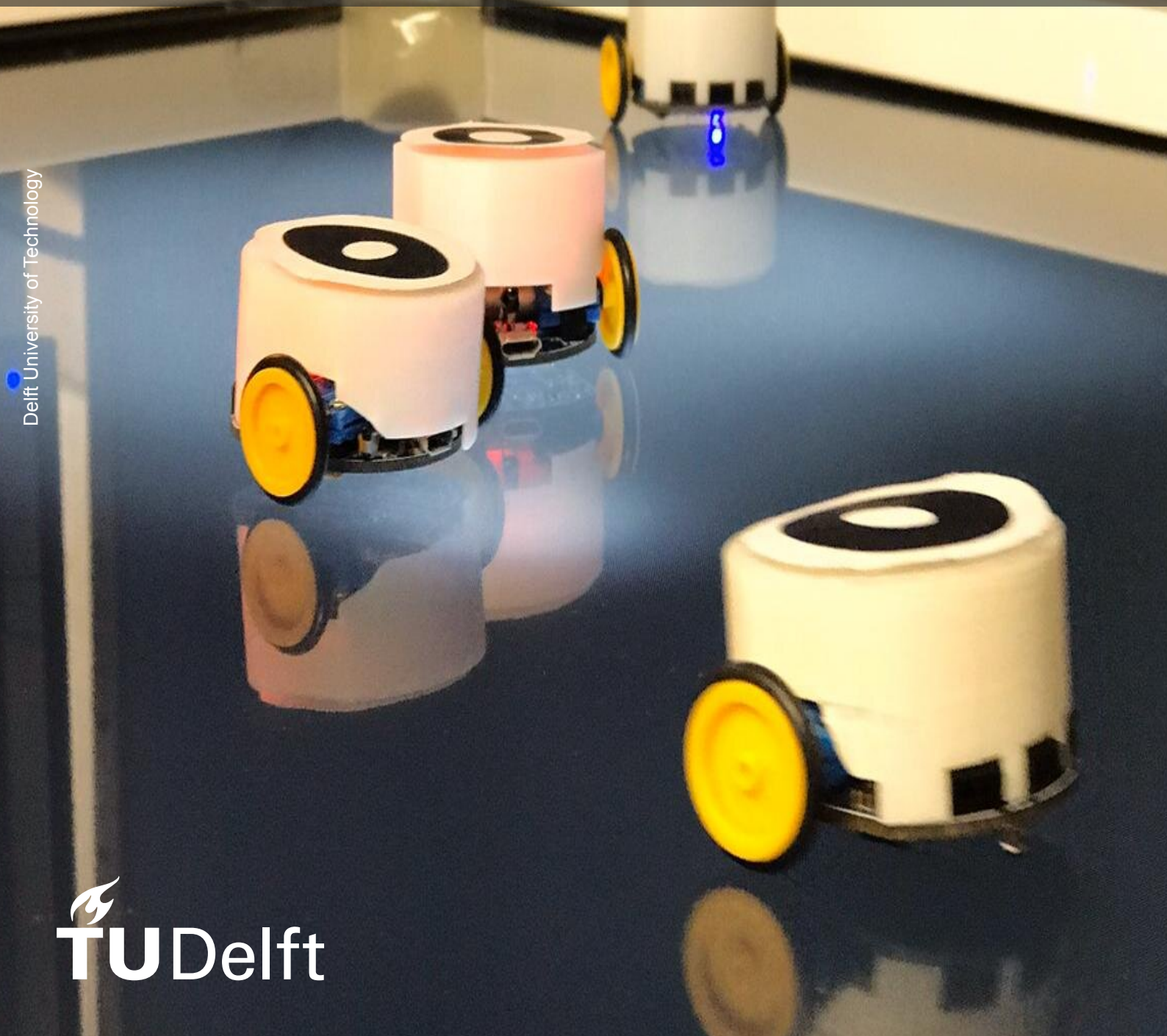
Swarming

A Control Algorithm for Swarming Autonomous Robots

EE3L11: Bachelor Graduation Project

Ivar Hendrikson &
Thijmen Hoenderboom



Delft University of Technology

TU Delft

Swarming

A Control Algorithm for Swarming Autonomous Robots

by

Ivar Hendrikson &
Thijmen Hoenderboom

Student Name	Student Number
Ivar	4593227
Thijmen	4926153

Instructor: Dr. RangaRao Venkatesha Prasad
Teaching Assistant: Ashutosh Simha
Project Duration: April, 2022 - June, 2022
Faculty: Faculty of Electrical Engineering, Mathematics and Computer Science, Delft

Cover: The Colias-V robots: A system for swarm robotics applications inspired by pheromone communication in insects by Ingrid Fadelli
Style: TU Delft Report Style, with modifications by Daan Zwaneveld

TU Delft

Preface

This thesis is part of the Bachelor Graduation Project for the Bachelor of Sciences of Electrical Engineering at the Technical University of Delft. This paper contains only one out of the three parts that, together, form the entire research scope: The swarming of Robots. We would like to thank our daily supervisors dr. ir. Ashutosh Simha and ir. Suryansh Sharma as well as our head supervisor prof. Venkatesha Prasad for all the guidance and help that they were kind enough to give us. Furthermore we want to thank dr. ir. Chris Verhoeven, for the insights and guidance that he gave us during the Green Light Assesment, as well as for the open invitation to approach him with our ideas. Finally, we would like to thank our colleagues, Melle Minten, Sven Dukkers, Jeroen van Uffelen and Lars de Kroon for their dedication as well as the enjoyable and productive working environment which they created.

Ivar Hendrikson &
Thijmen Hoenderboom
Delft, July 2022

Abstract

A swarm at its most basic description consists of multiple individuals all performing actions based on their immediate surroundings. The challenge for robotics is creating a control algorithm that processes these surroundings and takes appropriate actions as a consequence. The focus of this document is the bird flocking algorithm. For this reason, the main focus points for the control algorithm will be on heading alignment, neighbour distance maintenance & collision avoidance.

Contents

1 Introduction

1.1. Swarming Behaviour

Swarming is a phenomenon widely seen in nature. It occurs when a lot of smaller entities make decisions based on their immediate neighbourhood, without receiving a centralised directive. The behaviour of the individual is simple, but when a lot of these individuals are put together, complex behaviour can form. An example of this swarming behaviour in nature is the flocking of birds [1]. These birds use flocking behaviour, which is a type of swarming, to avoid predators, navigate, and even gain aerodynamic benefits.
This iconic feature of swarming: complex behaviour out of simple instructions, is a research topic of interest. This is of such interest, that a lot of research is dedicated to mimicking it with drones or vehicles. But this is not for no reason. A lot of technological applications are arising that swarming may be useful in [2]. One of the most famous researches within this field is the boids bug, created by Craig Reynolds in 1986. He simulated swarm behaviour following only 3 basic rules [3]. These rules are as follows:

1. Move within an equivalent direction as your neighbours
2. Remain close to your neighbours
3. Avoid collisions with your neighbours

Many subsequent and current models use variations on these rules, often implementing them through concentric ”zones” around each animal. This is elaborated more in Section 1.4.1.

1.2. Goal of the Project

As mentioned before, the subject of swarming has become a hot topic. For this reason, we aim to create a working, scalable swarming algorithm. Our final design should be able to show swarming behaviour, using the cars and budget that were provided to us. The complete list of requirements are described in Chapter 2.

1.3. Problem Definition

Creating a ‘functioning swarm requires research. To perform this more efficiently, the project is split up into 3 parts. These are as followed:

- Localization
- Communication
- Control

1.3.1. Localization

The first part of the project is localization. Localization aims to locate the members of the swarm, with an additional goal to also locate obstacles in the vicinity of the swarm. Using the location of other

members, individual members should coherently be able to decide in which way they can and cannot move. Localization does not only aim to just locate other members of the swarm but does so without the usage of external measuring instruments such as beacons or a central radar system. Such a localization system is called decentralised.

1.3.2. Communication

The second part is communication. Communication is responsible for the exchange of information between swarm members. In larger swarms, it is not necessary, and can even be disadvantageous, for each member to communicate with every other member in the swarm. Often it is only necessary to keep track, and therefore communicate with the neighbouring members. Consequently, communication also aims to identify which members in the swarm are neighbouring the current member and use this information to only communicate with these neighbouring swarm members. It lastly aims to build a platform to have wireless logging of the swarm network to observe the behaviour of the swarm which will be useful for observational purposes and debugging.

1.3.3. Control

The third part of the project is control. This is also the focus of this thesis. The control subgroup designs a control algorithm for the individual swarm members. This algorithm functions on information that is provided by the communication- and localization sub-parts. It aims to design the robots and their controls for each swarm member to manoeuvre through a two-dimensional space. It furthermore aims to follow a simplified version of the boids algorithm [3]. Where each member tries to stay a certain distance away from their respective neighbours while trying to reach a globally similar heading. The robots should make all decisions autonomously.

1.4. State of the Art Analysis

1.4.1. Swarming Algorithms

In the field of robot swarming a lot of research is being done. For this reason, to allow multiple robots to stay in a swarm, different approaches can be taken. The explored options are potentials, vector fields and virtual physical objects like springs. First is (social) potentials, this implies that every object has a certain attraction or repulsion potential to a robot. This can be done for example linearly [4][5] or using a Gaussian kernel [6]. All these potentials are then summed up to a force or a velocity. The second approach is similar to potentials, except that they are not created on objects, but portrayed as a field throughout the space [7][8]. This is similar to the potential method but expands it throughout the near-space instead of only attaching it to certain objects. The last approach is mimicking natural physical principles and applying them to the robots. A virtual physical object or principle is created to maintain formation. This can be done using different kinds of physical objects and principles like visco-elasticity [9], Newtonian physics [10] or spring systems [11][12].

1.4.2. Obstacle avoidance

Once the swarm can maintain an orientation, it should be able to travel and avoid possible obstacles. In fact, this principle is already implemented by another method to avoid robots in the swarm. By applying the same algorithm to an obstacle, but using only repulsion and not attraction, the swarm is able to avoid detected obstacles. This can be done for all cases explained in Section 1.4.1. When using the virtual spring method, a spring can be fixated to the closest point of the obstacle to a robot [12]. When using a vector field, repulsive and evasive fields can be created surrounding the object [8]. The object avoidance can be extended by avoiding larger obstacles by following along the object wall [5].

1.5. Structure of Thesis

The thesis will be structured as follows. In Chapter 2, the requirements for the final goal will be discussed. The following Chapter 3 & Chapter 4 will describe the process of designing a working swarming algorithm, by first starting in a simple 1-dimensional plane and then expanding this to 2 dimensions. Finally, Chapter 5 will discuss the final results of our work, as well as discuss what can be done in future works.

2

Programme of Requirements

The requirements that are to be met to reach a successful project are described in this chapter. The goal of the project is to create a working swarm.

2.1. Requirements

2.1.1. Mandatory Requirements

Functional requirements

1. The robots must perform swarming behaviour.
2. The robots/swarm must autonomously avoid obstacles.
3. The robots must be able to sense their surroundings.
4. The robots must be able to measure their own heading.

Non-Functional Requirements

1. The swarm must function indoors.
2. The swarm should consist of at least 6 agents.
3. The total cost is less than €150.
4. The swarm should be scalable.

2.1.2. Trade-off requirements

1. The robot's sensors should have a high frequency of measurements.
2. The robots should measure obstacles and other robots with an accuracy of 30 cm.
3. The robots should have a detection range of at least one meter.
4. The robot should maintain a distance of 20 to 30 cm from other robots.

