Thesis report for the MSc Embedded Systems

SWARM BEHAVIOUR FOR THE ZEBRO ROBOT

Jurriaan de Groot

Swarm Behaviour for the Zebro Robot

by

Jurriaan de Groot

Thesis for the partial fulfilment of the requirements for the degree of

Master of Science
in Embedded Systems

at the Technical University of Delft
to be defended on the 29th of November, 2017

CE-MS-2017-13

Associate Professor	Dr. ir. Chris Verhoeven	TU Delft
Supervisor	Dr. ir. Edwin Hakkennes	Senior Consultant Technolution
Master Coordinator	Dr. ir. Arjan van Genderen	TU Delft
Thesis Committee	Dr. ir. Stephan Wong	TU Delft

Acknowledgement

During my studies, I have found an increasing interest in robotics and AI. Being able to help work on the Zebro robot has been an incredible learning experience. While working on Zebro, I have had the pleasure to meet and work with remarkable people.

First of all, I want to thank Chris Verhoeven for making the Zebro Project possible. His cheerful attitude and confidence in the project has motivated me along the way.

I would like to thank Edwin Hakkennes for his organization, guidance, and for his feedback on my work. His suggestions have helped me give shape to the final result.

I want to thank Arjan van Genderen for his advice during my thesis and my masters. He's always available and has helped me find the right courses to finish my masters.

And of course, I'd like to thank all other members of the Zebro team. Working with many people of different specializations has been an incredible experience, and their positive attitude made the lab a fun place to be.

Lastly, I would like to thank my parents for supporting me on my lengthy study journey. They were always ready to provide love and support along the way, this would not have been possible without them.

Jurriaan de Groot

Abstract

Nature has found many interesting solutions to similar problems that we face with robots. Swarming in robotics is an interesting field to reduce cost and risk, and provide a scalable solution to certain problems. The Zebro Project of the TU Delft performs research in the field of swarm robotics using the six legged robot, called a Zebro. Swarm robotics consists of using a large amount of small, independent robots that respond to one another in order to create a distributed, global behaviour.

The robots have been simulated using the Unity engine, to investigate methods to accomplish swarming behaviour. The simulated Zebros and sensors are modelled after real life designs. The algorithm is based on bird flocking behaviour to create mobile and cohesive swarming behaviour. The Zebro robots are not ready for testing yet, but the algorithms have been prepared for testing using a Raspberry Pi.

Index

1. Introduction

We are often inspired by nature for its ingenious solutions to problems we face as humans. In the field of swarm robotics, robots try to imitate the behaviour of social animals and insects, and let them cooperate as a group to achieve a certain task. The Zebro, a six legged robot capable of traversing rough terrain, has been designed as a versatile platform for research purposes, and can be fitted with different modules to prepare them for specific goals [1].

Swarming with robots has a number of important features. Each robot is relatively cheap compared to one, complex robot. A broken robot can be replaced more cheaply and easily without interrupting the operation of the rest. The swarm also does not have a central controlling unit, so each robot should make its own decisions and cannot directly influence the movement of another. An individual robot will only respond to its immediate surroundings, and does not have knowledge of the entire swarm.

This means that Zebros can be used for tasks which require a scaling solution. Zebros can be added and removed from the swarm at any given moment, and the swarm will still operate the same way. This means that more Zebros can be deployed for larger tasks, or can be taken away when the remaining task at hand does not require as many robots anymore. The Zebros could be used for search & rescue, such as finding survivors after an earthquake, or for exploration, such as patrolling and mapping the surface of the Moon or Mars.

The robots are not yet ready to experiment with. The Zebros are first modelled and simulated digitally to find a suitable swarming algorithm. The design of the virtual Zebro has been made to resemble the physical Zebro as closely as possible, to minimize the differences of the transition from simulation to robot. This research focuses on creating a simulation for a swarm of Zebros, and the design of the algorithms to achieve swarming behaviour. The algorithms can then be used to control the desired heading of the Zebro robot.

2. Background

2.1 Introduction

The act of swarming has been found in nature in many different types of animals and insects, in various forms. It allows groups of animals to work and survive together without the need of a leader, and it happens with varying sizes of animals/insects.

This method of cooperation is also interesting for the field of robotics. Swarms of smaller robots can be used for tasks where the scope is varying, or not immediately clear. Robots can be added or taken out of the swarm to fit the problem.

Because the swarm functions without a leader or otherwise externally controlling entity, the swarm will continue to function if any of its members dies/breaks down. This makes the system robust and broken individuals can be replaced easily. These features make a swarm a potential solution for specific challenges, such as problems that require adaptability and robustness [2].

A swarm has its disadvantages however. Without controlling entity, influencing the swarm becomes difficult. This also makes it unpredictable, more difficult to understand and slower to get started.

This chapter displays the information found regarding swarming, both in nature and in robotics. We will investigate what swarming is, how it can be realised and applied and how nature has applied these in various animal behaviours to achieve specific goals.

2.2 Swarms in Nature

The idea of creating a cooperating swarm of robots comes from nature. There are many reasons in the animal and insect kingdom to work together. The main reason is survival, staying within a group means there’s less chance of being taken down by a predator.

Schools of fish cluster together in very close proximity, without colliding. The fish closely respond to directions of others around them, allowing the school to move and turn at high speeds. The patterns on their bodies often help confuse predators to increase their chance of survival together. Staying together also helps foraging.

Ant colonies work together in large numbers in a nest. When looking for food, foraging ants leave a pheromone trail in the environment when they are successful on the way back. Other ants can follow this trail and end up at the same food source. When also successful, they also release a trail. The best trail will have the most ants and eventually the optimal route will be strongest, and followed by most ants.

Birds flock together and fly in formations when migrating. The formation reduces wind resistance for birds flying in the back of the formation, conserving energy. Foraging as a group can yield more food and reduce the chance of being caught by a predator. Birds gather information for the direction of their destination by using a variety of sensors, such as a sun compass, magnetic fields and visual landmarks.

2.3 Types and Criteria of Swarm Behaviour

Self-Organization - Self-Organization (SO) in nature, is a process where sudden order appears without an external agent. The initial trigger often comes from a random event, which is then reinforced through positive feedback. Due to this nature, it's typically robust and able to withstand perturbation. Examples of self-organization can be found in physical, chemical, biological, robotic and cognitive systems. A chemical example is crystallization, an example in nature is animal swarming [3].

Swarm Intelligence - A swarm is a population which is self-organizing, interacts with each other within the swarm, without any central instance controlling the group (A swarm is decentralized). The swarm often works together to achieve a goal, where each individual of the swarm acts based on its current state, location and environment. A swarm often starts off chaotic, and slowly turns into an organized group through communication between members, with intelligent behaviour as a result. There are a number of algorithms based on this approach. Two of these are Flocking with Particle Swarm Optimization and Ant Colony Optimization [4] [5] [6] [7].

Flocking Behaviour - Flocking behaviour can be observed in groups of birds, called a flock, that are in flight or foraging. Birds in a flock adjust their movement to avoid predators and seek food. Based on their neighbours within the flock, the individual agent is modelled with these simple rules:

- Cohesion: Steer towards the local position of flock mates to remain close
- Separation: Steer away from the local position to prevent crowding
- Alignment: Steer towards the average heading of local flock mates

The result is that all agents move in a formation with a common heading, while avoiding collisions between agents. The Particle Swarm Optimization algorithm uses the principles of this behaviour to find a minimum solution [8] [9].

Particle Swarm Optimization - The Particle Swarm Optimization (PSO) algorithm is an iterative method that tries to improve a candidate solution by giving a measure of quality and is inspired by fish schooling and bird flocking behaviour. The algorithm starts off with a number of candidates (dubbed here as particles), and these particles are moved around in the search space by following the current optimum particles. Each particle keeps track of its coordinates which are associated with the best solution (fitness). The particles also keeps track of the best attained solution so far by neighbouring particles (local best), and the algorithm keeps track of the global best. The optimization consists of accelerating each particle towards its own best and local best fitness values. The acceleration is weighted by a random term. PSO has been successful in finding the optimum faster and cheaper compared to other methods such as Genetic Algorithms [10] [11].

The Ant Colony Optimization - The Ant Colony Optimization (ACO) algorithm, as the name suggests, is based around the behaviour of social insects such as ants. It is a probabilistic technique that can find good paths over time through graphs. In nature, ants wander around randomly, searching for food. When an ant finds a food source, it walks back to the nest, while leaving behind a trail of pheromones. When the trail is picked up by other ants, they will start following it to the food source and leave pheromones on the way back.

Over time, pheromones evaporate, losing strength. Longer paths have more time to evaporate due to longer travel time per ant, and as a result become less strong than shorter paths. Stronger pheromone trails will be followed more and thus become enforced by large amounts of ants. Eventually most ant will start following the shortest found trail [12] [13] [14].

2.4 Features of Swarm Robotics

A group of swarming robots is generally defined to have the following features.

Autonomous/Decentralized – Each individual robot should not be controlled from an outside source, and make its own decisions based on the environment, other robots, its own position and status, and the end goal.

Parallel – Each member of the swarm acts at the same time, creating a large group of parallel processes. A swarm can focus and act on multiple targets in a larger environment at the same time.

Scalability – The robotic swarm should be able to function with any number of other robots, from 10 to thousands, while still cooperating.

Limited Capabilities – Each robot does not have the tools to complete the goal on their own. A larger number is required to be able to succeed.

Local Sensing and Communication – Communication is done strictly within a few meters around each robot. This makes sure that the swarm is scalable to any degree: Communication between each robot with thousands of swarm members is impossible due to the overhead.

Homogeneous – The robots of the swarm should be homogeneous. Groups can exist where their function slightly differs from others, but the number of groups should be small [4] [15].

2.4.1 Differences: Multi-Robot Systems versus Swarm Robotics

Working with multiple robots does not mean that the system is a swarm. A multi-robot system is set up to overcome limitations in processing capability, and are generally controlled from a centralized point. The robots need to cooperate and collaborate to achieve their goal, but the main difference is that each robot is able to perform a meaningful part of the task. Without some of the participating agents, it may not be possible to reach the end goal (But parts of the goal can be reached successfully). Where in Swarm Robots, each individual robot is not able to complete any task on its own, it can do so with any size group of (almost) identical fellow robots.

2.5 Common Swarm Applications

Swarm robotics can aid in a number of situations, where a small number of robots might not.

Cover Area – Many small robots can gather information about their surrounding and cover very large areas. The size of the swarm can be adjusted to suit the area to cover, thanks to the scalability property. The swarm can be used for exploration and/or monitoring. While sensor networks can monitor large areas, swarm robots can instead move around and monitor the same area with fewer agents. Besides monitoring, the swarm robots can also find the source and even take actions, such as patching a leak temporarily by collectively blocking it off.

Dangerous Environments – Due to the redundancy and scalability, swarm robots can work in environments that are dangerous, even to robots. When some robots are lost during the task, the swarm continues to operate. The cost of losing a few robots is also less.

Tasks Requiring Scaling – When the size of a task is still unknown, or the workload changes over time, different amounts of robots can be put into action to suit the current demands. An example is an oil spill, where the spill begins large and a large amount of robots are needed. As the spill shrinks, less robots can work at the same time, and the numbers can be downscaled.

2.6 Methods of Communication

This section discusses a number of ways that animals and insects use to send information to each other.

*Direct communication – C*ommunicating directly is the easiest way to convey information. This can be done in many forms, such as the bee's dance to directly convey the location of a good new hive location. It requires one end to speak and the other to listen, which is not always preferred with many agents wanting to speak, or possible when a swarm is spread out.

Stigmergy – The environment itself can be a good way to leave information for others to find. This is called stigmergy, and is a method of communication and organization by modifying the surroundings in some way. In nature, some swarming insects leave scent marks (pheromones) in the environment to reinforce a certain type of behaviour. In an ant colony, ants leave pheromone trails when returning to the nest upon finding food. The pheromone trail encourages other ants to follow the trail to end up at the same food source to gather it in larger numbers. In mammals, scent marks are left in the environment to mark a territory, repelling competitors or attracting mates.

Another form of stigmergy with organization is sorting. Smaller groups are put together to form a bigger (higher density) group. This bigger group becomes more attractive to sort similar items into, reinforcing a sorting behaviour. This can be found in social insect hives, where larvae of the same growth stage are sorted together [16].

2.7 Simulation Programs

There are a number of suggested platforms that are suitable for simulating robots. Each program has advantages and disadvantages, some are more suitable for single robots, others allow for larger numbers of independent agents. The complexity differs between each program, and a balance should be found about technical depth as well as ease of use. Since a large part of the project is expected to be done in a simulation, it's important to pick a suitable simulation program [15].

The Player Project - The Player Project is a free software tool for robot and sensor applications, released under the GNU General Public License. It consists of three pieces of software: Player, Stage and Gazebo. Each program can work alone, but can interconnect for expanded features. The Player Project is available for Linux.

- *Player* - Player is a program that provides a network interface to robot and sensor hardware. It allows you to connect with mobile robots over the local network. Its interface allows programming a control program in any language.
- *Stage* - Stage is a multiple robot simulator in a two-dimensional environment. The robots it simulates are greatly simplified compared to Player, to provide better performance with large numbers of robots. It allows moving and sensing with a variety of sensor types and can generate heat maps of robot locations. Stage is a great tool for testing swarming algorithms.
- *Gazebo* - Gazebo is a 3D multiple robot simulator (Whereas Stage is simulated in 2D) where multiple robots can be simulated in more realistic environments (e.g. outdoors). Written programs can be used between Gazebo and Stage, and works together with Player. The Player Project software is supported on Linux and MacOS [17].

UberSim - This simulator is designed to simulate soccer robots in a realistic scenario. It uses an ODE physics engine to simulate physics and interactions. Extra robots and sensors can be programmed in C and added to the simulator [18].

USARSim - Urban Search And Rescue Simulator is designed in the 2.0 Unreal Engine, for research and education. It is a multi-robot simulator that uses the Unreal high accuracy physics engine. The Unreal Engine was originally designed for games, but works well for simulations [19] [20].

