publisher
127         self.pub = rospy.Publisher('/n_1/cmd_vel', Twist, queue_size=1)
128         self.velocity = Twist()
129
130         # subscribe to z_values topic
131         rospy.Subscriber('/n_1/z_values', numpy_msg(Floats), self.
       controller)
132
133         # subscribe to controller_variables
134         rospy.Subscriber('/controller_variables', numpy_msg(Floats),
       self.update_controller_variables)
135
```

```
136     def update_controller_variables(self, data):
137         ''' Update controller variables '''
138         if self.running < 10:
139             # Assign data
140             self.controller_variables = data.data
141 
142             # Safe variables
143             self.running = np.float32(self.controller_variables[0])
144 
145 
146     def controller(self, data):
147         ''' Calculate U based on z_values and save error velocity in log
    arrays '''
148         if self.running < 10:
149             # Input for controller
150             z_values= data.data
151             Ktheta = 500
152 
153             #get z_values
154             z12 = np.array([[z_values[1]], [z_values[2]]])
155             z13 = np.array([[z_values[7]], [z_values[8]]])
156             z14 = np.array([[z_values[4]], [z_values[5]]])
157 
158             # Relative distance
159             d12 = z_values[0]
160             d13 = z_values[6]
161             d14 = z_values[3]
162 
163             print "Relative distances = ",d12,d13,d14
164             print "Desired distances = ",d12d,d13d,d14d
165 
166             # Decompose cd
167             # Lower bound for error
168             c12d = cd[0]
169             c13d = cd[1]
170             c14d = cd[2]
171 
172             # Decompose cthetad
173             # Desired cosine values
174             ctheta12d = cthetad[0]
175             ctheta13d = cthetad[1]
176             ctheta14d = cthetad[2]
177 
178             # Cosine of internal angle
179             ctheta12 = 1 - 2 * (r / d12)**2
180             ctheta13 = 1 - 2 * (r / d13)**2
181             ctheta14 = 1 - 2 * (r / d14)**2
```

```
182
183             theta12 = np.arccos(ctheta12)
184             theta13 = np.arccos(ctheta13)
185             theta14 = np.arccos(ctheta14)
186
187             print 'Desired angle = ',theta12d,theta13d,theta14d
188             print 'Actual angle = ',theta12,theta13,theta14
189
190             # Sine of internal angle
191             stheta12 = np.sin(theta12)
192             stheta13 = np.sin(theta13)
193             stheta14 = np.sin(theta14)
194
195             # Error in cosine value
196             etheta12 = ctheta12 - ctheta12d
197             etheta13 = ctheta13 - ctheta13d
198             etheta14 = ctheta14 - ctheta14d
199
200             # Inner distance error
201             einndis12 = d12 - d12d
202             einndis13 = d13 - d13d
203             einndis14 = d14 - d14d
204
205             #Control law
206             v12 = r * ((etheta12 * c12d) / ((etheta12 + c12d)**3))
207             v13 = r * ((etheta13 * c13d) / ((etheta13 + c13d)**3))
208             v14 = r * ((etheta14 * c14d) / ((etheta14 + c14d)**3))
209
210             sa12 = 1 / (2*r) * stheta12
211             sa13 = 1 / (2*r) * stheta13
212             sa14 = 1 / (2*r) * stheta14
213
214             gplus12 = 2 * sa12 * z12
215             gplus13 = 2 * sa13 * z13
216             gplus14 = 2 * sa14 * z14
217
218             mgplus12 = 4 * (sa12)**2 * d12**2
219             mgplus13 = 4 * (sa13)**2 * d13**2
220             mgplus14 = 4 * (sa14)**2 * d14**2
221
222             parV12 = -2 * v12 * (1-ctheta12) * stheta12 * ((gplus12)/(
    mgplus12))
223             parV13 = -2 * v13 * (1-ctheta13) * stheta13 * ((gplus13)/(
    mgplus13))
224             parV14 = -2 * v14 * (1-ctheta14) * stheta14 * ((gplus14)/(
    mgplus14))
225
```