2.1.3. Assumptions

At the start of this project, some hardware choices were made available by our supervisors. Multiple different forms of chassis, as well as different micro-controller options. This led to the following assumptions:

1. The robots used for the swarms exist out of given hardware.
2. The robots are non-holonomic.
3. The swarm must work in 2 dimensions.
4. The provided localization only provides a distance to an object in a certain direction, but not what this object is.
5. Communication only provides the heading and speed of robots within a defined region.

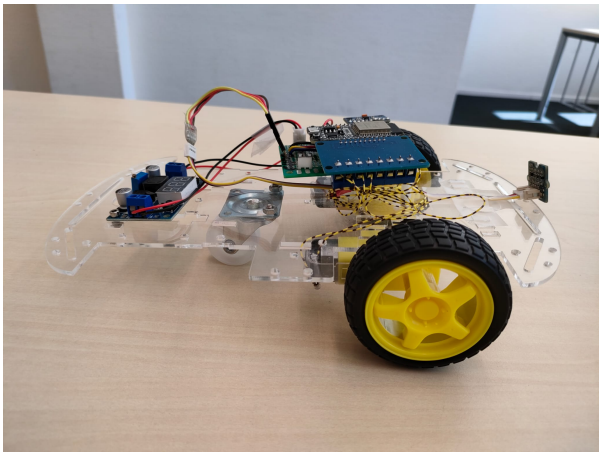


Figure 2.1: Vehicle intended for swarming

2.2. Hardware

2.2.1. Robots

This section will discuss the hardware used in this project. The ”Autonomous robots” that are used, are non-holonomic, 3-wheeled vehicles. They are outfitted with an ESP-WROOM 32 development board, as well as a motor control chip. 2 of the wheels are driven by a PWM signal, generated by the ESP. Whilst the third wheel is an unpowered castor wheel. The vehicle is shown in figure 2.1.

2.2.2. ESP-32 WROOM

The ESP-32 WROOM development board functions as the main decision-making unit of the robot. It is a low-cost, low-power system on a chip micro-controller. The chip is capable of WiFi communications, which allows the robot on which is outfitted to communicate with other nearby robots. Furthermore, the board is outfitted with an Inter-IC-bus (I^2C). This feature allows other integrated circuits to communicate with the micro-controller. This is necessary for the required magnetometer in the form of an IMU. The I^2C bus is also used by a Time of Flight (ToF) sensor, but this is not directly of importance for this thesis.

2.2.3. Inertial Measurement Unit

The inertial measurement unit consists of an integrated gyroscope, accelerometer and magnetometer. The magnetometer is most of interest, as the magnetometer determines the heading of the robot using the earth’s magnetic field. This is a feature necessary for swarming. The robot is equipped with the MPU-9250. This IMU has a full-scale range of $\pm 4800 \mu T$. Magnetometer measurement values can be retrieved at a frequency of 200 Hz using this chip.

2.3. Software

2.3.1. Espressif

The Espressif framework is used in building, flashing and monitoring the code on an ESP chip. ESP-IDF is a tool written in python that comes with Espressif. It allows the user to interact with and flash instructions to build its program on an ESP chip.

2.3.2. Clion

ESP-IDF flashes C code onto the ESP chip. The C code is written in Clion. Using a Cmake framework, independent modules are created.

2.3.3. PyCharm

PyCharm for Python is the main tool in which simulations are made. It also functions as a platform from which code can be run to communicate with the robots using WiFi.

3

1-Dimensional control

The first step in fully controlling the robot is to control it in one dimension. This is done by characterizing the robot and modelling it using physical measurements. To control the speed of the robot, a PID controller is used. This controller is then rated according to the desired design parameters. The rating is used to optimize the behaviour of the robot. Finally, the results are measured to ensure that the robot is properly controlled.

3.1. Speed control

To control the robot, its speed needs to be adjustable by the output of a micro-controller. The robot used uses two simple DC motors. The turning speed of this motor can be controlled with the use of Pulse Width Modulation (PWM). By turning the motor on and off for a certain percentage of time during one cycle, called the duty cycle, the motor changes speed. The micro-controller is able to output such a PWM signal. This signal is then fed to the motor driver in order to control the motor and in turn, the speed of the robot.

3.2. Robot measurements

The robot used is described in section (2.2.1). Since it is a simple robot with low mass and a relatively low maximum speed, a simple linear model suffices. This model was created by driving the motors at a certain duty cycle for a predetermined amount of time. The total distance driven was then used to calculate the speed, at which the robot could drive for that specific duty cycle. From these measurements, a linear estimate was created as the final model. This resulted in the following linear model compared to the obtained measurements obtained:

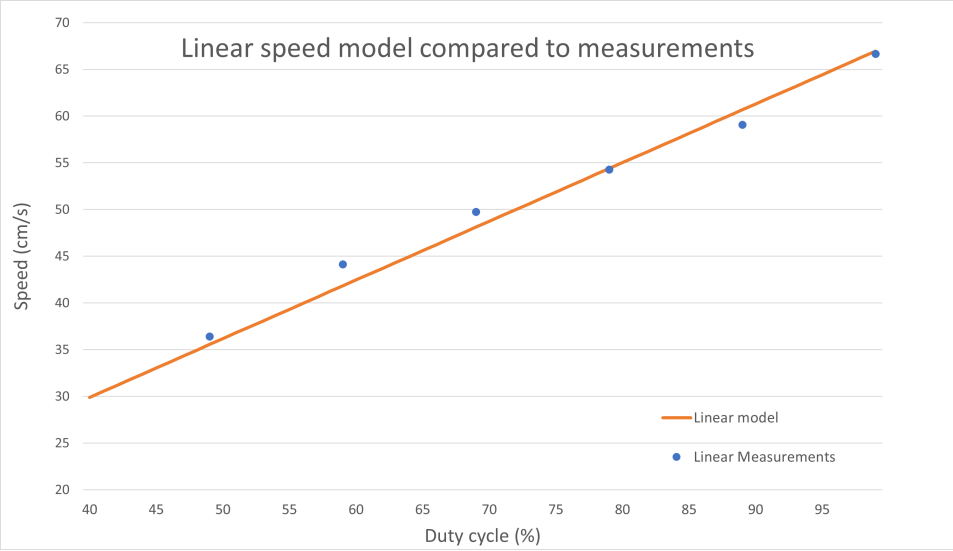


Figure 3.1: Relation between speed and duty cycle modelled and measured

As can be seen from the figure above, the robot does not move with a duty cycle of less than 40%. This is because the robot's static friction is higher than the force generated by the motors driven by the PWM signal at 40% duty or less. From 40% to the maximum 100% the relation between duty (D) and speed (v_{car}) is approximately linear in the form:

$$v_{robot} = a \cdot D + b \tag{3.1}$$

In the above equation, a and b are constants determined per robot. This speed is used to determine the position of the robot in between localization measurements. This way, the position (x) at an infinitesimally small time window dt can be determined.

$$x(T) = \sum_{t=0}^{T} v_{robot}(t) \cdot dt \tag{3.2}$$

3.3. Controller design

In order to make sure that the robot drives to the desired location accurately a controller needs to be designed. The controller receives the difference between the robot's current location and its' desired location (distance error). The controller returns an output proportional to this distance error, its derivative and its' integral. This is a PID controller, which results in the general output:

$$u(t) = K_p \cdot e(t) + K_i \int_0^t e(\tau)d\tau + K_d \frac{de(t)}{dt} \tag{3.3}$$

In this equation, $e(t)$ is the distance error K_p, K_i and K_d are the gain, integral gain and derivative gain respectively. These gain parameters can be tweaked to change the behaviour of the controller. Since the error is not known continuously but discretely, Equation 3.3 needs to be changed. The integral error can be calculated by summing all the previous errors (positional) or by incrementally updating the controlling output (incremental). The incremental implementation is preferred due to a lower cumulative error. This results in the following controller output:

$$\begin{aligned} du(k) = Kp \cdot (e(k) - e(k-1)) + \\ K_i \cdot e(k) + \\ K_d \cdot (e(k) - 2 \cdot e(k-1) + e(k-2)) \\ u(k) = u(k-1) + du(k) \end{aligned} \tag{3.4}$$

Here, $u(k)$ is the current distance error. The controller is bounded to $\{-100, 100\}$ so that it can be directly used as duty cycle for the motors. Using the controller in it a simple form with K_p, K_i and K_d all equal to 1, the following results are obtained:

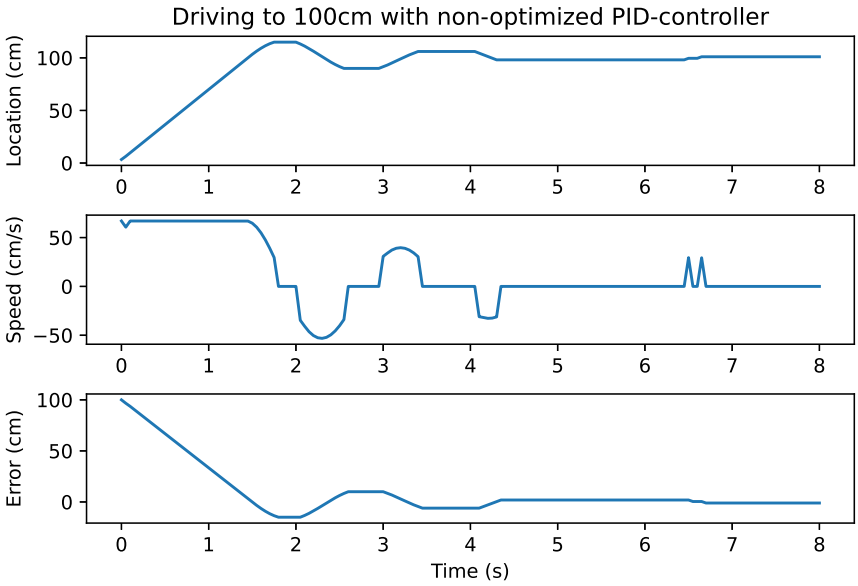


Figure 3.2: Robot behaviour controlled by a non-optimized PID-controller

3.4. Controller optimization

As can be seen from Figure 3.2, the behaviour of the robot and controller is not desirable when the controller is not optimized. There are a few things that can make the controller function better.

The localization constraint is that the minimal refresh period is $200ms$. However, since the location is known, with certain accuracy, according to Equation 3.2, the refresh rate of the controller can be much higher. The controller will then update inaccuracy errors once the real location is determined from localization. A higher refresh rate allows the controller to switch fast to an updated output so that the behaviour is more smooth. The same non-optimized controller as shown in Figure 3.2, which uses a refresh interval of $500ms$, is compared to the controller but with a refresh interval of $50ms$:

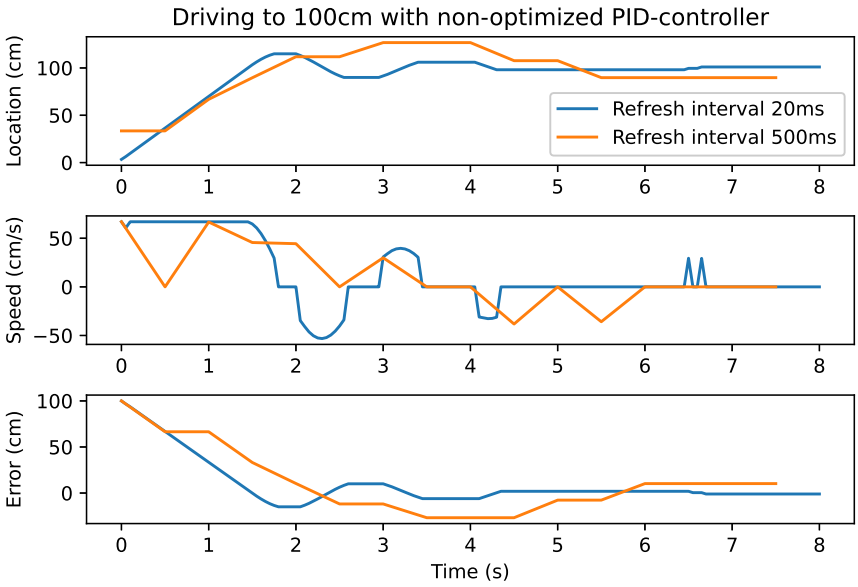


Figure 3.3: Robot driven by same controller, but with different refresh rate

As can be seen from the figure above, increasing the frequency of the controller smooths out the behaviour but it still leaves undesirable behaviour.