Enki - A 2D open source robot simulator written in C++. It has collision and limited physics on a flat surface. The simple design is also much faster, allowing simulation of much larger swarms of robots [21].

Webots - A 3D development environment in which multiple robots can be simulated at the same time. It features realistic simulations of commercially available robots and sensors, and allows the user to tweak each individual part in terms of colour, weight, friction, etc. [22]

Breve - A free program that enables users to create 3D simulations of multi-agent systems. The simulations are written using a language called *steve*, which is based on C, Perl and

Objective-C, and also features full support for simulations written in Python. It has physics simulation, collision detection and response [23].

*V-REP - A*n open source 3D robot simulator that allows control over each individual part through embedded scripts. These scripts can be written in C/C++, Python, Java, Lua, Matlab, Octave or Urbi. It can simulate multiple agents in real-time [24].

ARGoS - ARGoS is a multi-physics simulator designed for simulating large scale swarms of robots in real time. Each robot can be implemented as an individual plugin, and the program is designed to handle thousands of robots in a swarm at the same time. ARGoS is supported on Linux and MacOS [25].

MORSE - Modular OpenRobots Simulation Engine is a generic simulator for academic robots. The simulations are done with Python scripts that describe the robot and its environment, and are done in the Blender Game engine. The simulator comes with a set of standard sensors, actuators and robotic bases, and allows adding new ones easily. MORSE is focused on large, complex robots but can also handle a small amount of simpler robots in one environment. This makes MORSE less suitable for our application [26].

TeamBots - TeamBots is a Java-based collection of application programs and packages for multi-agent mobile robotics research. It supports prototyping, simulation and execution of multi-robot control systems [27].

Unity - Not aimed at robotic simulations, Unity is a game engine that offers physics simulations. Like USARSim is based on the Unreal game engine Unity can be used to simulate a swarm of robots using its inbuilt physics engine. The paper [28] describes a method of approach for modelling a swarm robot with sensors. Unity can be used with the programming languages C#, Javascript and Boo [29].

2.8 Hardware Study

One of the challenges and required components of swarm robots is locating and communicating with other members of the swarm. They need to estimate where other members are in relation to itself.

Hardware Platform - As one of the goals of the Zebro project is to make the system modular, the programming should be done on its own platform. The swarming behaviour is designed to be simple and only simple decisions should be made on an individual level.

Communication - In order to convey information from one Zebro to another, a form of communication is needed. Since the swarm is required to be scalable, the communication should be done locally, with other Zebros being able to join and lose connection to the local network.

Zigbee and Bluetooth - Zigbee and Bluetooth (specifically BLE: Bluetooth Low Energy) differ from each other on a technical level, but the goal of these network standards is to create a (small) local network to transfer data wirelessly. Both standards are capable of transmitting data in form of a broadcast, thus no pairing is needed and any number of nearby Zebros can receive messages.

Zigbee and Bluetooth differ in operating frequency and energy cost (though Bluetooth 5 has become much more energy efficient too). Zigbee operates at around 900 MHz versus 2.4GHz (Wifi frequency) for Bluetooth, which gives Zigbee a larger range through obstacles. Both standards have a range of about 10 meters [30] [31].

Infrared - Infrared can be used as a method of communication that requires direct line of sight from IR transmitter to receiver. The communication is done in a broadcast-method, where information is sent out without knowing who will receive it. An experiment with the e-Puck robot has shown it can also help with relative orientation between robots, based on the angle that the infrared light is received [32].

2.8.1 Orientation

Compass - One way to know the orientation is to add a compass. The compass measures the magnetic field to determine the direction of the north pole. This works because Earth has a strong magnetic field, however this is not present (as strongly) on other planets, e.g. Mars.

Gyroscopes and Accelerometers - Another method of determining the orientation is by using gyroscopes and accelerometers. One can estimate their position by using a landmark, and using your orientation and velocity to determine your relative position after some time. This method of navigation is called “dead reckoning”, and is subject to cumulative errors.

2.8.2 Range Finding

Infrared Triangulation - After determining your own relative position, you want to know where the other members of the swarm are in relation to your position and orientation. To prevent collisions and to stay with the group, it is important to know the distance to the nearest swarm robots. This could be determined by measuring the strength or quality of signals such as Bluetooth or Wifi coming from other swarm robots. This has many disadvantages however; it is not accurate, and with many signals happening at the same time, the signal quality sharply drops. Signals bouncing off walls can increase or decrease signal strength, making the calculation vary wildly.

A strong alternative is infrared; Since it uses light, it is detected with line of sight. If the robot cannot see the other swarm member, they won’t be able to communicate or measure their distance.

Infrared can determine distance based on a process of triangulation. The sensor has two components: One IR transmitter and an IR phototransistor. The transmitter sends out an infrared signal, which is reflected off of an obstacle (Or not, if too far away) ahead back to the phototransistor. The distance can be measured by determining the angle at which the infrared signal is received.

Infrared is okay for accuracy due to the nature of light; It is reflected differently off of the same surface depending on the material and colour. It is also affected by sunlight and has a narrow detection width. Infrared range sensors can be found in analogue or digital (byte) variants, and the maximum and minimum distance are depending on the sensor [33].

Infrared Time of Flight - A second method of measurement with infrared is by using Time of Flight (ToF). By using a precise clock, this type of sensor measures the time it takes for light to bounce back from a surface. The accuracy is a few millimetres, which would be a good option for the swarming application.

Ultrasonic - Another alternative are the use of ultrasonic sensors. Where IR uses light, ultrasonic sensors use sound for ranging. The ultrasonic sensor sends out pulses of sound, inaudible to humans. At the exact same time, the sensor sends a pulse to the microcontroller, which then keeps track of time. When the soundwave comes back, it is picked up by the sensor, and another signal is sent back. Based on the time, the distance of objects ahead can be calculated.

The sensor suffers from a maximum and minimum distance: Since sound is pulsed out in a cone, the strength of the wave diminishes quickly over distance. When it travels too far (2 to 8 meters, depending on the sensor and conditions), the returning pulse is too weak to be picked up by the ultrasonic sensor. The minimum distance (~30 cm) is caused by the speed of the sensor, if an object is too close the sensor the pulse is returned too quickly to do calculations with. It also suffers from “ghost echo”, where pulses bounce off of surfaces under an angle, then off of another surface, and are then returned. This means the pulse travels further, and the distance measurement is too far.

Ultrasonic sensors are more accurate than IR sensors and work regardless of surface area you would normally find in a room (sponges or foam for example absorb sound much stronger and lower range). Pulses from other robots can cause a lot of problems however, since every pulse is technically the same. When these pulses are received by other robots in the area, it can cause wrong measurements, and might not be suitable with many similar robots [34].

Both ultrasonic and infrared sensors are good options for range finding. The IR method is used by many other swarm robot researches [35] [36], such as the Kilobot [37], Colias [38], Khepera III, Jasmine, Kobot and SwarmBot [39].

2.8.3 Sensor Positions

An important part of the sensors is the placement on the Zebro. Sensors such as a compass, gyroscope and accelerometers will work on any location, but the infrared/ultrasonic sensors need a direct line of sight. Many swarm robots from other parties such as the e-Puck [40] are fitted with 6 to 8 infrared proximity sensors, that are put in a circle on the top. This allows the robot to scan around its entire body for obstructions. The Zebro has potential to put sensors on the top, but during its walking cycle, the legs might block sensors pointing sideways, causing interference.

2.9 Previous Student Research

In the past, a student team have worked on a model for swarm intelligence for the Zebro robot [41]. This section summarizes their achievements and important findings. The requirements at that moment were slightly different, such as the requirement that the communication module should be compatible with the Delfly communication module.

Their report about swarming focuses on wireless communication in the third chapter, where they have considered Wifi, Bluetooth and Zigbee. After comparing the different wireless standards, Wifi is not as suitable due to lacking an advertising feature. The BLE113 module has a ranging tool called iBeacon, that allows information broadcasting in set timeframes.

In the next chapter, the team discusses the possibilities of using the BLE113 as a range finding tool. After their measurements, they conclude that it could be used to find the distance to another module when they are within one meter. This could be used to determine when Zebros are too close, but it's not accurate enough to determine when they are too far away.

The behaviour of the swarm is discussed in the next chapter, where the team suggests three main behaviours: Individual, anti-collision and separation-prevention behaviour, with binary and fuzzy-logic based probabilities of which behaviour to follow. The next three chapters describe these three behaviour types.

The individual behaviour should operate when the other requirements are met, such as keeping a desired distance to all other Zebros and having high enough battery level. The goal proposed in the report is to cover as much new ground as possible, and doing this with as low as possible energy cost. Covering area twice is considered as lost energy.

The anti-separation behaviour section discusses the actions that could be taken to prevent separation in a couple of instances. The anti-collision behaviour activates when Zebros get too close. It discusses when to take action, such as when moving in the same/opposite direction, and what actions to take accordingly.

The last section discusses another behavioural approach. Instead of assuming small-scale collision or separation, they look at a large-scale regulation, without adopting a central controller. This is done by estimating the direction and location of individual Zebros through a set of possible methods and constructing a table with local Zebro positions and locations for each individual Zebro.

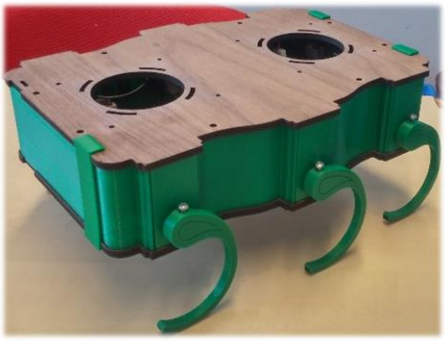
3. The Zebro Project

3.1 What

The Zebro Project was started in September 2013 and, shortly after, established in the TU Delft Robotics Institute as part of the Robotic Swarm theme. The project is centred around the Zebro, a six-legged robot (Zesbenige Robot) that is capable of traversing rough terrain.

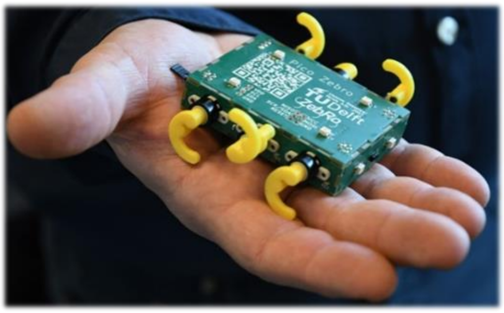
Three variants, or sizes, of the Zebro are being developed. The largest version, the KiloZebro, is nearly a meter long and significantly stronger than the other variants. The medium version is the DeciZebro. This design is intended to be deployed as a swarm with about 100 members. It is about 30 cm long and completely modular. The smallest version, the PicoZebro, is the size of a matchbox.

The KiloZebro is designed to be a powerful carrier, capable of transporting a number of Deci- and PicoZebros into the field. Its larger frame allows it to move over rough terrain more easily. It can also be deployed as a mobile charging dock for DeciZebros, or collect broken robots in the field.



The DeciZebro is designed to be mass produced. Its medium sized frame allows it to walk over grass fields and climb the kerb of a sidewalk. Internally, its design is divided in modules, such as the motor, communication and sensor module. It is made modular so that they can be swapped on the fly and designed to fit a specific purpose without the need to replace the other modules. Two holes on the top allow the Zebro to be equipped with extra tools, such as tools for vision or communication purposes.

The PicoZebro is small and can be deployed from the larger Zebro variants in large amounts. Its small frame is made up of 6 PCBs, giving it a light and compact frame. It is currently being controlled using an external computer, using a top-down camera and Bluetooth. The PicoZebros are small enough to work on for example a table.



A second team of students is working on a specialised version, the LunarZebro, designed to operate on the Moon and on Mars. The environments require a specialised design and the Zebros need a rocket to reach these locations in the first place.

3.2 Why

The Zebro project, as part of the Swarm Theme within the Delft Robotics Institute, serves as a platform to further research swarm behaviour in robots. Their mission is:

"*To design, plan and build self-deploying, fault-tolerant, inexpensive and extremely miniaturized robust autonomous roving robots to cooperate in swarms, capable of functioning on a wide spectrum of topology and environment that can quickly provide continuous desired information with the help of distributed sensor systems and carry and support payloads suitable for a wide range of missions.* " [42]

3.3 Who

The project is led by Chris Verhoeven, an associate professor in the department of microelectronics. Chris is also part-time employed at the faculty of Aerospace Engineering and since 2013 the theme leader of Swarm Robotics.

The project is supervised by Edwin Hakkennes, Senior Consultant at the company Technolution. Edwin makes sure that the project stays organized, and hosts team meetings every Tuesdays to discuss the progress of the members among the project.

The project groups consist of students from different schools and universities, and are split up in a number of subgroups that are working in parallel, such as the chassis design, locomotion controller, range finding and localization, and our group, the swarm intelligence. The swarm intelligence group consists of Pengqi Chen, a MSc Embedded Systems student, Shamburaj Sawant, a bachelor student working in his own time, Mario Collera, a PhD student, and myself. Within this group I have discussed our work mainly with Pengqi Chen, who designed his own simulation in Matlab, and with Shamburaj to a lesser degree due to conflicting time schedules.

4. Zebro Swarming

There have been a number of efforts around the world into the research subject of swarm robotics. Small robots designed for the purpose of swarming have been designed before, such as the Kilobot, Kobot, ePuck and SwarmBot. The Zebro is designed as a research platform for the TU Delft to investigate the possibilities of swarming. Aside of the Zebro, swarming behaviour is being researched and tested using drones, at the TU Delft faculty of Aerospace Engineering.

4.1 Goal

At the start of the project, a true end-goal was kept deliberately vague. A few uses were suggested, and an ambitious plan to deploy the swarm as a radio telescope on the Moon or Mars has been researched. For this purpose, a Lunar Zebro capable of operating on the Moon is being researched and developed.