```
226             #tparV12 = -v12 * ((4*r**2)/(d12**4))*z12
227             #tparV13 = -v13 * ((4*r**2)/(d13**4))*z13
228             #tparV14 = -v14 * ((4*r**2)/(d14**4))*z14
229 
230             # Individual control law
231             U = - Ktheta * parV12 - Ktheta * parV13 - Ktheta * parV14
232 
233             print "U robot1= ", U
234 
235             # Saturation
236             v_max = 0.2
237             v_min = 0.002
238             for i in range(len(U)):
239                 if U[i] > v_max:
240                     U[i] = v_max
241                 elif U[i] < -v_max:
242                     U[i] = -v_max
243                 elif -v_min < U[i]+self.U_old[i]+self.U_oldd[i] < v_min
    : # preventing shaking
244                     U[i] = 0
245 
246             # Set old U values in order to prevent shaking
247             self.U_oldd = self.U_old
248             self.U_old = U
249 
250 
251             # Append error and velocity in Log arrays
252             self.einndis12_log = np.append(self.einndis12_log, einndis12
    )
253             self.einndis13_log = np.append(self.einndis13_log, einndis13
    )
254             self.einndis14_log = np.append(self.einndis14_log, einndis14
    )
255             self.Un = np.float32([np.sqrt(np.square(U[0])+np.square(U
    [1]))])
256             self.U_log = np.append(self.U_log, self.Un)
257 
258             # Save current time in time log array
259             if self.k < 1:
260                 self.begin = np.float64([rospy.get_time()])
261                 self.k = 10
262             self.now = np.float64([rospy.get_time()])
263             self.time = np.float64([self.now-self.begin])
264             self.time_log = np.append(self.time_log, self.time)
265 
266             # publish
267             self.publish_control_inputs(U[0], U[1])
```

```
268
269         elif 10 < self.running < 1000:
270             self.shutdown()
271
272     def publish_control_inputs(self,x,y):
273         ''' Publish the control inputs to command velocities
274
275             NOTE: somehow the y direction has been reversed from the
    indigo-version from Johan and
276             the regular kinetic version. Hence the minus sign.
277
278         '''
279
280         self.velocity.linear.x = x
281         self.velocity.linear.y = y
282
283         # print 'cmd_vel NEXUS 1 (x,y)', self.velocity.linear.x, self.
    velocity.linear.y
284         #         rospy.loginfo(self.velocity)
285
286         self.pub.publish(self.velocity)
287
288     def shutdown(self):
289         ''' Stop the robot when shutting down the controller_1 node '''
290         rospy.loginfo("Stopping Nexus_1...")
291         self.running = np.float32(10000)
292         self.velocity = Twist()
293         self.pub.publish(self.velocity)
294
295 #         np.save('/home/s2036975/Documents/Master Thesis/experiments/
    experiment_x/E1_log_nx1', self.E1_log)
296 #         np.save('/home/s2036975/Documents/Master Thesis/experiments/
    experiment_x/E4_log_nx1', self.E4_log)
297 #         np.save('/home/s2036975/Documents/Master Thesis/experiments/
    experiment_x/U_log_nx1', self.U_log)
298 #         np.save('/home/s2036975/Documents/Master Thesis/experiments/
    experiment_x/time_log_nx1', self.time_log)
299
300         rospy.sleep(1)
301
302
303         pl.close("all")
304         pl.figure(0)
305         pl.title("Inter-agent distance error measured by Nexus 1")
306         pl.plot(self.time_log, self.einndis12_log, label="e1_nx2", color
    ='b')
307         pl.plot(self.time_log, self.einndis13_log, label="e1_nx3", color
```

```
    ='y')
308         pl.plot(self.time_log, self.einndis14_log, label="e1_nx4", color
    ='r')
309         pl.xlabel("Time [s]")
310         pl.ylabel("Error [m]")
311         pl.grid()
312         pl.legend()
313 
314         pl.figure(1)
315         pl.title("Input velocity Nexus 1 ")
316         pl.plot(self.time_log, self.U_log, label="pdot_nx1", color='b')
317         pl.xlabel("Time [s]")
318         pl.ylabel("Velocity [m/s]")
319         pl.grid()
320         pl.legend()
321 
322         pl.pause(0)
323 
324 
325 
326 if __name__ == '__main__':
327     try:
328         rospy.init_node('controller_1', anonymous=False)
329         controller()
330         rospy.spin()
331     except:
332         rospy.loginfo("Controller node_1 terminated.")
```

Listing 2: Controller 1