The second and more straightforward way to improve the controller is by tweaking the parameters. To choose what controllers to tweak, it is crucial to determine the desired behaviour. There are several parameters that can be prioritized. The following parameters, in order of importance, were chosen to define the desired controller:

1. *Steady-state error*: this is the error when the controller error has decreased enough for the controller output to no longer influence the speed. At this point, the controller is in *steady-state*. The distance error in steady-state should be as small as possible.
2. *Overshoot*: when error approaches zero the controller should slow down in order to smoothly achieve zero error. A non-optimized controller does not slow down enough and moves past zero error, once again increasing the error but in the other direction. In this case, it would mean the robot would drive past its' desired location and then move back and oscillate around the desired location for a small time. With a large overshoot, it could mean a robot would bump into another robot. Therefore the smallest possible overshoot is allowed for the controller.
3. *Settling time*: this is defined as the time for which the error does not exceed a certain error tolerance, in this case, $2,5\%$. Both the steady-state error and overshoot are important constraints, but they need to be achieved as fast as possible. For that reason, the settling time should be minimized as well, but not at the cost of either overshoot or steady-state error.

Using the above-mentioned design choices, the parameters can be tweaked. In order to achieve the maximally optimized controller, a fitness function is created. This fitness function gives a value to a specific controller. This value is based on the design choices, weighted according to their importance. So a larger steady-state error gives a much lower score than a larger settling time. By testing a large number of controllers and keeping track of their fitness function values, the best controller can be chosen. This results in the following controller:

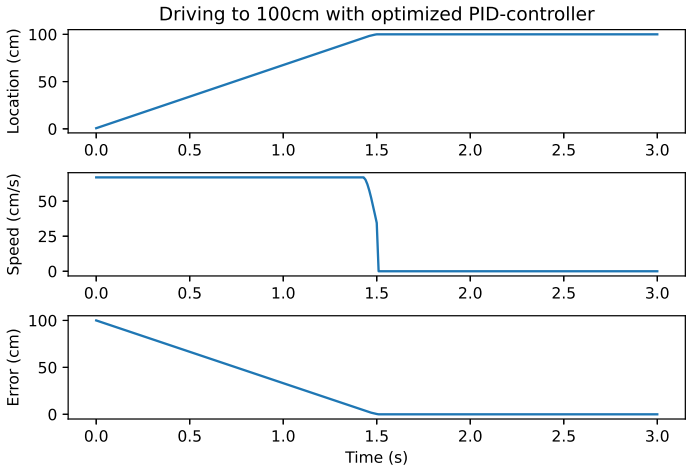


Figure 3.4: Robot driven by optimized controller

Table 3.1: Optimized controller parameters

Parameter	Value
K_p	28.1
K_i	$4,5$
K_d	3
Steady-state error	$322,8mm$
Overshoot	$21,0mm$
Settling time	$1,45s$

3.5. Results

The controller as shown in the figure above is tweaked to work at a frequency of $100Hz$ and driving to $100cm$. In order to test the controller, different distances need to be tested, which can be seen in Figure 3.5. The controller works properly except for a few small spikes after some time. However this is not a problem as the use of the robot is not to move to a stationary position and stop, but to swarm. This can be modelled by following a moving position instead of a stationary one. This is tested for 4 different speeds for the moving robot in Figure 3.6

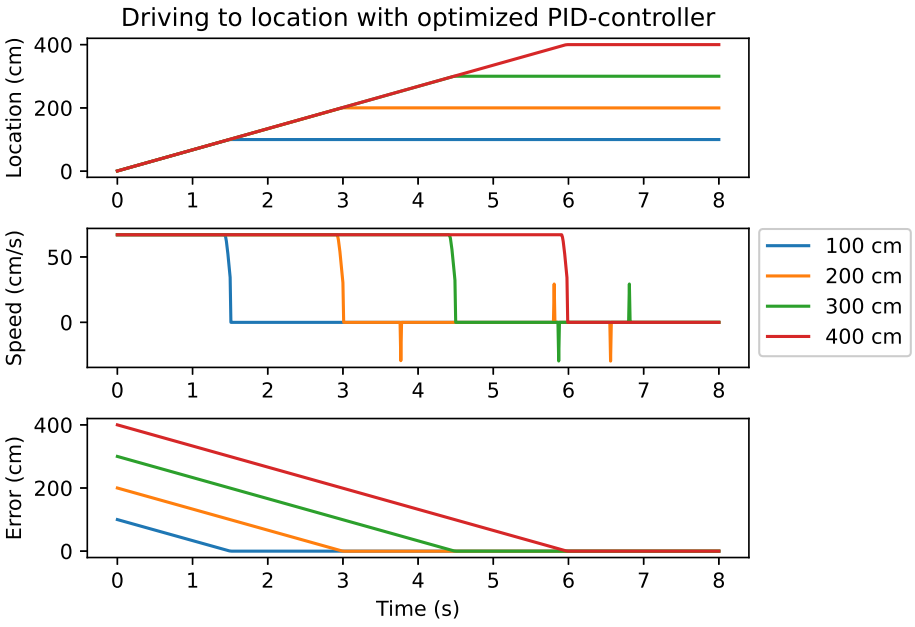


Figure 3.5: Robot driven by same controller, but with different refresh rate

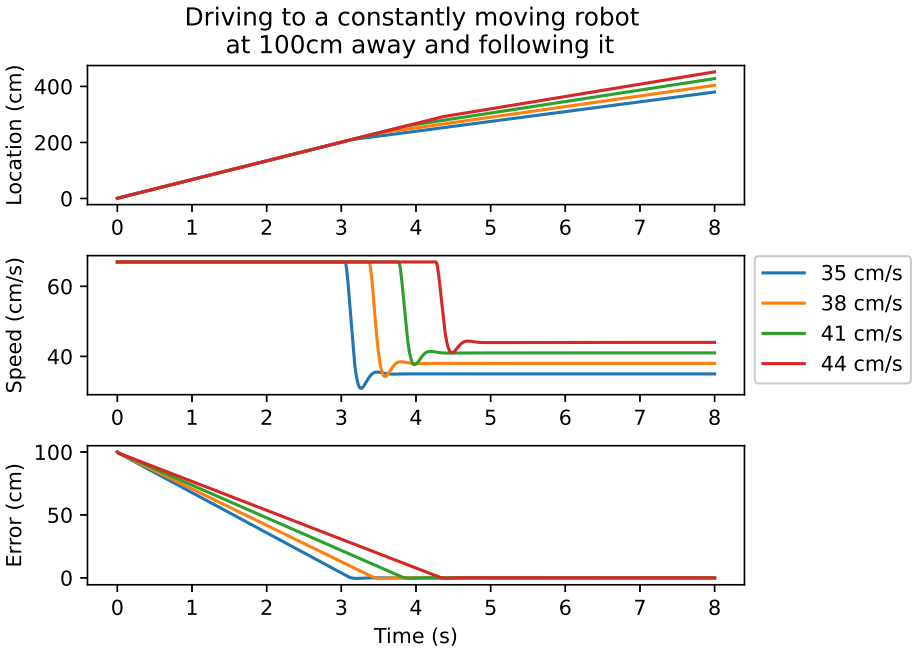


Figure 3.6: Robot driven by same controller, but with different refresh rate

4

2-Dimensional control

The robot is able to move to a certain location and follow a robot from a certain distance apart by moving to the desired location. For the robots to swarm they need to be able to move in at least 2 dimensions. Therefore the 1D model has to be expanded to 2D. The robot has 2 wheels to drive, as explained in Section 2.2.1 allowing it to rotate and drive forwards or backwards. The two DC motors driving the wheels need to be independently controlled. Similarly to Section 3.2, at first a model needs to be created. Then a controller, or in this case 2 controllers, is made to control the behaviour of the robot. This is then modelled tested and optimized according to the desired design choices.

4.1. Robot measurements

The robot's 1-dimensional speed behaviour has already been modelled in Section 3.2. This is under the assumption that the motors are rotating in the same direction at the same speed. For 2D control, the 2 motors will be driven at different speeds and directions. Similarly to the speed measurements, the robot is rotated at different speeds, driving the motors at the same speed in opposite direction. The heading of the robot is measured by a calibrated non-filtered IMU operating at $200Hz$ This results in the following plot:

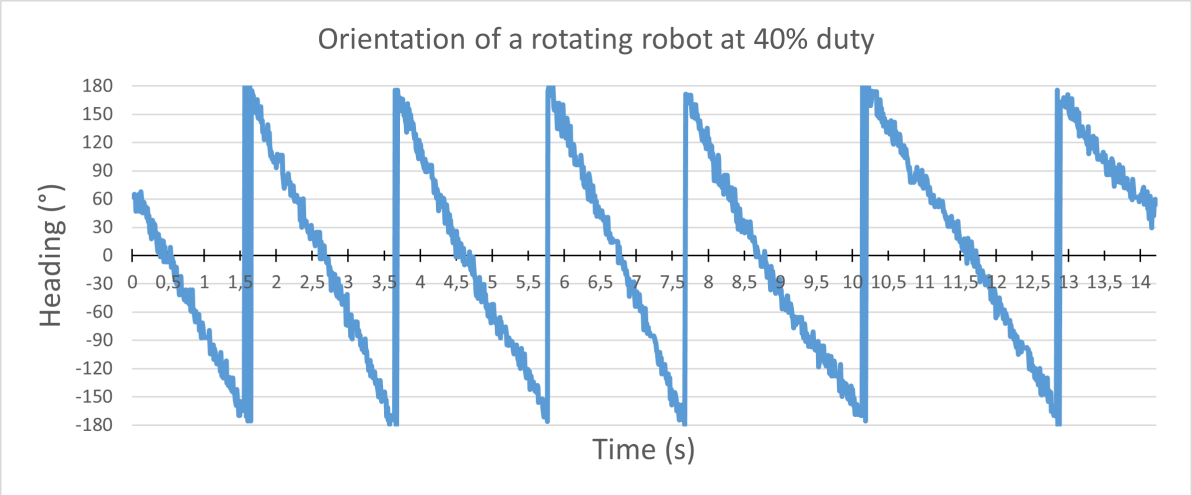


Figure 4.1: Orientation of a stationary rotation robot by a calibrated non-optimized IMU

From the figure above, the discrete time derivative is taken to measure the rotation speed of the robot:

$$\frac{\Delta\theta}{\Delta t}(k) = \dot{\theta}(k) = (\theta(k) - \theta(k+1)) \cdot f_{IMU} \tag{4.1}$$

A simple brick wall low pass filter is applied to remove the step between $180°$ and $-180°$:

$$\dot{\theta}(k) = \begin{cases} 0, & \text{if } \dot{\theta}(k) > 180 \\ \dot{\theta}(k) & \text{otherwise} \end{cases} \tag{4.2}$$

The remaining derivative values are averaged to determine the rotation speed per duty percentage(in $°/s$ per duty percentage) of a specific robot:

$$\theta_r = \frac{1}{n} \sum_{k=0}^{n} \dot{\theta}(k) \tag{4.3}$$

This average rotational speed is used to setup a linear relation based on the duty cycle:

$$\Delta\vec{o}_r = \theta_r \cdot \frac{D_{right-wheel} - D_{left-wheel}}{2} \cdot \Delta t = \omega_{robot} \cdot \Delta t \tag{4.4}$$

Here $\Delta\vec{o}_{robot}$ is the difference in orientation of the robot in degrees ($°$) based on the duty cycle of the PWM-signal used to control the left ($D_{left-wheel}$) and right($D_{right-wheel}$) motor. ω_{robot} is the robots rotational speed (in $°/s$) and θ_r is previously determined rotational speed per duty cycle percentage (in $°/s$ per duty percentage).