For the first version of swarming intelligence, the focus is on implementing flocking behaviour, a behaviour in nature that is observed in numerous animals such as birds, fish and insects. Flocking means, staying together as a group, move in the same direction and cover a distance as fast as possible as a group.

4.2 Research Questions

During the literature study, both hardware and software topics were discussed. The project should focus specifically on the swarming intelligence and it should not concern with the hardware used for sensors or locomotion, as these are worked on by other students. The hardware research has been very helpful however, to investigate possibilities and discuss the sensor information needed to make swarming possible.

We define the following research questions.

- Can a swarming intelligence algorithm work on the DeciZebro Robot?
- Can swarming be shown in a simulation, using the Zebro as model robot?

- Swarm Intelligence Questions
  - What is the performance of the swarm?
    - How can this be measured?
  - How does the performance change when the swarm size changes?
  - How do different parameters influence the swarm performance?

- Hardware Questions
  - What are the challenges from transforming a simulation to a physical test?
  - What type of processor is suitable to implement the algorithm?

4.3 Requirements

The requirements can be split in multiple subjects: technical requirements and behavioural requirements. Not all points are meant for the swarming intelligence specifically, but are important to model the real life limitations of the Zebro swarm in the simulation.

4.3.1 Technical Requirements

There are a number of important features that should be fulfilled to be able to count the system as a swarm. These points have also been discussed in the literature study, and are repeated here for completeness.

Scalability: It should work with any number of robots. New robots can be added to, or removed from the group during operation.

Autonomous and Decentralised: Each robot should make its own decisions based on its sensor data, gathered from its immediate surroundings. Outside knowledge is not allowed, such as an overhead camera.

Local Sensing and Communication: The robot sensors can only see their immediate neighbours, and communication can only be done in a limited area around the robot.

Homogeneous: While not technically a requirement, the first tests will be done on a homogeneous robot type and therefore the swarming intelligence should be designed with this in mind.

4.3.2 Behavioural Requirements

The discussion about this topic has been vague from the start. When asked what the swarm should be able to do, the response has been: "Everything".

The behaviour of the swarm should be designed for general movement, and when a task has been determined, the robots can be programmed to deal with the problem in a more specialised fashion. The main focus of the thesis is designing a flocking behaviour to keep the swarm together and agree on a heading.

4.4 Assumptions

There are many variables and unknowns in the project, including the Zebro robot itself. Since all parts are actively being worked on, the dynamics of the locomotion, the method and accuracy of range finding and communication have not yet been determined up until now. A physical test was not possible during my research.

To get my thesis started, the project will be designed and tested in a simulation. This will give a good indication if swarming intelligence is possible with the restrictions of the Zebro. The parameters used will be picked to resemble the dynamics of the robot, but they are not necessary accurate for the physical model yet.

5. Simulation Model

5.1 Introduction

In my literature study, I have researched a number of different robot simulation programs. However, many of the programs described in section 2.7 are either not available for free or offer simulations mainly designed for 3D physics. For 3D physics, this means that some form of 3D model has to be designed and often makes simulations more complex than necessary. Finally, the Unity engine was chosen to design the swarm simulation, using the 2D physics engine it offers.

5.2 Unity Engine

With some prior experience and inspired by the paper "*Swarm Robotics Simulation Using Unity*" [28], the game engine Unity was chosen for designing simulations for the Zebro swarm. The Unity engine is flexible, offers both 2D and 3D physics simulations and has an extensive documentation and many user problems have been solved online by its large community.

While certain functions need to be programmed manually such as camera controls, its flexibility also makes it easy to add functionality such as adding/removing Zebros and obstacles mid-simulation or reading out individual algorithm end results. Unity does not offer analysis tools, but simulation results can instead be written to an external file, and analysed in programs such as Matlab or Excel.

5.3 Virtual Zebro Design

5.3.1 Zebro Body

The Zebro design in the simulation makes use of Unity's 2D physics system and is modelled with a simple square collision box and a rigid body. This means, if a Zebro collides with a wall or another Zebro, they will not pass through each other but instead walls will block movement, and other Zebro collisions will cause the physics to play out in a more realistic manner. While technically the Zebros can climb over each other in real life, this is not possible in the simulation.

The rigid body is moved by applying forces, forwards for accelerating and backwards for decelerating. While the Zebros could move backwards, it is not allowed (or needed) in the simulation to keep the swarming simple. Rotation is done by rotating the rigid body in very small increments each update, meaning that it takes time for the Zebro to rotate a large angle. Furthermore, the rigid body has a linear and angular drag so that the Zebro will stop moving on its own when no forces are applied.

The current parameters give realistic looking movements, but they are not necessarily accurate for the real life model. It should still give a good representation of the effects of the swarming algorithms.

The Zebro is given an identification number between 1 and 255 and is assigned randomly. This number, the "ZebroID", is displayed on the grey box on the sprite (Figure 1).

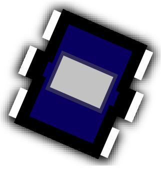


Figure 1: Zebro Sprite

5.3.2 Range Finding Sensor Designs

The sensors have been designed in two ways, one model is based on an early idea of using eight infrared sensors looking in all directions. The second model is based on a more recent omnidirectional design developed by a fellow group of students [43].

5.3.2.1 Fan-shaped Sensors

For the first design, a definitive version for range finding was not decided yet. It was speculated that the sensors would be based on infrared, aimed in all directions. This has been used in other swarm robot designs, such as the E-Puck and Swarm-Bots [40] [44].

The sensors of the virtual Zebro are modelled using ray casts. In computer graphics, ray casts are lines (rays) that are projected from a point towards a direction. When this ray cast collides with a collision box, it reports data such as which object it hits and at what distance. This information is stored as a data table, useable by the algorithms.

The initial design used 8 ray casts, meaning it had very little coverage. It was clear that this was not enough for swarming, and it will function better with as high a coverage as possible. By increasing the amount of ray casts, still evenly spread around, the swarming intelligence will work much better.

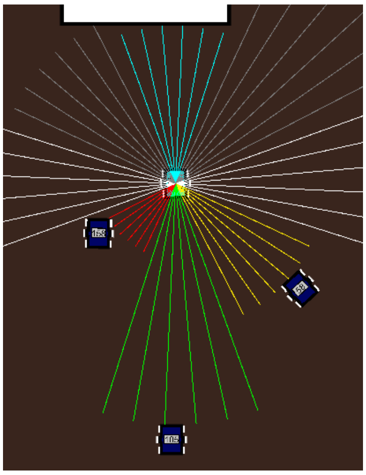


Figure 2: Fan-shaped Sensors

To go back to 8 sensors, the final data is combined to form eight fans of ray casts that would function as one sensor. It takes the closest detected obstacle and discards the other ray casts' information. As an example, when the sensors are modelled as 48 ray casts, each group of six would be combined. Thus grouping the ray casts allows for larger detection coverage around the Zebro while keeping the idea of the original sensor design.

5.3.2.2 Omnidirectional Sensing

When the bachelor group presented their design for their omnidirectional distance sensor, the previous design was changed to fit their method of detection. The omnidirectional sensor uses a combination of ultrasonic and radio, sent at the same time, and broadcast in all directions. By determining the direction of the ultrasonic pulse, and time difference between RF and ultrasonic, the distance and relative position can be determined. The ultrasonic pulse is sent and received using a unique cone-shaped antenna, pointing directly down onto the ultrasonic transducer [43].

To model this in Unity, every object within a set range is retrieved and stored. Taking their position in space and subtracting their own, the relative position and distance are found. The angle of the object can be calculated using the arctangent. Since every object is retrieved, this also includes objects behind other objects.

To prevent this from happening, the object data table is sorted based on distance. If any object is found within a specific angle and with larger distance, it is removed from the table, and ignored for further calculations. This way, attraction and repulsion are no longer happening in the same direction, as this causes Zebros to move closer to each other than intended.

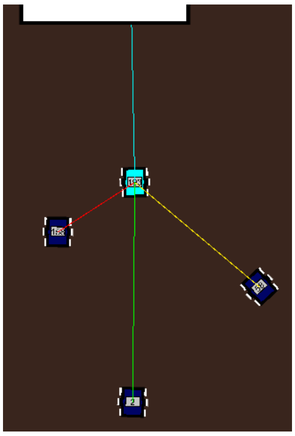


Figure 3: Omnidirectional sensor detections

The real life sensor is not able to detect walls, since it relies on receiving pulses from other Zebros. The wall detection can be done by infrared sensors instead. The detection of walls in the simulator is modelled using the method of the eight directional sensor.

5.3.2.3 Visualising detections

For debugging and clarification purposes, the detections are made visible with coloured lines. Based on the detected collision, the colour of these lines change to indicate which algorithm is used. For example, detecting a wall with a sensor at a close enough distance, the lines will colour cyan or blue. When detecting another Zebro, the lines will colour green for cohesion, red when separating and yellow when in between. Magenta is used to indicate a goal object is detected.

5.4 User Interface

The user interface is made to give some information about what is happening in the simulation. These elements have been added to increase the understanding of what is happening behind the scenes, and can greatly help finding issues when something is not behaving as expected.

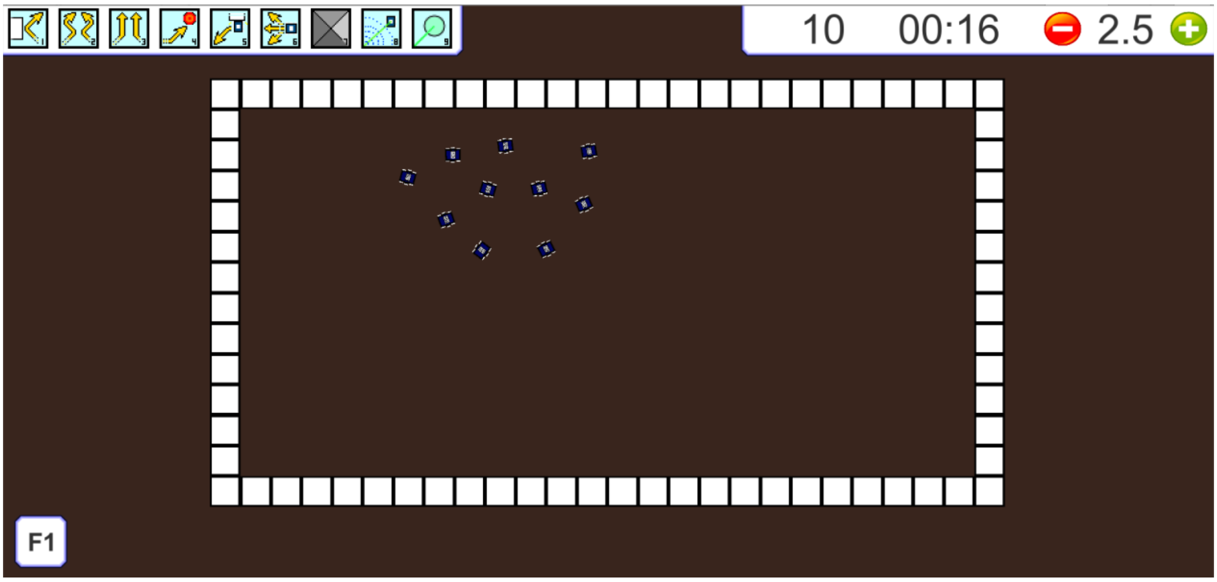


Figure 4: Simulation with User Interface

The icons in the top left of Figure 4 show which algorithms are currently active on all Zebros. They are coloured when active, and grey when turned off and they can be toggled with keys 1 to 9. From left to right, they represent (1) wall avoidance, (2) separation and cohesion, (3) alignment, (4) goal attraction, (5) unstuck mechanism, (6) all movement in general, (7) group colour algorithm, (8) switch between omnidirectional/fan sensor and (9) sensor noise. When unexpected behaviour occurs, these toggles can help identify the culprit. The bar at the top right displays the number of Zebros currently in the simulation, time passed and controls for simulation speed.

During the simulation, new Zebros and walls can be added and removed by clicking, to make testing quicker and more convenient. When adding or removing Zebros, the counter will update. The timer displays how long the simulation has been running for, based on the simulation speed, which can be found in the top right.

The result of the main algorithms can be read out on each Zebro individually by pressing left alt while hovering over with the mouse. In the white bubble, the vector results (rotation, speed) can be read out, from left to right: wall avoidance, cohesion/separation, alignment and final control result.

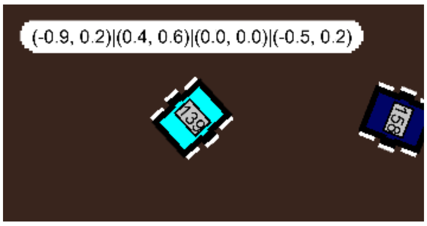

(-0.9, 0.2)|(0.4, 0.6)|(0.0, 0.0)|(-0.5, 0.2)

Figure 5: Reading out a Zebro

5.5 Virtual Test Environment

A number of arenas have been made to test the reaction of the swarm in different environments and swarm sizes. The main testing ground is a square room, which is used to test new additions to the code and to analyse the behaviour of the swarm when encountering a wall or corner. Obstructions in the second arena are placed to find out how the swarm breaks up or stays together when encountering a thin wall. The third arena simulates a smaller gap like a doorway, to see how well the swarm manages to walk through as a group. The swarm often breaks up because each Zebro sees the obstruction differently.