4.2. Robot modelling

In Section 3.2 a model is created to determine the speed of the robot based on the duty cycle. Equation 3.1 can be expanded to two different duty cycles:

$$|v_{robot}| = a \cdot \frac{|D_{left-wheel} + D_{right-wheel}|}{2} + b \tag{4.5}$$

Combining Equation 4.5 and 4.4 results in the following:

$$\begin{bmatrix} |v_{robot}| \\ \omega_{robot} \end{bmatrix} = \frac{1}{2} \cdot \begin{bmatrix} a & a \\ -\theta_{robot} & \theta_{robot} \end{bmatrix} \cdot \begin{bmatrix} D_{left-wheel} \\ D_{right-wheel} \end{bmatrix} + \begin{bmatrix} b \\ 0 \end{bmatrix} \tag{4.6}$$

This equation can be used to translate a desired speed and orientation into a duty cycle that can be use to directly control the two motors.

4.3. Orientation control

In order for the robots to swarm, they need to move in the same direction. This is done by taking the average heading of all the robots (r_i) within a range R:

$$\vec{o}_{robot,desired} = arctan\left(\frac{\sum_{r_i \in R} sin(\vec{o_i})}{\sum_{r_i \in R} cos(\vec{o_i})}\right) \tag{4.7}$$

When calculating this desired it is important to note that it is not as straightforward as taking the average of all angles. For example, the average angle between $359°$ and $1°$ is $180°$. Therefore the x and y projections of the orientation vectors ($cos(\vec{o})$ and $sin(\vec{o})$) are used. The resultant angle is that of the vector of the sum of the calculated projections.

Since the desired orientation is known, the orientation difference can be calculated using the current orientation:

$$\Delta\vec{o}_{robot} = \vec{o}_{robot,desired} - \vec{o}_{robot} \tag{4.8}$$

This orientation difference is then used as input for a PID controller. This is done because it is desirable for the robot to smoothly achieve its' desired heading without undesirable overshoot or oscillation. Using this, Equation 4.6 can written in the form:

$$\begin{bmatrix} D_{left-wheel} \\ D_{right-wheel} \end{bmatrix} = \begin{bmatrix} a & a \\ -\theta_{robot} & \theta_{robot} \end{bmatrix}^{-1} \times \left(2 \begin{bmatrix} |v_{robot}| \\ \omega_{robot} \end{bmatrix} - \begin{bmatrix} b \\ 0 \end{bmatrix}\right) \tag{4.9}$$

To model the actual movement, first the orientation is updated and then the robot is driven forward with the updated speed. This is not the perfect representation of what actually happens, but with a small enough time interval dt, the error is negligible. The robots position is updated as following:

$$\begin{aligned} \vec{o}_{robot}(k) &= \vec{o}_{robot}(k-1) + dt \cdot \omega_r \cdot \\ x_{robot}(k) &= x_{robot}(k-1) + dt \cdot v_{robot} \cdot cos(\vec{o}_{robot}(k)) \\ y_{robot}(k) &= y_{robot}(k-1) + dt \cdot v_{robot} \cdot sin(\vec{o}_{robot}(k)) \end{aligned} \tag{4.10}$$

In order to test the behaviour, the speed is set to a constant $20cm/s$. At first 2 robots are placed randomly with orientation $135°$ and $45°$ respectively. This is used to create a simple manually optimized PID controller by tweaking it to desirable behaviour:

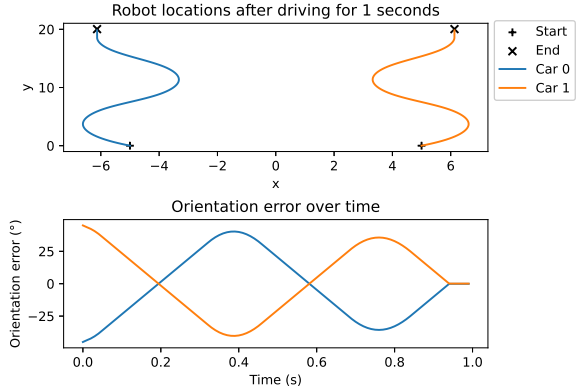


Figure 4.2: Orientation control without optimization

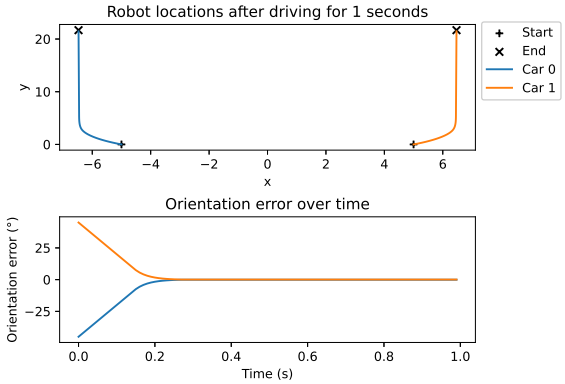


Figure 4.3: Orientation control with optimization

The manually optimized controller needs to be tested for different cases. In the first case two robots are placed along the same line with opposite orientation. In the second case several robots, in this case 6, are placed at random positions with random orientation. This way the robustness of the controller is tested:

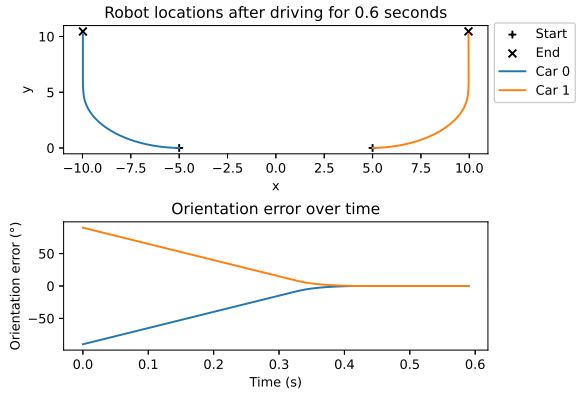


Figure 4.4: Controller test case 1: opposite orientation

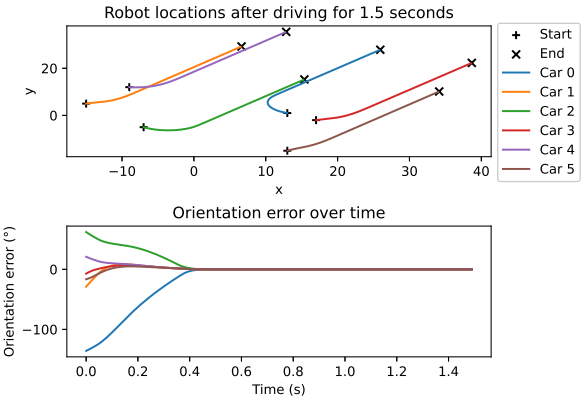


Figure 4.5: Controller test case 2: random position and orientation

4.4. Speed control

The second variable in the movement, besides the orientation, is the speed. The speed is calculated from Equation 4.5. For the swarm to stay together and not diverge, the desired behaviour is for them to all drive the same speed. This is done my averaging the speed of all robots (r_i) within a range R:

$$v_{robot,desired} = \frac{\sum_{r_i \in R} v_i}{\#(r_i \in R_0)} \tag{4.11}$$

Moreover, only averaging speed would eventually lead to a steady-state speed. However, this could mean the robot would not drive at all, or drive at maximum speed. This is undesirable because whenever a small error eventually and inevitably occurs, the robot would already drive at maximum speed meaning the error would increase at maximum as well. Therefore a speed is chosen for the robots to converge to. This speed ($v_{converge}$) would be fast enough to actually keep moving and be slow enough to still be able to correct for small errors when they occur. This is then adapted into Equation 4.12, resulting in the following:

$$v_{robot,desired} = \frac{\#(r_i \in R_0) \cdot v_{converge} + \sum_{r_i \in R} v_i}{2 \cdot \#(r_i \in R_0)} \tag{4.12}$$

As explained in Section 3.2, the robot is modelled using a simple linear model. Thus the change in speed is modelled as instantaneous. Therefore the speed of the robot does not need to be calculated through a controller, but can directly be used in Equation 4.4. This equation is not yet limited to the duty cycle of the robot, being $[-100, 100]$. To compensate for this the output is normalized to 100:

$$\begin{bmatrix} D_{left-wheel} \\ D_{right-wheel} \end{bmatrix} = \frac{\begin{bmatrix} D_{left-wheel} \\ D_{right-wheel} \end{bmatrix}}{\left|\begin{bmatrix} D_{left-wheel} \\ D_{right-wheel} \end{bmatrix}\right|} \cdot 100 \tag{4.13}$$

Combining everything results in the following behaviours:

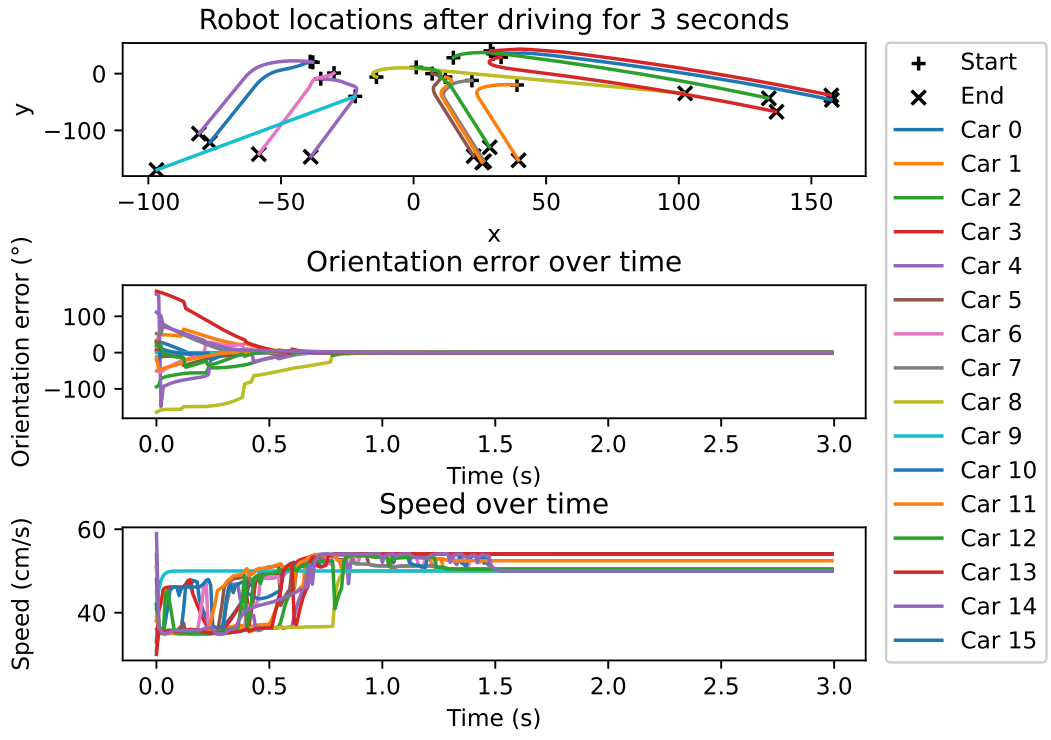


Figure 4.6: Swarming with speed and orientation control

4.5. Attraction and repulsion control

The second and third rule of flocking, as explained in Section 1.1, is to avoid collisions and remain close to the neighbours. This means that the robot needs to attract and repel its neighbours. This can be done in two ways: using orientation or using speed. Both follow an algorithm using different regions around the robot to indicate behaviour, similar to other swarming algorithms[4][6]. The differences are that an attraction region is also introduced. Also, the behaviour is not uniform throughout the circle. Three regions are defined: repulsion, neutral and attraction. These are radii of circles in order of size:

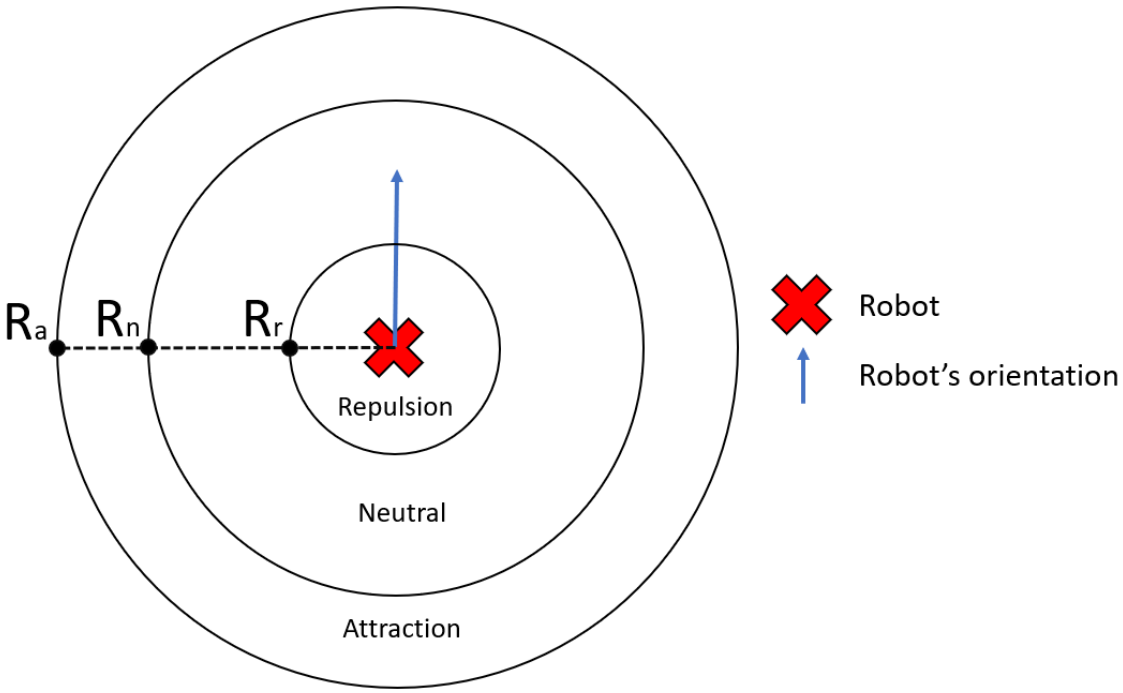


Figure 4.7: Virtual regions induced around the robot

The above picture is not to scale. In this figure, R_a, R_n and R_r are the radii of the attraction, neutral and repulsion regions respectively. The actual determining of the radii is an optimization process and depends on different factors like desired behaviour, robot size, desired swarm size etc.

4.5.1. Orientation based attraction and repulsion

The first way to induce attractive and repulsive behaviour is by changing orientation. In Section 4.3 orientation is controlled by taking the average orientation of all the surrounding robots in a region. When all robots are at this desired heading they will move parallel to each other and will not converge. To compensate for this, when a robot is in a certain region, the desired orientation will overshoot or undershoot the average heading in order to move closer or further away from another robot. This is done based on an overshoot or undershoot constant $d\theta$. Because every robot in the swarm has the same behaviour, the attraction will be performed on two sides. This means one heading will increase with $d\theta$ and another will decrease with $d\theta$. Therefore the total average heading will remain the same, and thus the heading does not need to be updated for surrounding robots. For two robots it will look as follows:

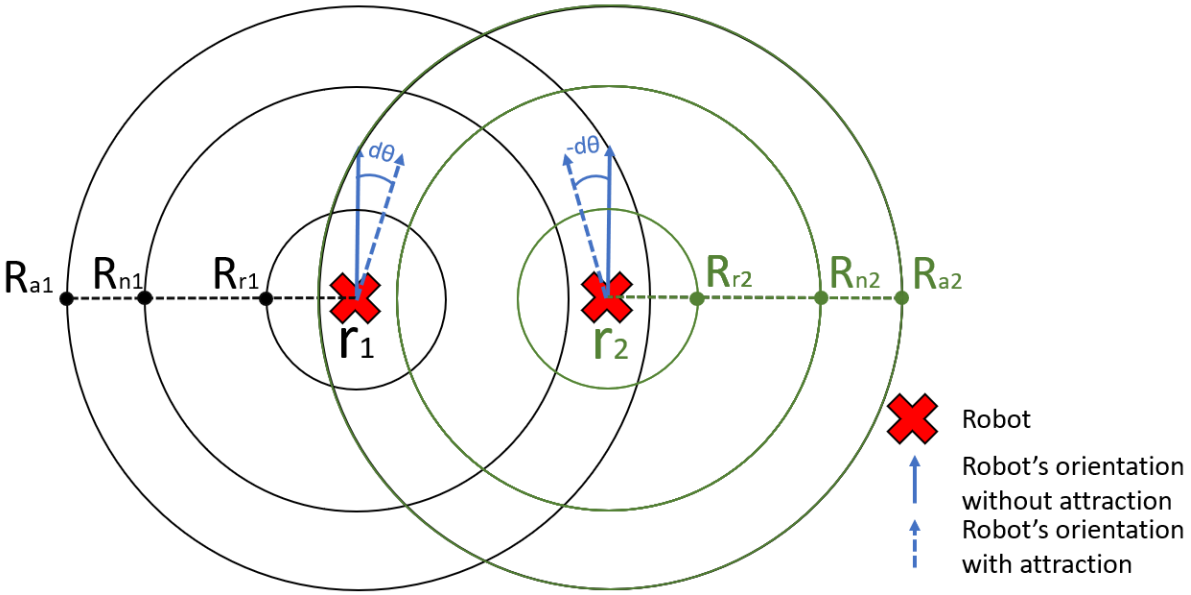


Figure 4.8: Attraction using orientation over- and undershooting

The same over- and undershooting can be used for repulsion:

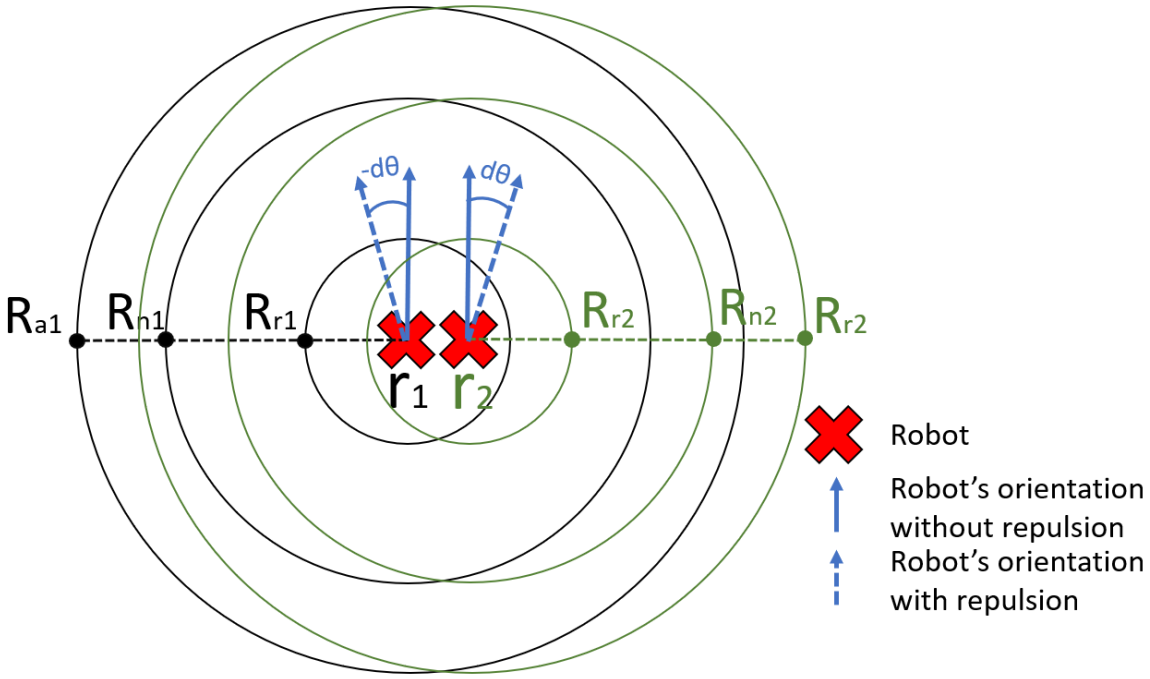


Figure 4.9: Repulsion using orientation over- and undershooting

The decision on whether the orientation needs to be increased or decreased depends on which side of the robot the robot is located at:

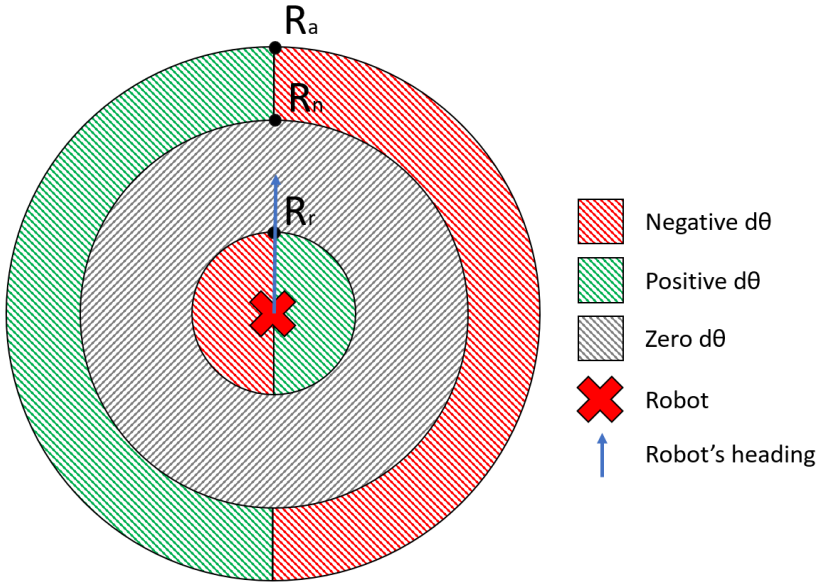


Figure 4.10: Repulsion and attraction using orientation over- and undershooting regions around a robot