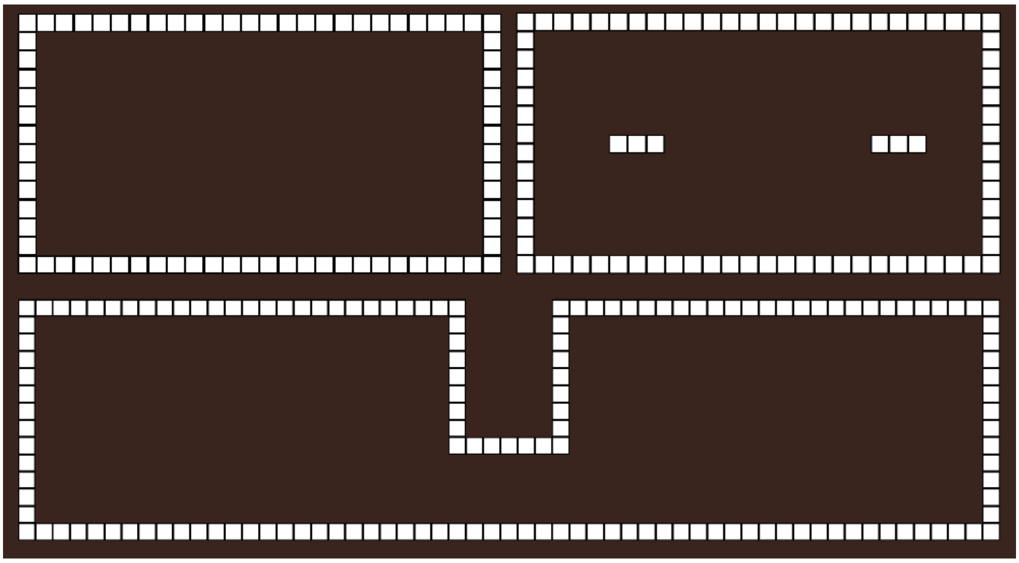


Figure 6: Examples of testing arenas. A square room, a room with thin obstructions, and a room with a “doorway”.

6. Swarming Algorithms

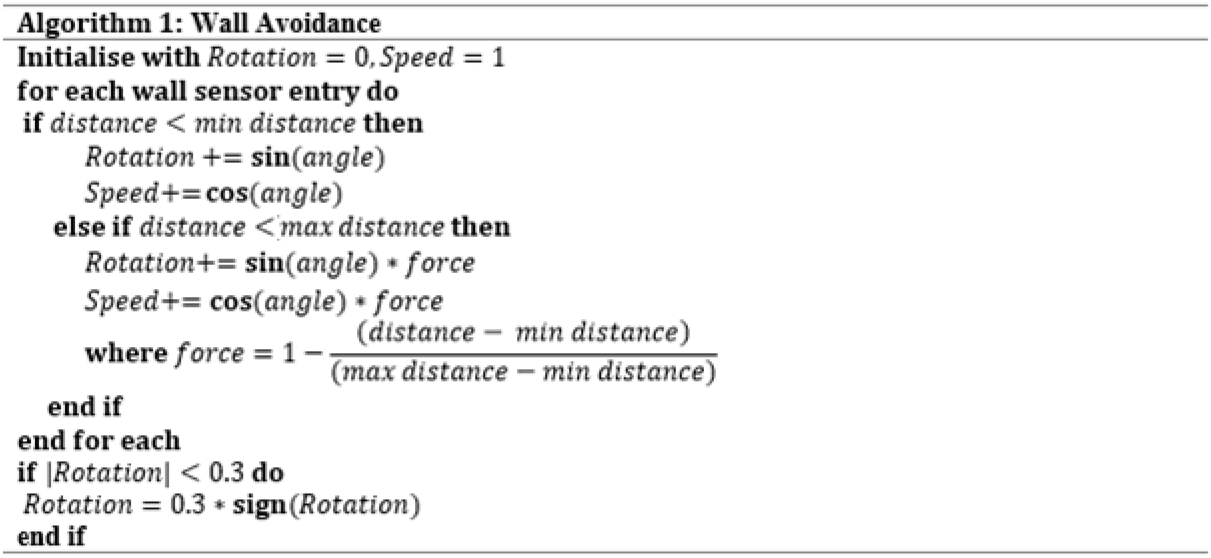
The algorithm for swarming is split into three main sub-algorithms, Wall Avoidance, Separation and Cohesion, and Alignment. Each algorithm provides a result based on the sensor data and the results of the algorithms are combined into a forward force and a rotation.

The desired speed is calculated as a value between 0 and 1, with 0 being stand-still and 1 top speed. The Zebro will accelerate or decelerate towards the provided value. While the Zebro is capable of walking backwards, it is designed to stop and rotate on the spot instead, until it is allowed to move forward again. The rotation is given as a value between -1 and 1, with -1 meaning a maximum rotation speed clockwise, 0 is no rotation, and 1 maximum speed counter clockwise. The rotation and speed are updated every time the screen is refreshed, at 60Hz, but can be set to run at any frequency (see 7.6). The code for the algorithms can be found in Appendix A.

6.1 Wall Avoidance

The first algorithm is used when the Zebro detects a wall within the set detection area, and is described in Algorithm 1. The detection has a minimum and maximum range; at the maximum range there is no action (0), and from there the action linearly ramps up to 1 at the minimum range. The resulting action is based on the location the wall is detected: an obstacle straight ahead will cause the Zebro to slow down, until it completely stops. The walls detected at the sensors on the side will cause the Zebro to rotate away. A minimum value has been added to force a decision more quickly and prevent the Zebro from getting stuck.

In the algorithm, the angle and distance are collected from the Zebro's sensors. The angle is the angle at which a wall is detected, and is relative to the heading of the Zebro. It is calculated in degrees with zero meaning straight ahead. After the calculations, if the resulting rotation and speed of the algorithm are smaller than -1 or larger than 1, they are set to -1 and 1, respectively.



```
Algorithm 1: Wall Avoidance
Initialise with $Rotation = 0, Speed = 1$
for each wall sensor entry do
 if $distance < min\ distance$ then
        $Rotation \mathrel{+}= \sin(angle)$
        $Speed \mathrel{+}= \cos(angle)$
    else if $distance < max\ distance$ then
        $Rotation \mathrel{+}= \sin(angle) * force$
        $Speed \mathrel{+}= \cos(angle) * force$
        where $force = 1 - \frac{(distance - min\ distance)}{(max\ distance - min\ distance)}$
    end if
end for each
if $|Rotation| < 0.3$ do
 $Rotation = 0.3 * \mathrm{sign}(Rotation)$
end if
```

The effect of the force can be summarised by the following plot, describing the relation of force to distance to the detected object.

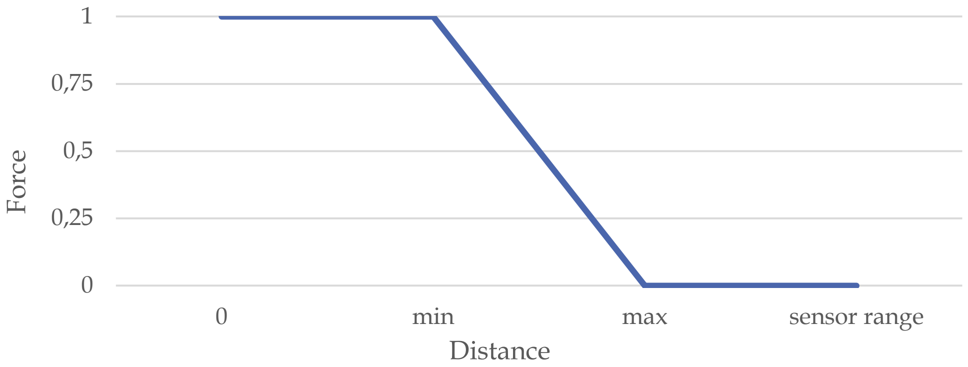


Figure 7: Distance to Force for the Wall Avoidance algorithm

6.2 Separation and Cohesion

The separation and cohesion algorithm is used when the sensors detect another Zebro. The algorithm tries to keep other Zebros at a certain distance by steering towards or away, or by slowing down, depending on the relative angle the Zebro is detected at.

The sensors can be split into three ranges; the outer ring is the attraction range: When another Zebro is detected in this ring, the Zebro is attracted towards the others to close the distance and stay together. In the inner ring, the Zebros are repelling each other to prevent collision. In between the inner and outer ring, the separation is considered to be at an ideal distance, and no action is taken.

The effect of the force can be summarised by the following plot, describing the relation of force to distance to the detected object.

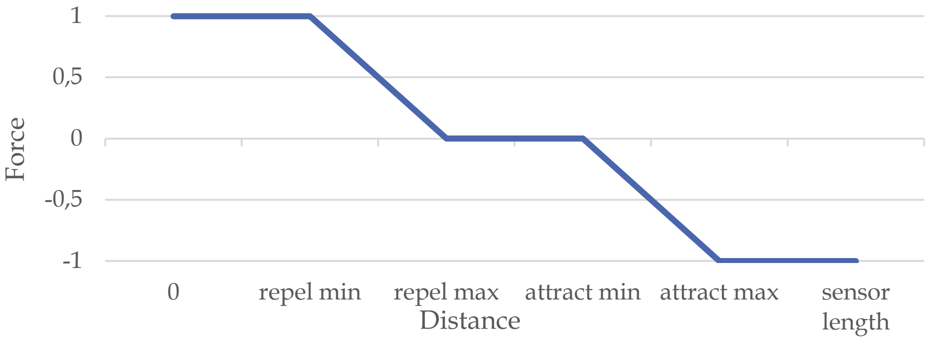
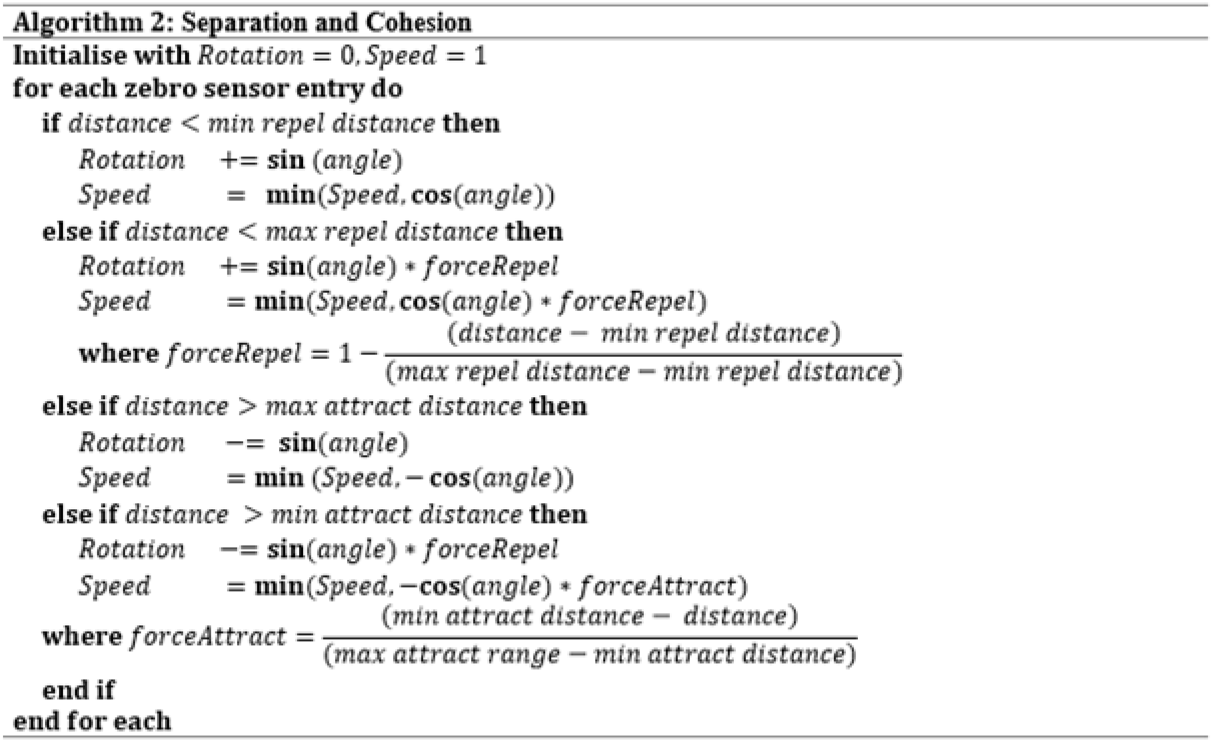


Figure 8: Distance to Force for the Cohesion and Separation algorithm

Algorithm 2 explains the calculations done to achieve the final result. The results are added together to ensure that the Zebro will be able to react strongly to just one neighbour, but also find a suitable balance with neighbours detected on both the left and right sides of the robot. The result is finally limited to a magnitude of 1 to prevent the algorithm from having a too large effect when combining the different algorithm results.

The angle and distance are collected from the sensors. The angle is the angle at which another Zebro is detected, relative to the current Zebro, where zero means straight ahead.

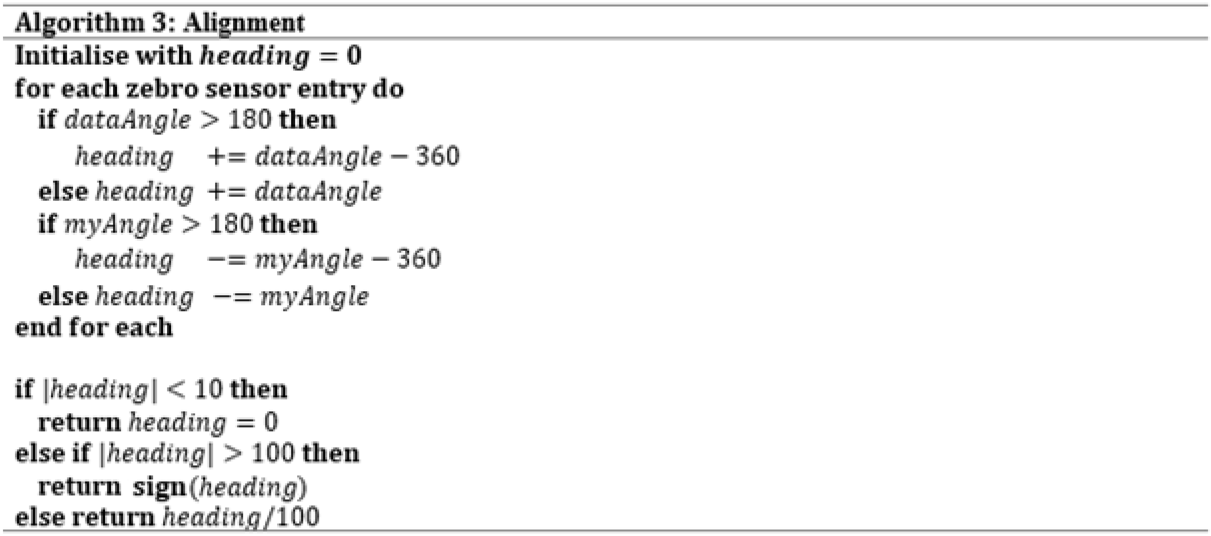


```
Algorithm 2: Separation and Cohesion
Initialise with $Rotation = 0, Speed = 1$
for each zebro sensor entry do
  if $distance < min\ repel\ distance$ then
    $Rotation \mathrel{+}= \sin(angle)$
    $Speed = \min(Speed, \cos(angle))$
  else if $distance < max\ repel\ distance$ then
    $Rotation \mathrel{+}= \sin(angle) * forceRepel$
    $Speed = \min(Speed, \cos(angle) * forceRepel)$
    where $forceRepel = 1 - \frac{(distance - min\ repel\ distance)}{(max\ repel\ distance - min\ repel\ distance)}$
  else if $distance > max\ attract\ distance$ then
    $Rotation \mathrel{-}= \sin(angle)$
    $Speed = \min(Speed, -\cos(angle))$
  else if $distance > min\ attract\ distance$ then
    $Rotation \mathrel{-}= \sin(angle) * forceRepel$
    $Speed = \min(Speed, -\cos(angle) * forceAttract)$
  where $forceAttract = \frac{(min\ attract\ distance - distance)}{(max\ attract\ range - min\ attract\ distance)}$
  end if
end for each
```

6.3 Alignment

With only the separation and cohesion algorithm, the problem arises that two Zebros would continuously move towards and away from each other in a zig zag pattern, since they only try to attract and repulse. For this algorithm, it is assumed that Zebros can share some basic information as long as they are in sensor range. The information taken from other Zebros in this case is their heading, for example from their compass. Taking this data from all nearby Zebros and averaging them, will give the average difference in heading. The Zebro will rotate until the average heading is less than 5 degrees off. When all nearby Zebros rotate towards this average heading, the group moves parallel in the same direction. This algorithm is only active when the total result of the wall avoiding and cohesion/separation algorithms combined is below a threshold. Without the threshold, the rotation against the wall avoiding or separation algorithms might cause an unwanted collision. This algorithm only results in a rotation.

In the algorithm, dataAngle is the detected Zebro's angle relative to the world, in degrees. myAngle is the angle of the current Zebro relative to the world, in degrees. The resulting heading is the difference in orientation compared to other detected Zebros.



```
Algorithm 3: Alignment
Initialise with $heading = 0$
for each zebro sensor entry do
  if $dataAngle > 180$ then
     $heading \ {+}{=} \ dataAngle - 360$
  else $heading \ {+}{=} \ dataAngle$
  if $myAngle > 180$ then
     $heading \ {-}{=} \ myAngle - 360$
  else $heading \ {-}{=} \ myAngle$
end for each

if $|heading| < 10$ then
  return $heading = 0$
else if $|heading| > 100$ then
  return sign$(heading)$
else return $heading/100$
```

6.4 Data Merging

After the above three algorithms, three values for rotations and two for speed are returned. The minimum speed value from the wall avoiding and cohesion/separation is used, because it is important to stop for any immediate collision threats. The rotations are added, meaning that it is possible for algorithms to cancel each other out.

The Alignment algorithm is only added to the results if the rotation result of the Wall Avoidance and Separation and Cohesion are between -0.4 and 0.4. This means that, when a Zebro is in a good spot within a swarm, it should try to head in the direction of all the others. However if there are obstacles to avoid or other Zebros it wants to move towards/away from, the Alignment algorithm is not included to ensure the other algorithms can fulfil their intended results.

A fourth algorithm is added called Goal Attraction, which will be discussed in 6.5.1. This algorithm is only added when the rotation of the previous three algorithms is between -0.6 and 0.6. This limitation tries to prevent collisions and reduce the chance that the swarm breaks up.

The values are given to the movement methods, that will cause the Zebro to move. If the final rotation is smaller than -1 or larger than 1, it is set to -1 and 1, respectively.

6.5 Additional Algorithms

A few other algorithms have been implemented to give the Zebros some kind of a goal to go after, or generally help the swarm.

6.5.1 Goal Attraction

In the simulation, a red goal can be added that, when touched, turns green and becomes invisible to Zebros, essentially marking it as "found" (Figure 9). When a goal node comes in range of a Zebro sensor, the Zebro is attracted to the goal. This Zebro then raises a flag that lets other Zebros know that it has found a goal. Other Zebros will follow this Zebro like a goal object. Only Zebros that directly see a goal will give themselves the flag and become 'attractive', while those looking at a Zebro with the flag will not. Raising this flag will attract a larger part of the swarm towards the goal.

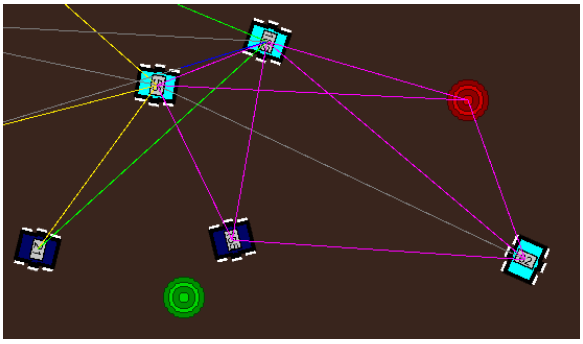


Figure 9: Zebros attracted to a goal (red)

6.5.2 Returning to Base

Eventually, the swarm has to be collected after tests or when their task is complete. To try and pick up every Zebro individually each time is a painful task and it would be helpful to be able to send the entire group towards a convenient location.

In the simulation, the entire swarm can be rotated towards a specific point. When a button is pressed, all Zebros will stop moving, get their distance towards this point and start a timer of about three seconds. In real life finding this distance could be achieved by estimating the Bluetooth signal strength of a broadcast. Making the swarm stop in place shortly will make sure the distance measurement is most accurate, and allows the swarm to rotate in any direction. While the swarm comes to a halt, each Zebro will look at the distance measured by their immediate neighbours.

When the distance they detected is smaller, the neighbours will become 'attractive' and the Zebro will rotate to look at it. The opposite happens when the neighbour distance is larger, the Zebro rotates to look away. On average, the swarm will rotate in the general direction towards where the broadcast was done. After the timer reaches three seconds, the swarm continues with the original algorithms and will try to align, pulling the Zebros with a different heading into the same direction.

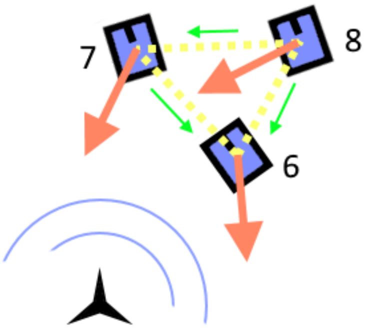


Figure 10: Visualization of algorithm. Numbers 6, 7, and 8 are the estimated distances to the source.

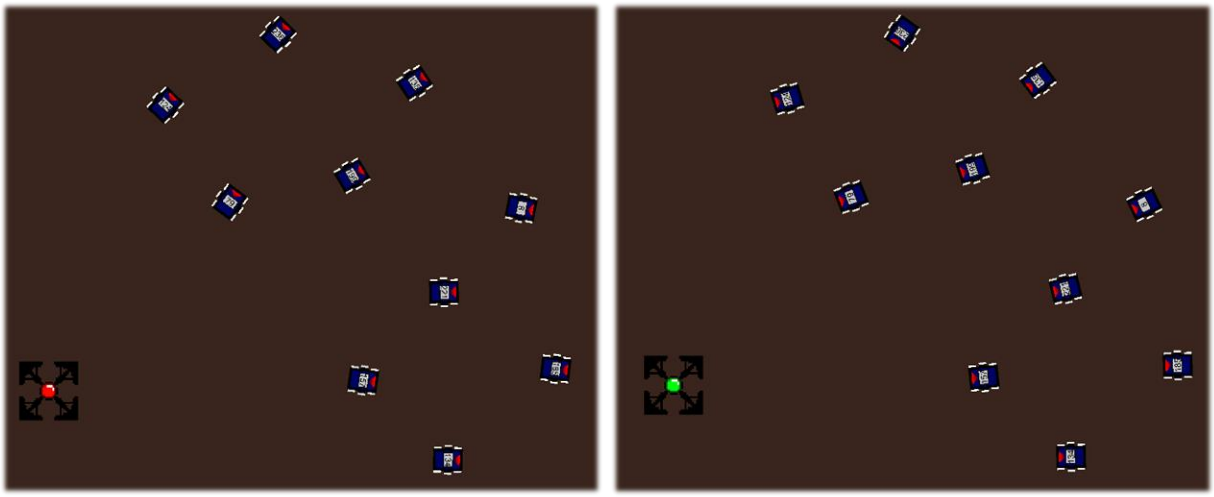


Figure 11 Before and three seconds after the algorithm. The front is made red for clarity.

6.5.3 Returning to Swarm

When encountering obstacles, it is possible for the swarm to split up into two or more groups. The swarm can generally function with less members in each group, but a single Zebro cannot. This algorithm aims to try to reunite a lone Zebro when it has been split off of the rest. When a Zebro has not detected any other Zebros for a few seconds, it will stop and directly rotate to the direction where it has last detected another Zebro. It is not guaranteed that it will find back the rest of the swarm in all situations.

An option for the simulation is to let a lone Zebro break down instead of turning. This can be used for testing, to test the performance over a longer period of time, and determine how many are still operational after a given period of time.

6.5.4 Group Colours

To give a visual identification when a group splits up, this method will colour the sprite of the Zebro based on the lowest ZebroID detected within its local swarm. At a set time interval, all Zebros will read out the ID of their immediate neighbours, and if it is lower than their currently held ID, it will use that instead. The lowest local ID will propagate through the swarm and after a set number of rounds, all Zebros will display a colour with the (HSV) hue determined by the ID. After a pause, the ID is reset to its originally assigned value, and the process is repeated.

Because the IDs are assigned randomly in the simulation, it’s possible for two swarms to have the same lowest value and thus the same colour. The IDs are randomized for demonstration and to give the Zebros a visual distinction.

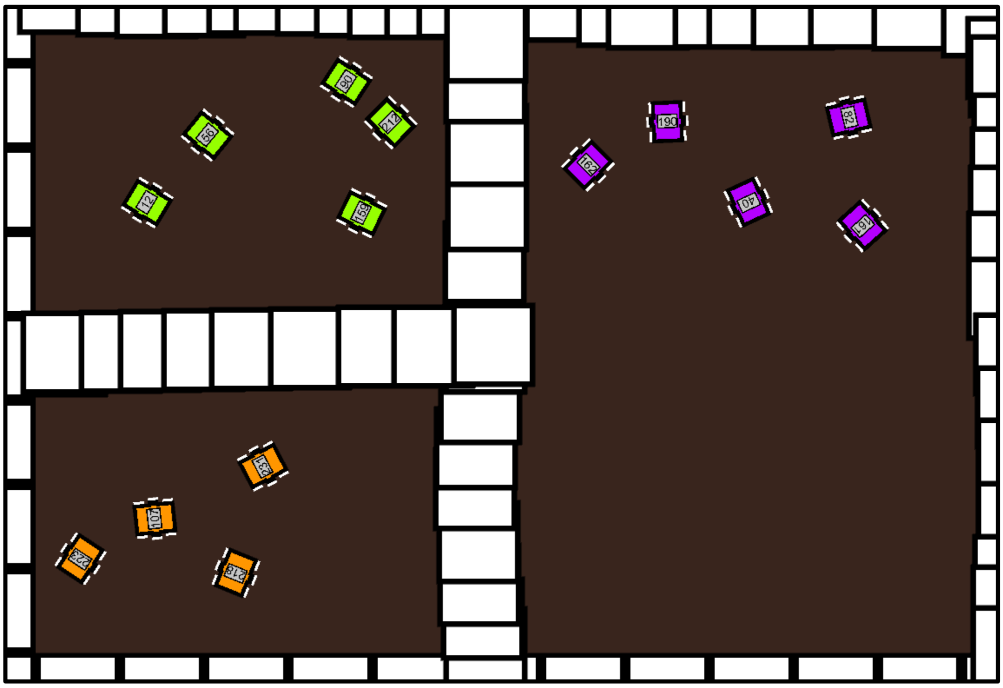


Figure 12: Zebros taking different colours

7. Simulation Results

7.1 Introduction

A swarm does not have an ideal way of behaving, and quantifying how well the group "swarms" is difficult to determine in numbers. Ideally, the group of robots should stay together, but the group splitting up can still be desirable depending on the application. In all cases however, the robots should not wander off on their own, since a single robot does not have the capability to perform their task.

The swarm has been experimented on in a number of different ways. Tests have been done with different sizes, from 10 up to 100 robots to determine whether the swarm will group up as expected when more members are added. The environment has been changed to find if the swarm would split up, swarm around one specific object or find a different route.

To give some form of qualitative measure to the swarm's performance, one could measure the average speed of the centre of the swarm over time to determine the distance covered. Another test finds the biggest distance from the centre to the outermost robot to find their grouping density. The results of these experiments have been described and summarised here.

7.2 Observations

During the development of the simulation, Some parts of the Zebro design have been tweaked depending on how it performed. During this time, some observations have been made.

The initial tests have been done using the fan-shaped sensors. A feature of this model is that each sensor detects the first object it sees and the objects behind it are not detected. The first version of the omnidirectional sensor detected all objects within its radius, including those that would be hidden behind other objects. In large swarms, this caused Zebros to both attract and repel in the same direction, making the swarm too packed together.