The effect of this attraction and repulsion can be increased by increasing $d\theta$. As soon as the robots are out of the attraction region from another, the over- and undershooting orientation difference $d\theta$ is zero. This means that without any scaling, there would be an instantaneous desired orientation difference of $d\theta$. To make this a little more smooth, a simple linear scale is applied to reduce the instantaneous error when switching regions:

$$f(r) = \begin{cases} \frac{Ra-r}{Ra} & \text{if } R_n > r > R_a \\ 0 & \text{otherwise} \end{cases} \tag{4.14}$$

Increasing $d\theta$ decreases the time it takes for the robots to enter the neutral region of the other robots. However it also increases the instantaneous orientation error when switching to the neutral region. To show the effect, two robots are placed in parallel driving forward for different values of $d\theta$:

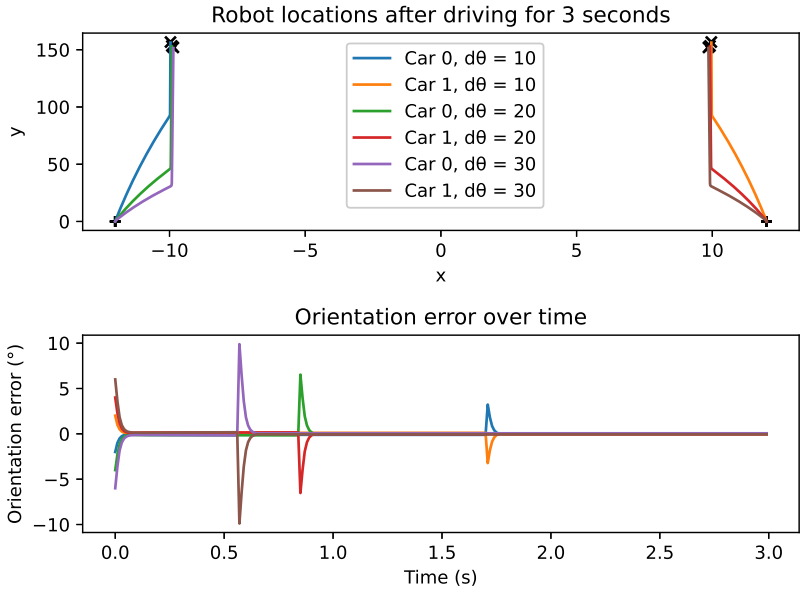


Figure 4.11: Effect of changing dθ

4.5.2. Speed based attraction and repulsion

When robots are driving horizontally parallel with respect to each other, this over- and undershooting allow the robots to attract and repel. However, when robots are driving on the same vertical line in the same direction, changing orientation will never lead to any attraction or repulsion when driving at the same speed. Therefore the speed needs to change with respect to the total average speed of the surrounding robots to either repel or attract. This is done in a similar way as repulsion over- and undershooting, but then vertically. The speed is either decreased or increased with dv:

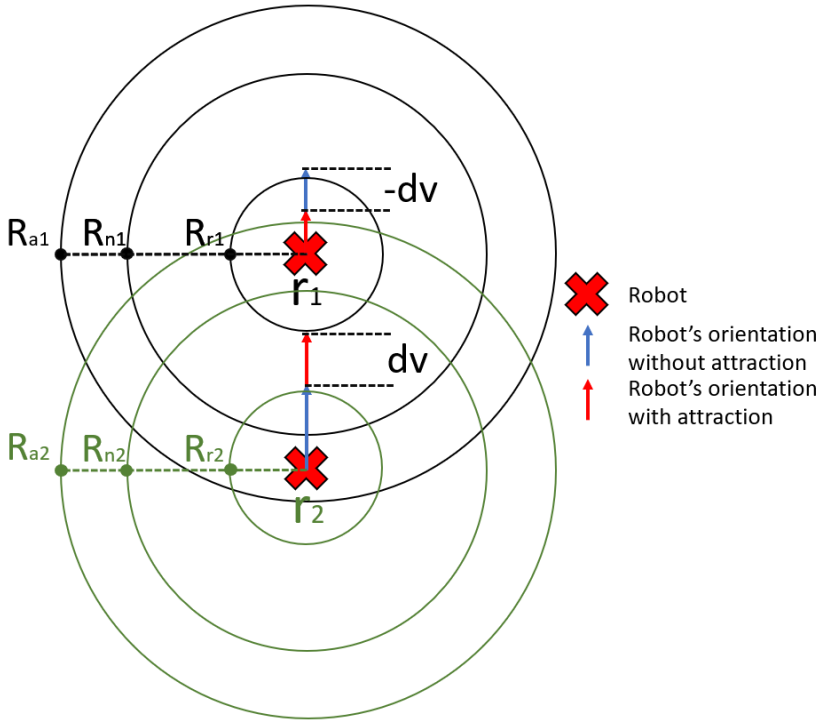


Figure 4.12: Attraction using speed over- and undershooting

The same over- and undershooting van be used for repulsion:

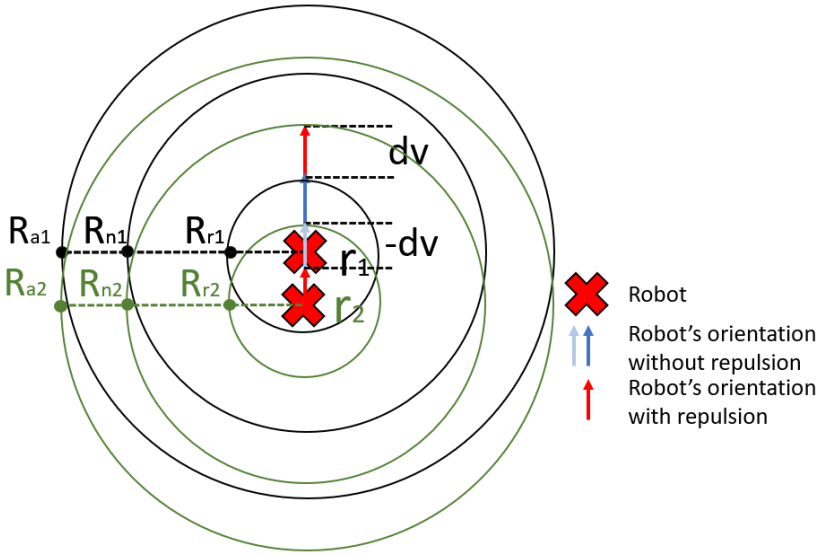


Figure 4.13: Repulsion using speed over- and undershooting

When looking at this for all regions around the robot, the following is obtained:

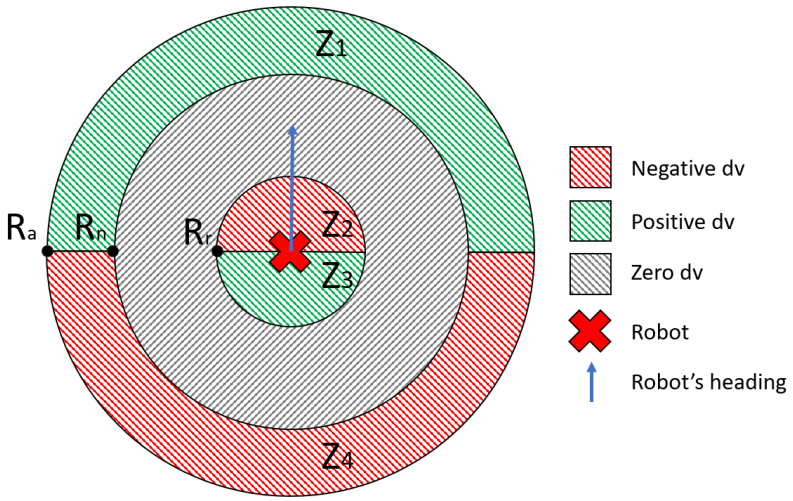


Figure 4.14: Repulsion and attraction using speed over- and undershooting regions around a robot

To calculate dv, the number of robots that are in a certain zone is counted. The zones are distinguished in order to allow different compensation for attraction and repulsion. That way, priority can be given to avoiding collisions by making the repulsion constant ($C_{repulsion}$) larger than the attraction constant ($C_{attraction}$):

$$\begin{aligned} dv = C_{attraction} \cdot (\#(c_i \in Z_1) - \#(c_i \in Z_2) + \\ C_{repulsion} \cdot (\#(c_i \in Z_3) - \#(c_i \in Z_4)) \end{aligned} \tag{4.15}$$

Similar to orientation based repulsion and attraction, dv is scaled with distance by Equation 4.14. To test this, two robots are placed on a vertical within the attraction or repulsion region of the other robot. In these figures $R_a = 30,\ R_n = 20$ and $R_r = 10$:

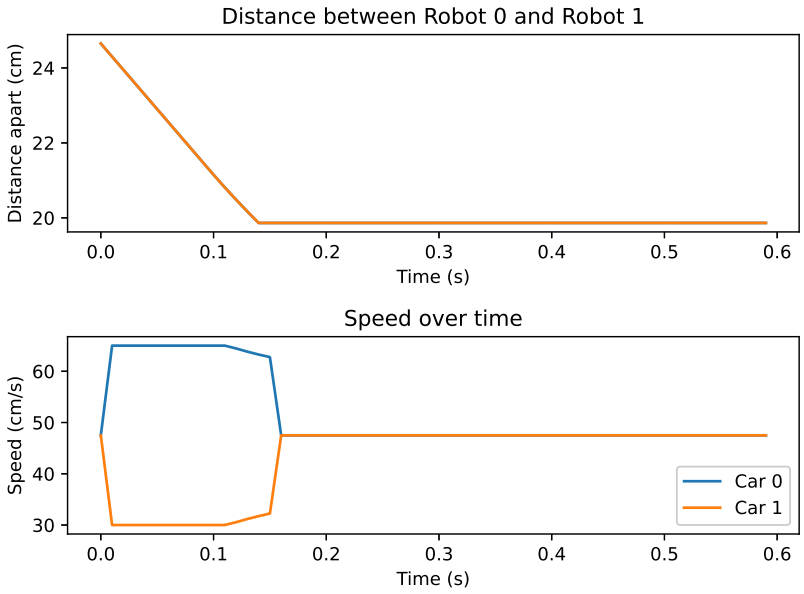


Figure 4.15: Speed based attraction

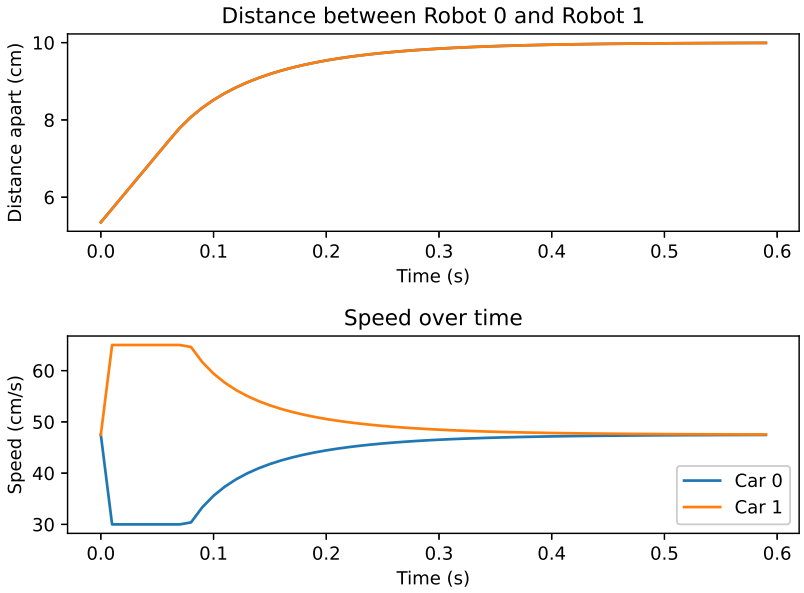


Figure 4.16: Speed based repulsion

4.6. Integrating

Now that the robot is able two attract and repel other robots using two methods, they can be combined. This is done by merging Figure 4.10 and Figure 4.14 which results in the following:

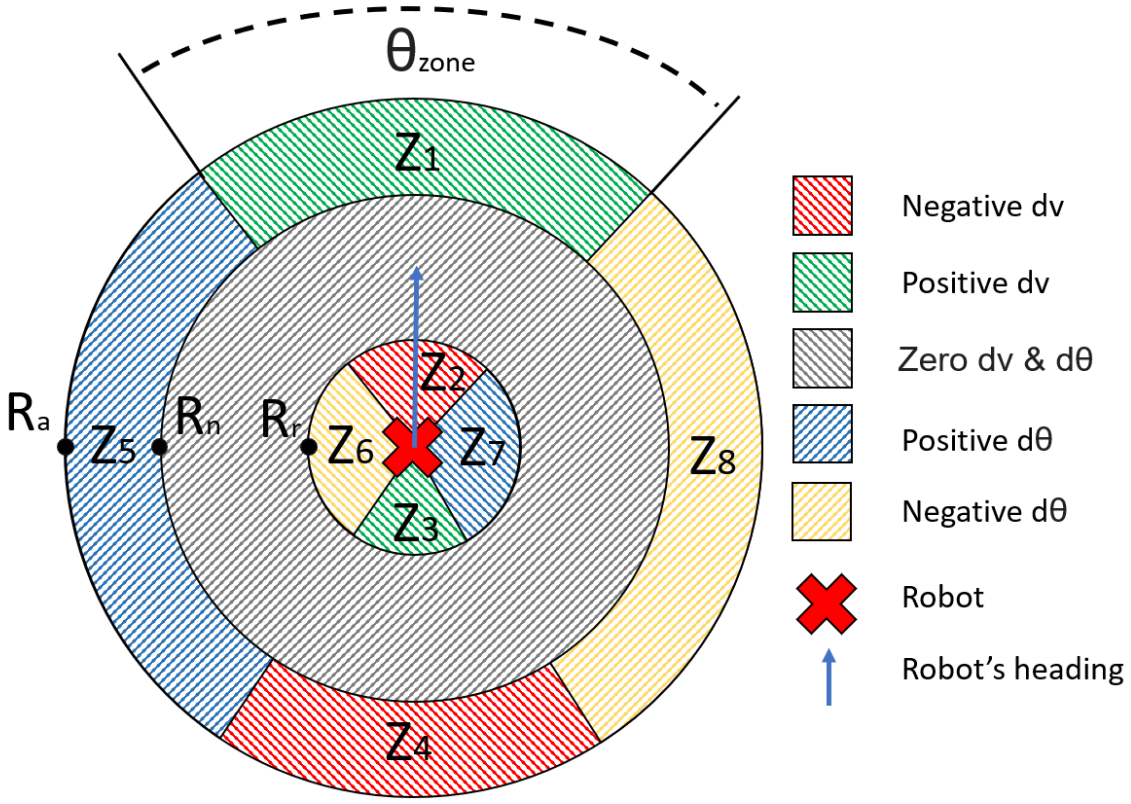


Figure 4.17: Repulsion and attraction using speed and orientation under- and overshooting

The regions are decided by the robot size:

Table 4.1: Regions radii based on physical hardware dimensions

Parameter	Value
R_r	$50cm$
R_n	$100cm$
R_a	$150cm$

By combining these regions some constraints are formed:

1. When a robot is getting too close ($25cm$ or closer) the robot is instructed to stop moving until the heading is corrected so that driving will not lead to the robots converging even closer.
2. When a robot attracts or repels using speed under- and overshooting, the current orientation is maintained to drive closer or further away instead of averaging orientation.
3. Similarly, when a robot is attracting or repelling using orientation under- and overshooting, the current speed is maintained to move away from or closer to the other robot(s) instead of averaging orientation.
4. The previous two constraints lead to small continuity between regions. The previous correction done to try and repel or attract is passed on to the different zone by maintaining it.
5. Effectively, this means the heading is only averaged in the neutral region.

Furthermore, this also leads to more design choices:

1. $c_{attraction}, c_{repulsion}$: the constants that determine the proportional gain to error. If they are unequal, one is prioritised over the other.
2. $d\theta_{max}$: the maximum angular over- or undershooting due to orientation correcting. Leaving this out could mean that the robot would have angular over- or undershooting larger than 360 °.
3. θ_{zone}: This is the angle for Zone 1 through 4. Increasing this will lead to the robot correct using speed more often. Similarly, decreasing this angle will lead to the robot correcting using orientation more often.
4. $v_{converge}$: The speed at which all robots should drive when they are in a steady-state or not receiving any other commands.
5. PID gains of the orientation controller: The orientation controller has been optimized for certain behaviour. However, with more robots, these parameters need to be tweaked again.

4.7. Results

To test the behaviour, 6 robots are placed at a random location with random orientation. The algorithm is then run for 10 seconds allowing the robots to move. Sometimes the robots are placed in an impossible to recover from position. This is tolerable as this is not a realistic situation. Therefore these situations are filtered out. The figure below shows an example of a successfully run simulation. As can be seen two small flocks have been created (top left and bottom right) moving in the same direction while maintaining the desired distance from one another. It also illustrates the example of a robot not being able to join a flock. This is in this case simply due to the fact that only 6 robots have been placed. Additional figures for different number of robots can be found in Appendix A

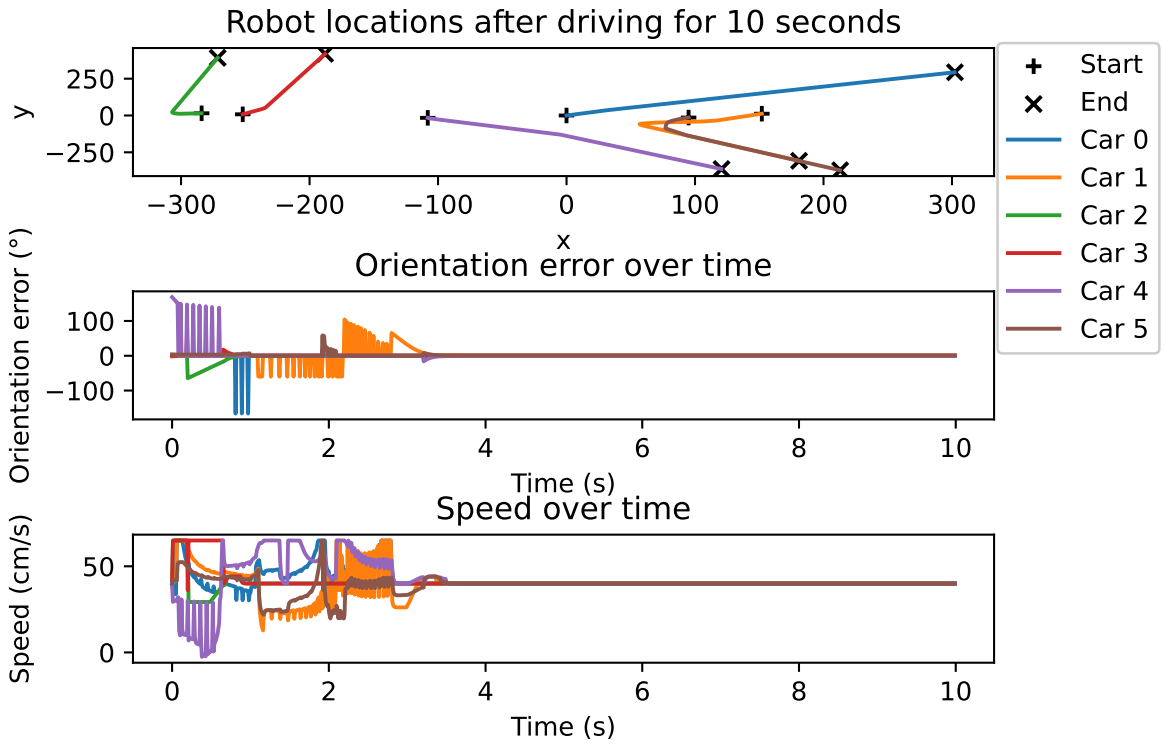


Figure 4.18: Final model of the robot with minor tweaked parameters

5

Conclusion & Discussion

5.1. Conclusion

This research on swarming algorithms has led to the following results and insights into swarming. The algorithm which was developed is able to guarantee that a certain distance is kept between robots. In this way collisions that may cause system failure can be avoided. To prove this, first, a certain distance was maintained in 1 dimension (Chapter 3). This was forced by using a PID controller, which relates the error in distance to a desired robot motor velocity. This was expanded with an orientation controller, for 2-dimensional heading alignment. This controller (Section 4.3) averages the headings of all nearby robots and outputs the desired orientation for the robot to converge to. The other variable of the robot, the speed, is determined by the speed controller (Section 4.4). It decides on a duty cycle value for the individual wheels, by averaging the speeds of all robots in a region and including a speed at which all robots should converge towards. The final part of the control algorithm is the attraction and repulsion control (Section 4.5). This section shows a method to promote close proximity to its neighbours, but also prevent collisions. It decides on action depending on where neighbours are in relation to the deciding robot. With all these controller parts integrated, the algorithm complies with all the requirements of a flocking swarm:

1. It averages the orientation of nearby neighbours and thus will move in an equivalent direction as its neighbours by using the orientation controller.
2. The attraction/repulsion controller instructs the robots to remain close to their neighbours.
3. The attraction/repulsion controller also prevents collisions between neighbours.

5.2. Discussion

The results of the research have led to an algorithm for swarming. Our goal of creating a flocking algorithm has thus been reached. However, this general model needs to be tweaked to specific user cases. Simply running the algorithm on randomly placed robots with random orientation leads to collisions sometimes. This is mainly due to the fact that the parameters are not optimally tweaked. To get the best results out of the algorithm, each parameter needs to be tweaked to the user's desired behaviour. This makes the algorithm very versatile, but also has a rather high threshold due to the mandatory tweaking.

5.3. Further Work

The current algorithm complies with the set of swarming requirements. However, it is not perfect. Further work in this subject could focus on creating a more in-depth physical model for the robot that will be used to run the algorithm. Also, smoothing out the hard edges of the algorithm could be done as future work. During the development of the algorithm, and especially towards the end, a lot of edge cases were found. It should be possible to create an algorithm which suffers less from this or smooths this out. This would allow the algorithm to be more general, and may even be less computationally expensive. Furthermore, with the algorithm developed, it is now also possible to integrate it with sensors

on a physical robot. This will require some further time, and integration problem solving but should result in a working swarm.

Bibliography

[1] Aaron J Corcoran and Tyson L Hedrick. “Compound-V formations in shorebird flocks”. In: *eLife* 8 (June 2019). DOI: 10.7554/elife.45071. URL: https://doi.org/10.7554/elife.45071.