7.3 Performance Analysis

This method was created to find how the performance of the swarm changes with swarm size. The performance is measured by determining the average speed of the centre of the swarm. To keep the results consistent, the same type of formation and starting angle is used at the start for each group. The square starting formation has a size of n^2, $n \in \{3, 5, 7, 9\}$, starting with 9, 25, 49 and 81 Zebros.

The swarms are deployed in the same large room, and each swarm size is tested 5 times. The centre speed recorded once per second, and written to a text file so that it can be easily used in a spreadsheet for evaluation. The tests are done with and without walls: Tests with walls are running for three minutes, without walls for one minute to allow the swarm to align and explore. The tests with walls will give an indication how quickly a swarm recovers from encountering an obstacle, while a clear field shows the performance impact of separation and cohesion while (mostly) aligned. Due to the small variations in alignment, the Zebros can detect different neighbours over time, causing them to steer to repel or attract. These first tests are done without introducing any kind of noise to the sensor data and sensor information is updated continuously. Due to the nature of a swarm, a small difference in position can completely change the results of the simulation. The results have been averaged because events happen around the same point in time, and give an indication of the response of each swarm size.

7.3.1 Performance Test: Fan Sensors

This section discusses the performance of the infrared-inspired sensors. This model has been tested for a longer time, and has given promising results.

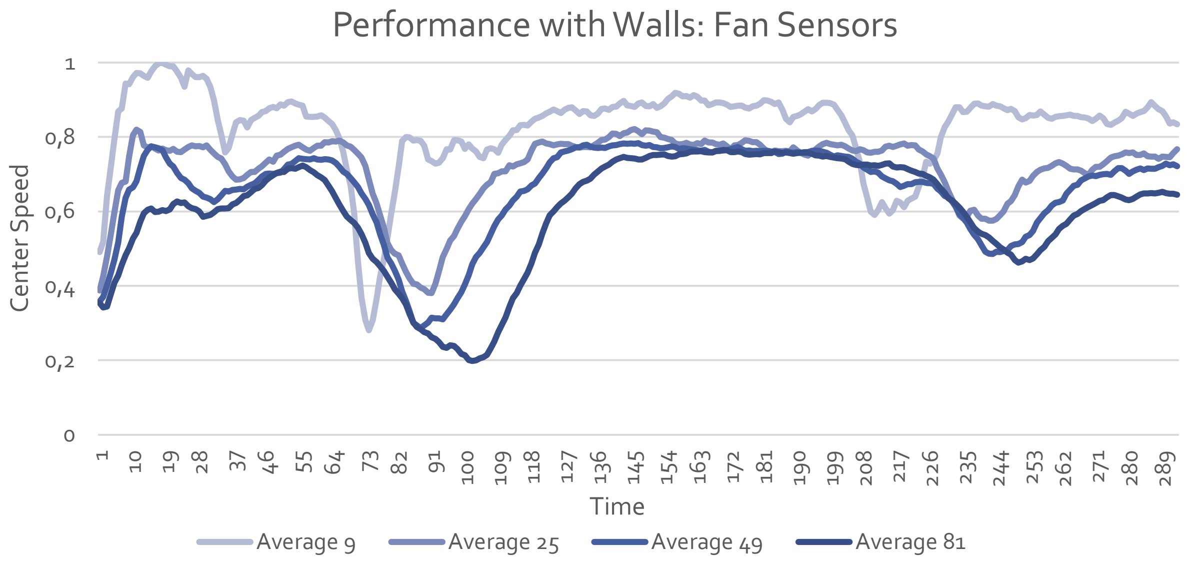


Figure 13: Graph showing performance over time, within arena

The first graph in Figure 13 displays the average speed of all Zebros over time. After about 60 seconds, the swarm encounters a wall. When a small group of Zebros encounter a wall, the group slows down, turns and quickly moves again. With larger swarms, the time before the swarm has turned away takes longer. To get back to full speed, they need to reform their group as well, finding an equilibrium between each other. The swarm size of 81 takes about 75 seconds before they are moving at their original speed, while the swarm of 9 Zebros recovers in about 15 seconds.

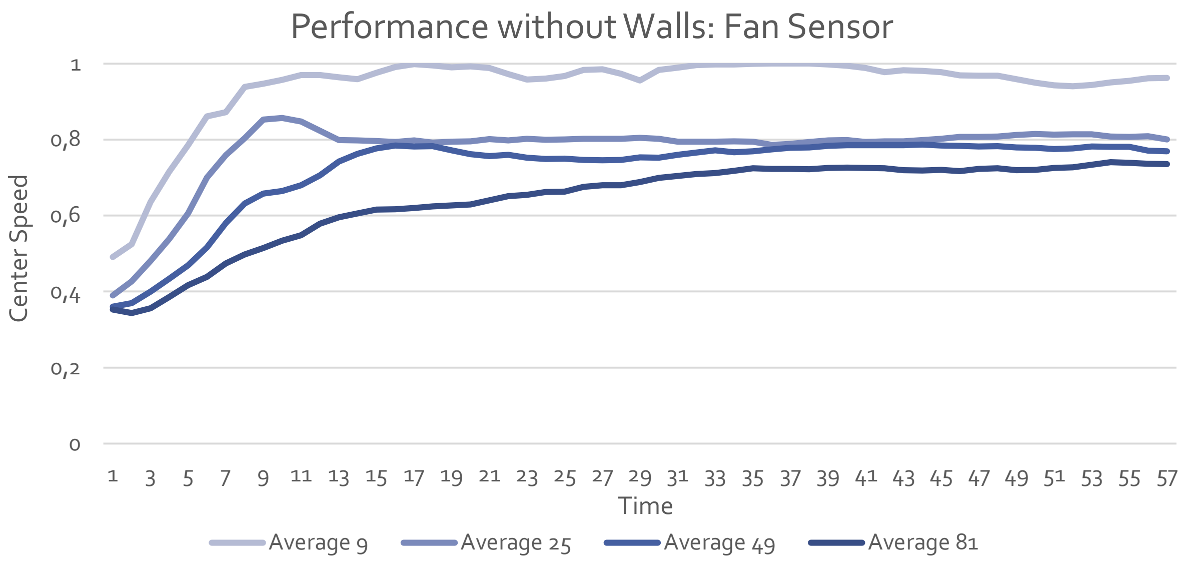


Figure 14: Graph showing performance over time, without obstructions

The second graph (Figure 14) is the performance within the swarm when no obstructions are encountered. The swarm tries to find a distance equilibrium between each other. Zebros at the front of the swarm occasionally detect those further behind it, which causes the swarm to slow down.

7.3.2 Performance Test: Omnidirectional Sensor

The same test has been done for the omnidirectional sensor design. The parameters have been slightly adjusted for the difference in detection. The number of neighbours has been set to 5 because of the better overall performance.

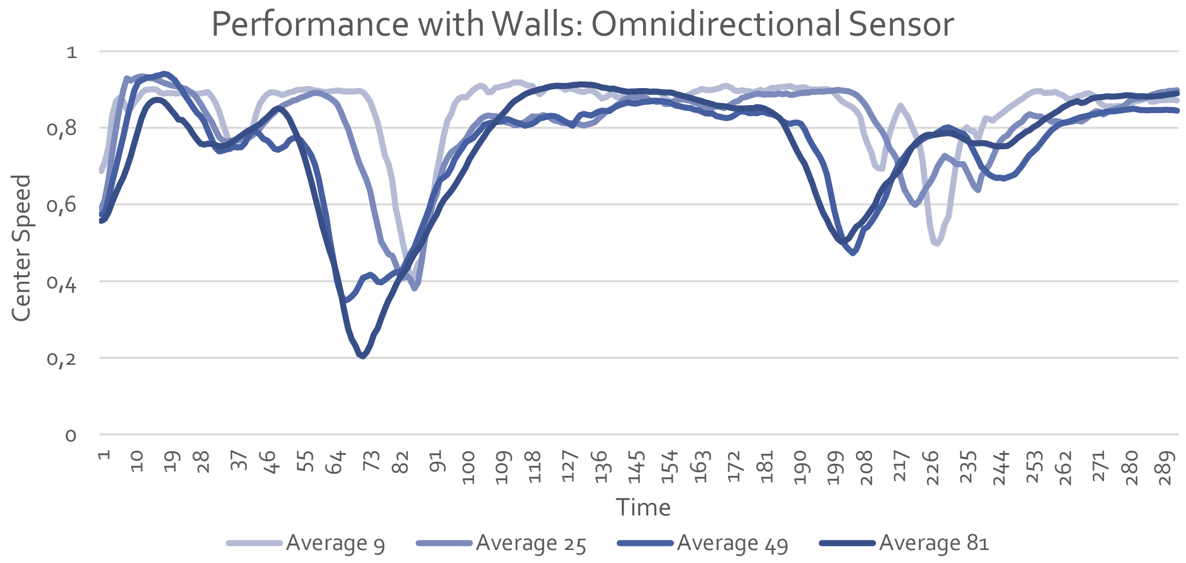


Figure 15: Graph showing performance over time, within arena

The results in Figure 15 show that the overall performance of the omnidirectional sensor is more consistent. The swarm is able to find an equilibrium more quickly, with the 81 size swarm recovering in 45 seconds after finding a wall. The difference in consistency from the fan sensors could be caused by the fact it can only detect in eight directions, and when a detected Zebro enters another section's range, the effect on movement can be large. For the omnidirectional sensor, this detection difference is smaller and continuous, meaning no sudden big changes in movement resulting in a more consistent movement pattern.

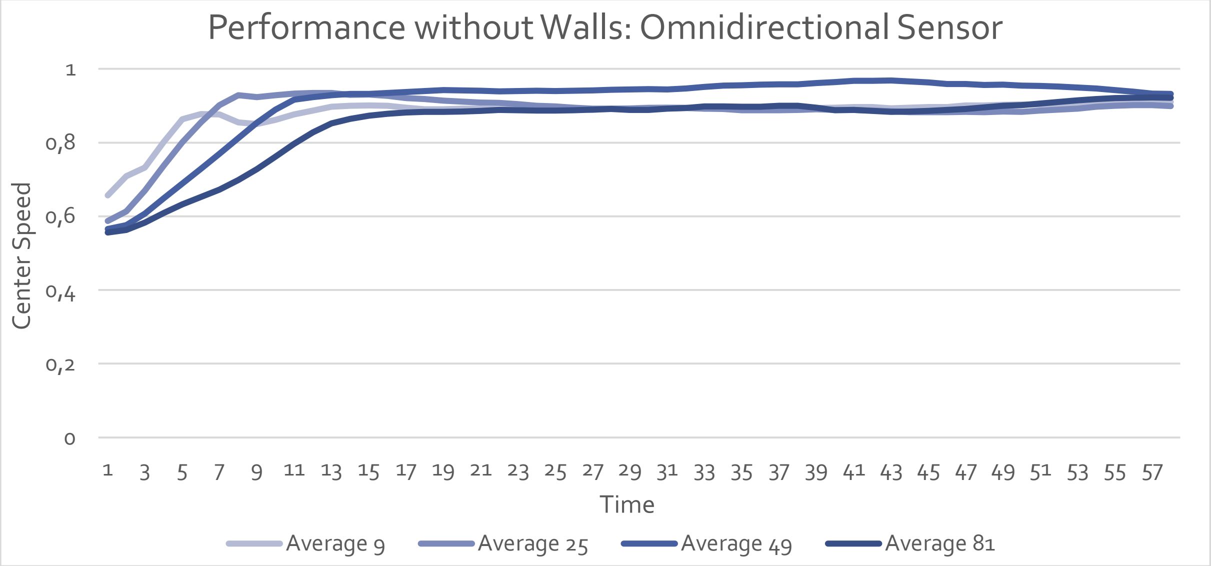


Figure 16: Graph showing performance over time, without obstructions

Without obstructions (graph in Figure 16) , the swarm spreads out more quickly than with the fan sensor design, and remains at a higher speed with larger swarms. An important reason the fan sensors had lower performance, is due to the detection of Zebros further back. In their position, that detection did not have much relevance. By only considering the closest neighbours with the omnidirectional sensor, this detection is ignored and the slowdown is not applied. The effect of changing the visible neighbours at any time will be further investigated in section 7.4.

7.4 Number of Neighbours

For one robot within the swarm, it's not necessary to keep track of all Zebros picked up by the sensor. When another Zebro enters on the edge of the sensor detection range, its attraction force will be large, causing the two to rotate towards each other. This can create a disturbance within the swarm, causing it to slow down, change direction or change formation. However, looking at too little Zebros at any moment makes it easier for the swarm to break up or cause collisions.

In this test, the omnidirectional sensor design is used to test the effects of lowering the number of neighbours visible at any moment. The closest neighbours are taken, and the ones further away are ignored. Testing has been done from 1 to 6, and all visible, as the maximum neighbours at any time.

With 1 and 2 neighbours, the group splits up quickly. Even when groups stay close at first, the group does not align and slowly diverges. When individual groups meet, collisions often happen due to the slow response. With 3 and 4 neighbours the swarm operates as expected in small groups, but it is not enough to keep a larger swarm (50+ robots) together. While collisions no longer happen, the swarm breaks up into smaller groups too easily.

With 7 or more visible neighbours in small groups, the swarm stays together and aligns as expected. Due to sight being obstructed by other robots, Zebros further away are ignored, but occasionally a gap opens up, causing a Zebro on the edge of the swarm to steer sharply into the group. This becomes a more common problem in large swarms, and causes a lot of individual robots to move around unexpectedly, slowing down movement even when the swarm is heading in one direction.

The ideal number of neighbours depends on the size of the swarm. For larger swarms, it appears to be 6: It gives enough information for larger swarms to align but limits long distance detections through the swarm. For smaller swarms (such as 10 Zebros), 4 maximum neighbours seems to perform better. Because the group is smaller, it does not need as much information about others to stay as a group.

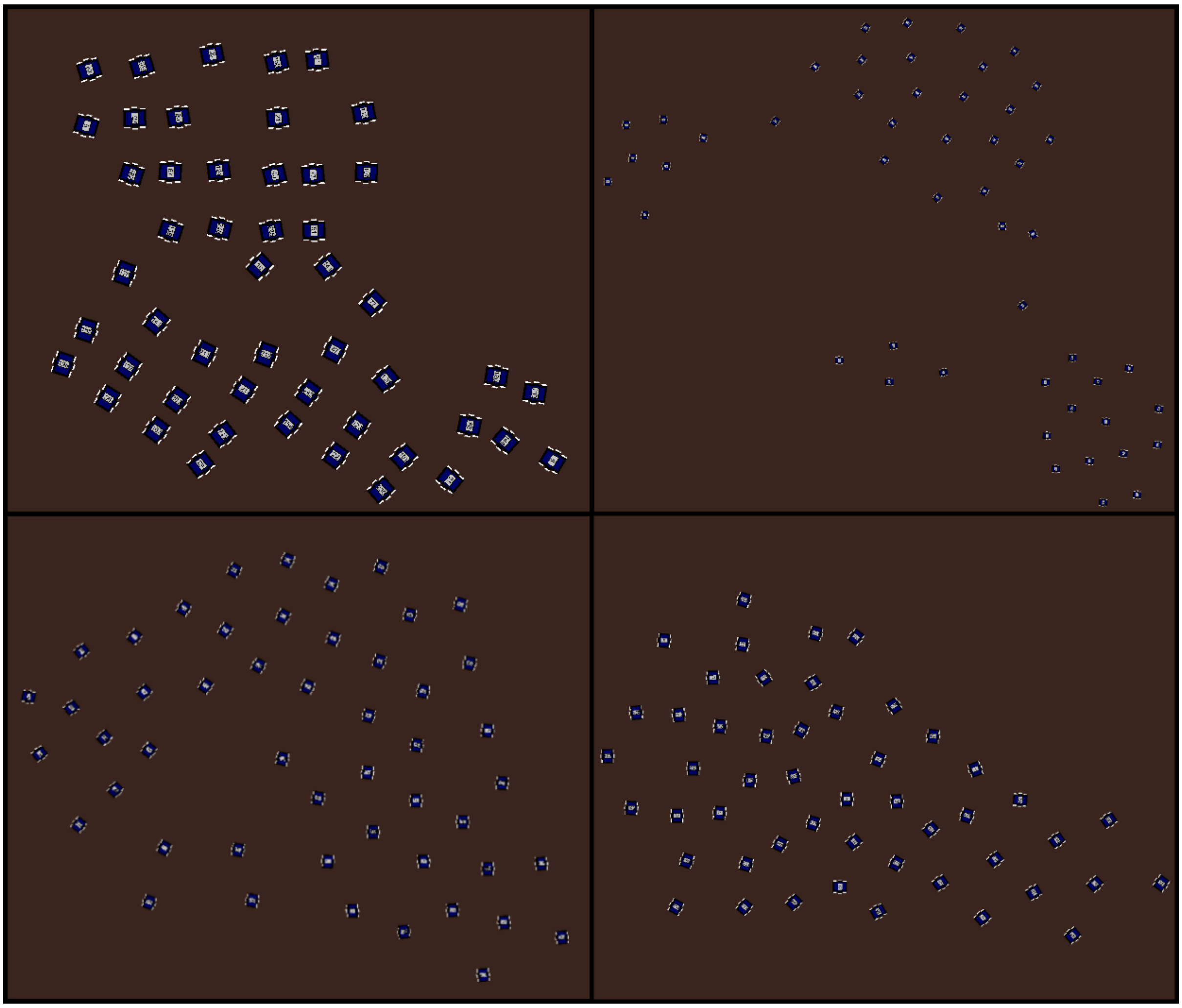


Figure 17: Comparison of cohesion with different numbers of visible neighbours, after 30 seconds. Top left: Starting position. Top right: 4 neighbours. Bottom left: 6 neighbours. Bottom right: All neighbours.

Figure 17 shows the resulting cohesion after 30 seconds after being placed as in the top left image. The starting position sets the Zebros closely together and some in different orientations, which forces the group to expand outwards and agree with a direction. With 4 neighbours (top right), the group is clearly separating. With 6 neighbours (bottom left), the group forms a much tighter group and aligns towards one direction. When detecting all neighbours (bottom left), the group forms a cohesive group, but many Zebros are turning into the swarm and the swarm does not find a state where all Zebros align as easily, thus slowing it down.

7.5 Effects of Added Noise

The sensors designed by the bachelor group have been estimated to have a constant 20 cm noise in detected distance. An experiment has been done with this form of noise implemented into the sensors. After the sensor information has been collected, the distances are increased or decreased at random with up to 10% of the sensors maximum distance value. The report does not specify the error for the angle of detection. The angle is increased or decreased randomly with up to 5 degrees. [43]

The noise does not seem to have much impact on the cohesion and separation behaviour. The alignment of each Zebro is communicated differently and is not affected. The noise causes small changes in direction, but are not large enough to cause a disturbance. The real life swarm is not expected to be hindered much by the noise from its sensors.

7.6 Effects of Algorithm Frequency

The simulation calculates the sensor data and updates the desired movement results continuously. This update frequency is not realistic for the real variant: the legs need time to turn and move the robot, the wall sensor has to rotate to pick up new signals from different directions, and the Zebros will not communicate constantly. This means there is a period of time where the algorithms work with old data and this can cause the Zebro to overshoot movements. This is an important difference with the real version, and this section investigates the effects.

The Zebros largely remain together, though breaking up becomes easier with longer time between calculations. With delays of 0.5 and 1 seconds, the swarm still operates mostly as expected. With longer delays, alignment becomes much harder however. The Zebros overshoot the general alignment causing them to rotate back and forth between sensor data refreshes. This can be partially solved by updating the previously calculated general heading with the robot's new heading while turning, to stop turning before receiving the updated sensor data. This is not done now because the sensor information is expected to refresh frequently enough.

With larger delays, collisions will also happen more often. This can be reduced by increasing the distance the Zebros want to stay away from each other. A delay of 1.5 seconds or longer make alignment in its current form almost impossible. The locomotion will have to be updated frequently enough to respond accordingly to changes to nearby Zebros and the environment. When this is not possible, the maximum speed of the Zebros can be lowered to give the sensors more time instead.

7.7 Effects of Sensor Coverage

Some experiments have been done by changing the sensor coverage. In early tests with few ray casts to represent the fan sensors, Zebros would be able to “hide” between ray casts and diverge. This lead to increase the amount of ray casts used to represent the fan sensors, until the coverage was high enough. When Zebros don't see each other, they do not act and this makes swarming difficult.

Another experiment was done using the omnidirectional sensor design: All Zebros detected behind, are removed from the sensor detections, and only detections in the front 180 degrees are used to act on. This will be of interest for the design of the Zebro range finding sensor.

Swarming can be done using only front detections, but this means that the Zebro leading the swarm has no idea what happens behind it. Thus if the swarm is steered away, the Zebros at the front will continue and the swarm becomes split up. Similarly, if a Zebro in the back becomes delayed, the rest will not slow down to let it catch up. Some issues occur when the swam reaches an obstruction, but this could be solved by modifying the behaviour.

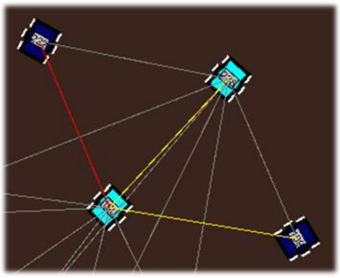


Figure 18: Zebros see only ahead

8. From Simulation to Zebro Controller

8.1 Introduction

With most of the simulation completed, the algorithms can be transported onto a controller. The design of the simulation has been split up in a modular way, so that each method is easily transferrable to most other languages. Information such as the sensor data or final result data that is sent to the locomotion controller use Unity specific libraries, since these parts will be replaced by their real-life counterpart and thus are not a part of our swarm intelligence. Instead, this data is used to visually represent the swarm in the engine.

8.1.1 Type of Processor

To connect all different components, the swarm intelligence has to be made portable. There are many options available on the market, which are divided into two main processor types: a microcontroller or a microprocessor.

A microcontroller is a chip that has all components needed to operate, built into one chip. It has a fixed amount of RAM and ROM, and other peripherals such as serial ports and digital/analogue IO. Microcontrollers are usually programmed using C since it stays closer to the controller's hardware limitations.

Microprocessors are much more general-purpose. It does not have any built-in components besides the CPU. RAM, ROM and peripherals are implemented separately on a circuit board. Thus the external components can be larger to suit more general applications. Microprocessors can run an operating system and support most object oriented programming languages, making it very versatile for development.

The calculations done for the swarm intelligence are relatively simple and don't require much processing time. The real Zebros move more slowly than in the simulation and the locomotion, sensors and communication need time to process their information. Processing power should not be a limitation.

The size of these modules should be taken into account, since there is limited space on the Zebro itself. Both boards can be made very compact. Pre-existing boards in this case are very attractive since no extra time has to be spent designing, testing and soldering a PCB. The Arduino [45] platform is very popular for microcontrollers, the Raspberry Pi [46] for microprocessors. Both platforms have multiple versions in different sizes and processing power.

In the future, more complicated tasks will be built upon the basic swarming code. OO languages would make programming of these modules easier and faster to develop, which would be a good reason to choose a microprocessor. If the application becomes much more complex, the modularity of a microprocessor also becomes advantageous.

For the initial testing, the Raspberry Pi Zero W was chosen. This version of the Raspberry Pi is made to be as small and cheap as possible. The non-Wifi version costs as little as 5€, which includes a circuit board with a number of modules, such as a connection for HDMI, two USB connectors and a slot for an SD card. However, the SD card is required as main storage device. The Wifi version costs a little more (11€) but allows easy access using SSH which makes programming and data transfer much more convenient. The cost of the board is low enough to be a good option for deploying them in a large amount of Zebros, and can be used when the Zebros functionality needs to be expanded.

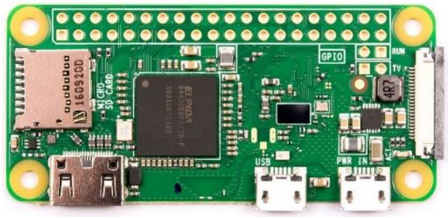


Figure 19: Raspberry Pi Zero W

8.1.2 Programming Language

For the Raspberry Pi, most OOP languages have been ported. The primary language is Python, that comes included with the Pi's Linux based OS, Raspbian, but as long as a language can be compiled for the Pi's processor, the ARMv6, it can be used. Java, C, C++, C#, Javascript, HTML5 and PERL are other available examples.

The choice of programming language mainly comes down to personal preference, as most programming language generally have the same capabilities. Python in this case, has the advantage that a lot of libraries have been written specifically for the Pi, which makes implementation easier when starting from scratch.

While not as commonly used as Python, C# Mono will be able to run on the ARMv6 processor and since the simulation is programmed in C#, it would make porting the code a lot simpler. New code can be quickly modified and tested on the simulation or on the physical Zebro. Because of these advantages, the C# language was chosen.

8.1.3 I²C Communication

To communicate between hardware in a single hardware system, a number of communication protocols have been designed. The communication on the Zebro uses the I²C: Inter-Integrated Circuit.

The I²C communication uses a Master-Slave system. The master sends out commands to connected Slave modules, controlling both sending and receiving. The slaves can only respond to these commands, and cannot start sending data to a master themselves. I²C uses only two bi-directional lines: SDA, the Serial Data Line, and SCL, the Serial Clock Line. The SCL is controlled by the Master, to synchronise the data transfer, and the SDA as the data transfer line.

Each slave in the system has an address that is used to select which module to communicate with. This setup allows multiple slaves and even masters to be connected to the same two wires. However, two masters cannot communicate, and collision should be prevented.

Using this in the Zebro, the Swarm controller is used as a master, requesting information from the localization and communication modules, and sending information to the locomotion controller. The communication timing is determined by the master, and it should be noted that information is not requested and sent too frequently. The other modules might not have information ready or processed yet. For example, the legs need time to rotate physically, so sending information multiple times within one-leg revolution is not useful.

Other forms of communication are UART and SPI. UART, Universal Asynchronous Receive/Transmission, only allows for communication between two modules. It uses two data lines, one for receiving and one for transmitting. Since communication is asynchronous, a data rate should be set beforehand. The inflexibility makes it less suitable for the Zebro communication.

SPI allows for multiple slaves (but not masters) in one system. It expands on the UART, using a data line for transmitting and receiving, but adds a SCK, Serial Clock to synchronise communication, and SS, Slave Select. The design needs a Slave Select wire for each new Slave, meaning the system needs more and more wires when new modules are added. While this hardware interface has higher transmission rate and range, they are not necessary, and the simpler setup of the I²C bus is a bigger advantage.

8.2 Raspberry Pi Algorithm Evaluation

The algorithms for cohesion, separation and alignment have been implemented on the Raspberry Pi Zero W, the Wifi variant of the Pi Zero. Using the Wifi channel, communication between the simulation and the Pi has been made using the TCP protocol, where the Pi connects as a client to the simulation. One of the virtual Zebros is being controlled by the code running in C# Mono on the Pi.

The sensor information is determined in the simulation and is stored in an array. This array is sent over Wifi to the Raspberry Pi, that then runs the algorithms. The speed and rotation are returned, and the virtual Zebro is moved accordingly. The resulting behaviour is as expected, and can’t be distinguished from the computer-controlled Zebros.

This setup follows closely the setup that will be used in the real Zebros. The sensors will provide their information through I²C in the same way. The end result will then be passed on to the locomotion controller, that makes the Zebros walk in the appointed direction and speed.

8.3 Zebro Wall Sensor Evaluation

While the Zebro communication is still a work in progress, the ultrasonic sensor used to detect walls is finished. Its operation and the Wall avoidance algorithm have been tested by letting a Zebro walk around the hallway. The sensor is set to detect obstructions up to a meter ahead, and it rotates left and right using a small servo, so that it can scan in a cone ahead. The speed of the servo can be adjusted, but needs to stop for the ultrasonic sensor to complete its range finding. With five measurement points, one scan round takes about three seconds, with the centre three scans taken twice.