[2] Melanie Schranz et al. “Swarm Robotic Behaviors and Current Applications”. In: *Frontiers in Robotics and AI* 7 (Apr. 2020). DOI: 10.3389/frobt.2020.00036. URL: https://doi.org/10.3389/frobt.2020.00036.

[3] Craig W Reynolds. “Flocks, herds and schools: A distributed behavioral model”. In: *Proceedings of the 14th annual conference on Computer graphics and interactive techniques*. 1987, pp. 25–34.

[4] Tucker Balch and Maria Hybinette. “Social potentials for scalable multi-robot formations”. In: vol. 1. Feb. 2000, 73–80 vol.1. ISBN: 0-7803-5886-4. DOI: 10.1109/ROBOT.2000.844042.

[5] Dongdong Xu et al. “Behavior-Based Formation Control of Swarm Robots”. Dutch. In: *Mathematical Problems in Engineering* 2014 (2014), pp. 1–13. DOI: 10.1155/2014/205759.

[6] Chase Vickery and Sayed Ahmad Salehi. “A Mean Shift-Based Pattern Formation Algorithm for Robot Swarms”. Dutch. In: *2021 7th International Conference on Automation, Robotics and Applications (ICARA)* (2021). DOI: 10.1109/icara51699.2021.9376415.

[7] Alejandro Ruiz-Esparza-Rodriguez, Moises S. Gonzalez-Chavez, and Victor J. Gonzalez-Villela. “A Flocking Behavior Model with Artificial Potential Fields for the Coordinated Displacement of Robotic Swarms”. Dutch. In: *2020 International Conference on Mechatronics, Electronics and Automotive Engineering (ICMEAE)* (2020). DOI: 10.1109/icmeae51770.2020.00019.

[8] Xiao Liang et al. “Swarm control with collision avoidance for multiple underactuated surface vehicles”. Dutch. In: *Ocean Engineering* 191 (2019), p. 106516. DOI: 10.1016/j.oceaneng.2019.106516.

[9] Belkacem Khaldi and Foudil Cherif. “Swarm robots circle formation via a virtual viscoelastic control model”. Dutch. In: *2016 8th International Conference on Modelling, Identification and Control (ICMIC)* (2016). DOI: 10.1109/icmic.2016.7804207.

[10] William M. Spears et al. “Distributed, Physics-Based Control of Swarms of Vehicles”. Dutch. In: *Autonomous Robots* 17.2/3 (2004), pp. 137–162. DOI: 10.1023/b:auro.0000033970.96785.f2.

[11] B. Brandstatter and U. Baumgartner. “Particle swarm optimization - mass-spring system analogon”. Dutch. In: *IEEE Transactions on Magnetics* 38.2 (2002), pp. 997–1000. DOI: 10.1109/20.996256.

[12] Jakub Wiech, Victor A. Eremeyev, and Ivan Giorgio. “Virtual spring damper method for nonholonomic robotic swarm self-organization and leader following”. In: *Continuum Mechanics and Thermodynamics* 30.5 (2018), pp. 1091–1102. DOI: 10.1007/s00161-018-0664-4.

A

Additional result figures

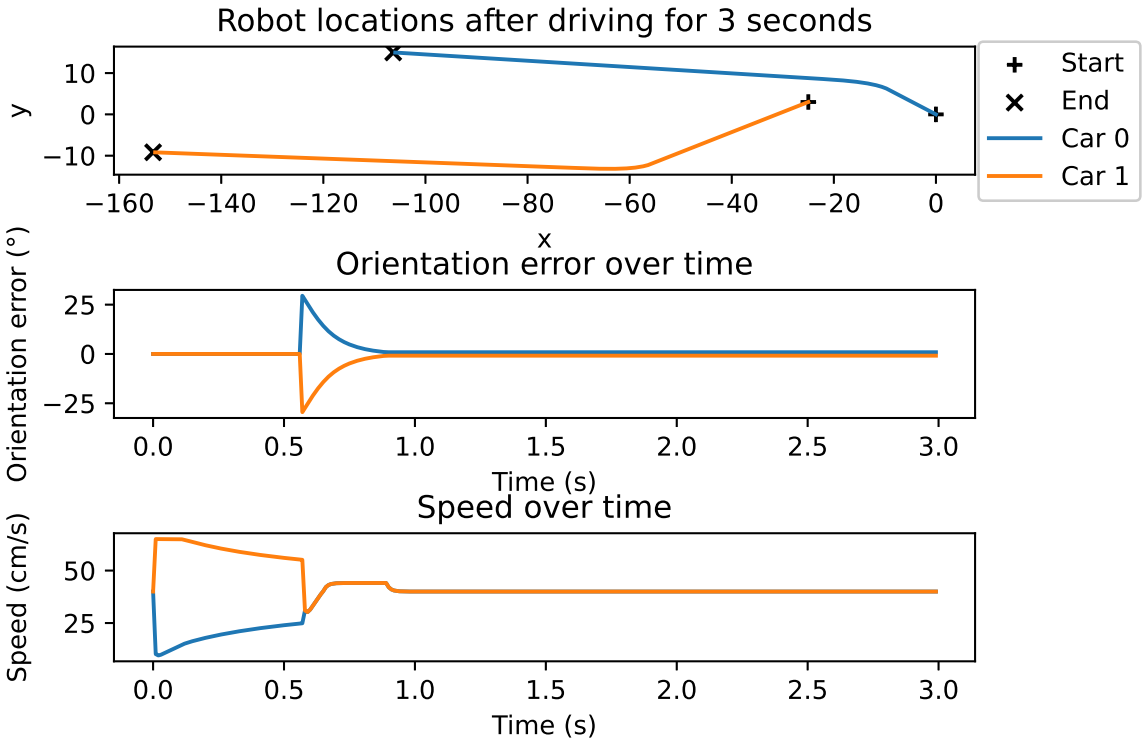


Figure A.1: Final model of the robot with minor tweaked parameters with 2 robots

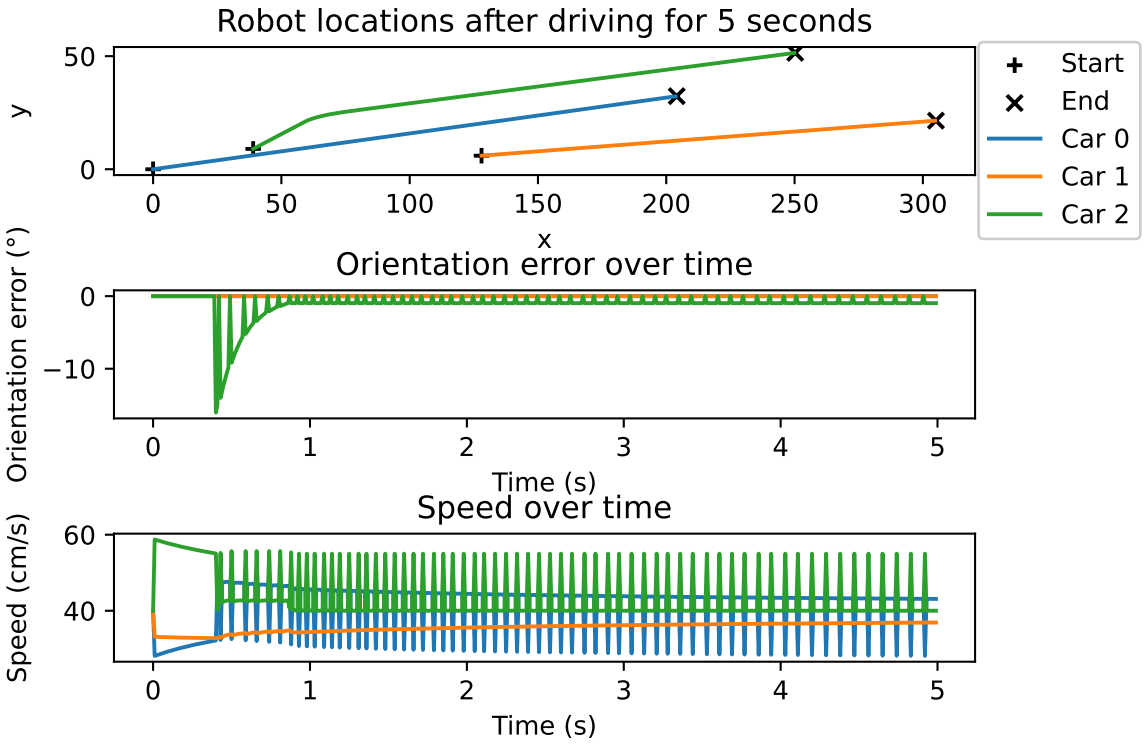


Figure A.2: Final model of the robot with minor tweaked parameters with 3 robots

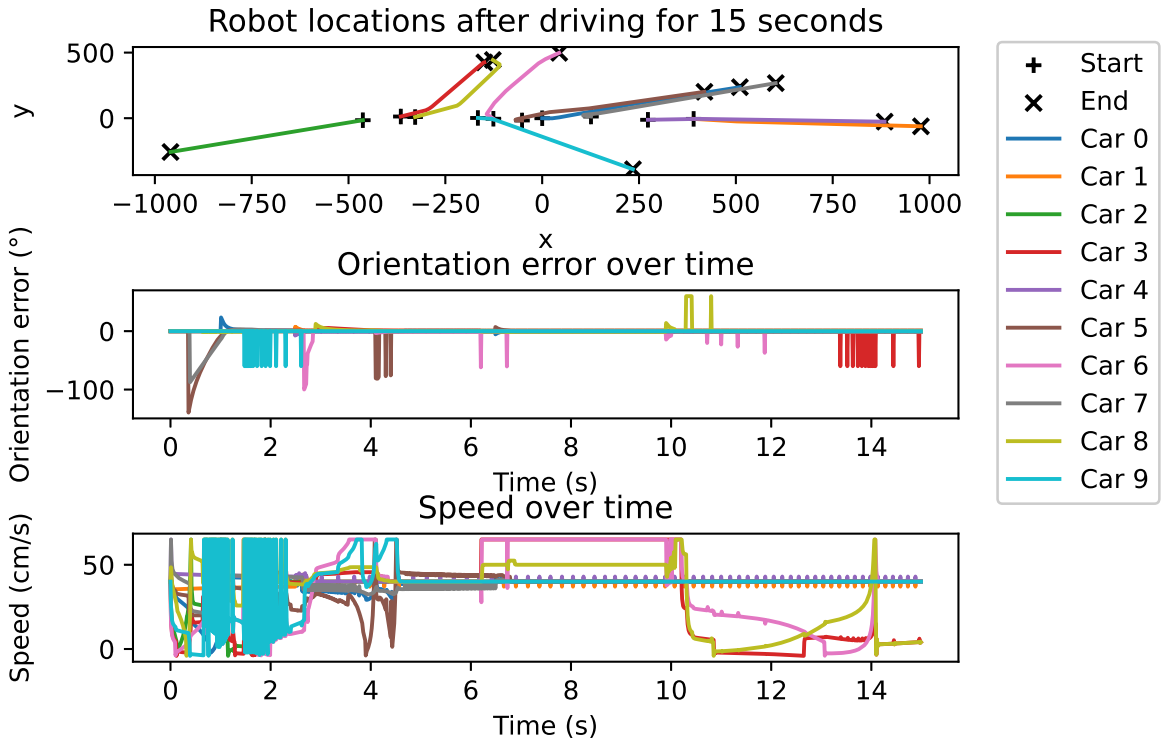


Figure A.3: Final model of the robot with minor tweaked parameters with 10 robots