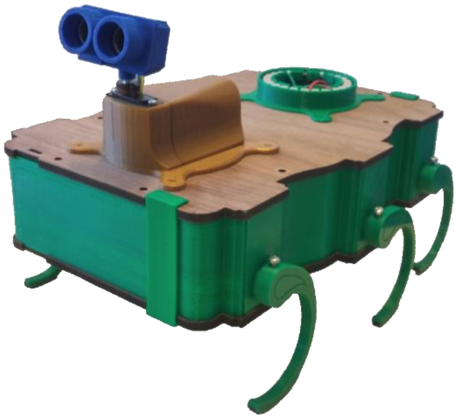


Figure 20: Zebro fitted with ultrasonic module

The Raspberry Pi is programmed with only the wall avoiding algorithm and the locomotion controller. Information between the two is shared using a temporary text file. This could be updated by using a set memory location instead.

The communication using I^{2}C, which communicates the sensor information from an Arduino Nano to the Raspberry Pi, and from the Pi to the microcontroller on the leg modules, has given a few problems. The Pi does not currently have libraries that allow for clock stretching, meaning it breaks up communication if the slave takes too long to respond after a Write command. The Pi has been able to communicate reliably with the leg modules after lowering the communication rate of the Pi. The communication with the Arduino Nano still sometimes fails, but by resending, all information will be received.

A Zebro has been equipped with the ultrasonic sensor, and has been put into a hallway. It walks forward until it encounters a wall and responds to it. The Zebro is capable of avoiding walls in a similar way as the simulation, with no collisions. The scanning in its current form is quick enough to make the Zebro stop and turn if something is placed a bit ahead of it. The ultrasonic sensor can detect other Zebros (as obstacles) which allows them to avoid each other. The ultrasonic sensor alone is not enough to achieve cohesion or alignment.

9. Conclusion

In the simulation, swarming is a success. Groups of Zebros will move to each other, keep enough distance and align using these algorithms, much like a flock of birds. The Zebro has been shown to avoid walls both in the simulation and during a physical test using an ultrasonic sensor, which has been very promising.

However, the fact that it has been achieved digitally, does not mean it translates accurately into real life scenarios. The mechanics of the real world Zebro are different, such as walking with legs that can cause changes in heading, sensors that take some time to process their information and other aspects such as battery and temperature management.

Because the Zebro communication module is not finished yet, testing the algorithms on physical Zebros is not possible yet. The algorithms will have to be tested extensively in physical experiments to find the problems that these mechanics may cause. The alignment algorithm could overshoot its rotation when the movement and/or sensors are not updated frequently enough. This could be remedied by lowering the rotation speed when nearing the alignment angle, so that the overshoot will be reduced. The real life Zebro moves and rotates slower than in the simulation, so it needs to be tested whether this change is needed.

The simulation itself can easily be modified and further built upon, when the algorithms need to be updated or if the Zebros need to be tested for a more specialised task. New parts can be added and combined with the existing code. The Unity files for the simulation will be available at the Zebro Project.

10. Zebro Leg Module

Aside from the swarming focus of my thesis, I have assisted with the development of the leg modules of the Zebro. The PCBs have been designed by Lisanne Kesselaar, an electrical engineering student at the TU Delft and member of the Zebro team. Steyn Huurman had started working on the software in C++, but was unable to finish it in time. This chapter is a summary of its design and my contribution to its software.

The PCBs are fitted with an ATXMega32A4U microcontroller, a temperature sensor connected through an internal I^{2}C line, a current sensor connected to the AD Converter, an USART connector for serial communication and a brushless DC motor, that is controlled through an H-Bridge with a PWM signal. The motor position can be set in a specific position using a photo encoder and encoder wheel with slits, and a reference point is made using a Hall sensor and a magnet slotted into the leg. Switches on the PCB can be set to indicate the location within the robot (Front, middle back, left or right).

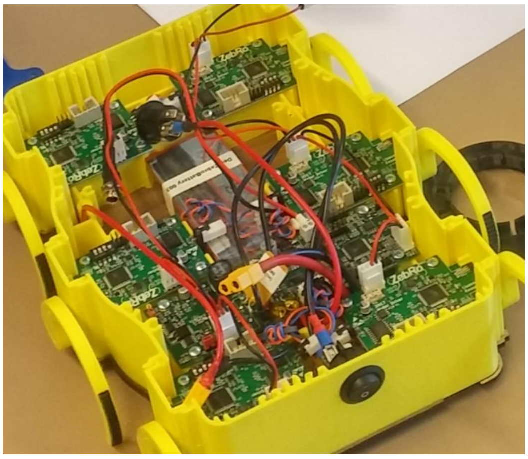


Figure 21: Inside the Zebro: 6 leg modules, battery and battery management system

The PWM signal that drives the motor is controlled with a PID controller. This controller works by setting a reference signal, in this case the desired position (set point) to rotate to. The input to the PID is the error: the set point minus the current position. The Proportional effect of the controller is directly based on the error: a larger error means larger proportional control. The Integral term wants to minimize the error remaining after the proportional control, increasing in effect when the P term shrinks. The Differential term is based on the speed of change in error, with larger delta error causing larger effect. This has a dampening effect.

The PID controller also uses time as an input. This is used to gradually move the set point to the final desired location. This way the turning speed can be slowed down, since the error to the shifting set point is lower than driving directly to the desired position. The P, I and D terms of the controller have been tuned by Laurens Kinkelaar, who has designed the locomotion controller.

The microcontroller is programmed to use interrupts to handle encoder updates and I^{2}C communication upon receiving data. The PID controller executes at a set frequency using interrupts from the real-time counter. While not handling interrupts, the microcontroller picks up information from the temperature sensor, and uses the ADC to read the current.

The I^{2}C is used to send and request information to and from the leg module. Since the module is set up as a slave, it can only respond to messages from a Master, in this case Laurens' locomotion controller. The first byte sent is the status, which determines whether the microcontroller should listen for more information to change the set point, or to load information into its read buffer. Writing can be followed up by a desired position, direction (clockwise, counter clockwise) and time. Reading can return the temperature, current or the motor's current position. Setting certain states can also indicate to stop the motors or reset using the Hall sensor.

Finally, the motors are reset to their neutral position when starting up. This ensures that the legs will be set in the correct position whenever the robot is powered up.

11. References

[1] "TU Delft Robotics Institute - Zebro," [Online]. Available: https://tudelftroboticsinstitute.nl/robots/zebro.

[2] K. Kelly, "Out of Control," http://kk.org/mt-files/outofcontrol/index.php, p. Ch2: Hive Mind.

[3] "Self-organization," [Online]. Available: https://en.wikipedia.org/wiki/Self-organization.

[4] A. Jevtic and D. Andina, "Swarm Intelligence and Its Applications in Swarm Robotics," *6th WSEAS Int. Conference on Computational Intelligence, Man-Machine Systems and Cybernetics, Tenerife, Spain,* December 14-16 2007.

[5] D. Corne, A. Reynolds and E. Bonabeau, "Swarm Intelligence".

[6] H. Wang, "Swarm Intelligence: Simulating Ant Foraging Behaviour," *MSc CAVE,* 2010.

[7] S. Roy, S. Biswas and S. S. Chaudhuri, "Nature-Inspired Swarm Intelligence and It's Applications," *I.J. Modern Education and Computer Science,* pp. 55-65, 2014.

[8] Himani and A. Girdhar, "Swarm Intelligence and Flocking Behaviour," *International Journal of Computer Applications,* 2015.

[9] H. G. Tanner, A. Jadbabaie and G. J. Pappas, "Stable Flocking of Mobile Agents, Part 1: Fixed Topology".

[10] Wikipedia, "Particle Swarm Optimization," [Online]. Available: https://en.wikipedia.org/wiki/Particle_swarm_optimization.

[11] X. Hu, "Particle Swarm Optimization," 2006. [Online]. Available: http://www.swarmintelligence.org/index.php.

[12] "Ant Colony Optimization Algorithms," [Online]. Available: https://en.wikipedia.org/wiki/Ant_colony_optimization_algorithms.

[13] A. Colorni, M. Dorigo and V. Maniezzo, "Distributed Optimization by Ant Colonies," *PROCEEDINGS OF ECAL91 - EUROPEAN CONFERENCE ON ARTIFICIAL LIFE.*

[14] Stützle, M. Dorigo and Thomas, Ant Colony Optimization, Cambridge, Massachusetts / London, England: MIT Press, 2004.

[15] Y. Tan and Z.-y. Zheng, "Research Advance in Swarm Robotics," *Defence Technology 9,* pp. 18-39, 2013.

[16] C. Melhuish, A. B. Sendova-Franks, S. Scholes, I. Horsfield and F. Welsby, “Ant-inspired sorting by robots: the importance of initial clustering,” 2005 Sep 16.

[17] “Player Project (Player/Stage/Gazebo),” [Online]. Available: http://playerstage.sourceforge.net/.

[18] “Homepage of UberSim,” [Online]. Available: http://www.cs.cmu.edu/~robosoccer/ubersim/ .

[19] “USARSim on SourceForge,” [Online]. Available: https://sourceforge.net/projects/usarsim/.

[20] S. Carpin, M. Lewis, J. Wang, S. Balakirsky and C. Scrapper, “USARSim: A Rbot Simulator for Research and Education”.

[21] “Homepage of Enki,” [Online]. Available: http://home.gna.org/enki.

[22] “Homepage of WeBots,” [Online]. Available: https://www.cyberbotics.com/webots.php .

[23] “Homepage of breve,” [Online]. Available: http://www.spiderland.org/s/.

[24] “Homepage of V-REP,” [Online]. Available: http://www.coppeliarobotics.com/.

[25] “Homepage of ARGoS,” [Online]. Available: http://www.argos-sim.info/index.php.

[26] “Homepage of MORSE,” [Online]. Available: https://www.openrobots.org/morse/doc/latest/what_is_morse.html.

[27] “Homepage of TeamBots,” [Online]. Available: http://www.cs.cmu.edu/~trb/TeamBots/.

[28] B.-S. Le, V.-L. Dang and T.-T. Bui, “Swarm Robotics Simulation Using Unity,” 2014.

[29] “Unity,” [Online]. Available: https://unity3d.com/.

[30] “What Is Bluetooth Wireless Networking?,” [Online]. Available: https://www.lifewire.com/definition-of-bluetooth-816260.

[31] “Difference between Bluetooth and Zigbee Technologies,” [Online]. Available: http://www.engineersgarage.com/contribution/zigbee-vs-bluetooth.

[32] “Infrared communication for the E-Puck robot,” [Online]. Available: https://www.youtube.com/watch?v=ngGDHwUJxpQ.

[33] Eric, “Society of Robots,” [Online]. Available: http://www.societyofrobots.com/member_tutorials/book/export/html/71.

[34] [Online]. Available: http://www.societyofrobots.com/sensors_sonar.shtml.

[35] B. Khaldi and F. Cherif, “An Overview of Swarm Robotics: Swarm Intelligence Applied to Multi-robotics,” *International Journal of Computer Applications,* no. Volume 126 - No. 2, pp. 31 - 37, September 2015.

[36] I. Navarro and F. Matia, “An Introduction to Swarm Robotics,” *ISRN Robotics,* vol. 2013, no. Article ID 608164, p. 10, 2012.

[37] M. Rubenstein, C. Ahler and R. Nagpal, “Kilobot: A Low Cost Scalable Robot System for Collective Behaviours”.

[38] F. Arvin, J. Murray, C. Zhang and S. Yue, “Colias, An Autonomous Micro Robot for Swarm Robotic Applications,” 2014.

[39] Matia, I. Navarro and Fernando, “An Introduction to Swarm Robotics,” *ETSI Industriales, Universidad Politécnica de Madrid, c/José Gutiérrez Abascal, 2, 28006 Madrid, Spain,* vol. Volume 2013, no. Article ID 608164, p. 10, 2012.

[40] “e-Puck Hardware,” [Online]. Available: http://www.e-puck.org/index.php?option=com_content&view=article&id=22&Itemid=13.

[41] M. Ceelen, C. v. d. Geer, F. Rouwen and S. Seuren, “Fundamentals for an autonomous Zebro Swarm,” TU Delft, 2015.

[42] “Zebro Project,” TU Delft, [Online]. Available: https://www.tudelft.nl/d-dream/teams/zebro-project/.

[43] K. Kouwenhoven and S. G. Verkamman, “Development of an Omni-directional distance sensing system for the DeciZebro,” TU Delft, 2017.

[44] S.-B. Project, “Swarm-Bots Hardware,” 2014. [Online]. Available: http://www.swarm-bots.org/index.php@main=3&sub=31&conpage=sbot.html.

[45] “Arduino,” [Online]. Available: https://www.arduino.cc/.

[46] “Raspberry Pi,” [Online]. Available: https://www.raspberrypi.org/.

[47] A. Campo, “Efficient Multi-Foraging in Swarm Robotics,” *IRIDIA Technical Report TR/IRIDIA/2006-27,* 2006.

12. Appendix A

This is part of the simulation code, specifically the methods that describes the cohesion, separation and alignment algorithms, as used in the simulation.

```
class SensData
{
    public Collider2D Collider { get; set; }
    public float Distance { get; set; }
    public float Angle { get; set; }
    public Vector2 Obstruction { get; set; }
}

    // ==== The main controller, calling other functions in order. Called in Update, thus every frame, or can be invoked at a certain interval.
    void ZebroCoreController()
    {
        if (SimController.enableFanSensor) ZebroFanSensors();
        else
        {
            ZebroOmniSensors();
            ZebroWallSensors();
        }

        if (SimController.enableSensErrors) AddSensorErrors();



        //After considering the wall collisions and detection of other zebro's, determine
        //a direction and speed for movement for the next frame.

        zFinalDir = Vector2.zero;
        zDir1 = Vector2.zero;
        zDir2 = Vector2.zero;
        zDir3 = Vector2.zero;
        zDir4 = Vector2.zero;

        //This section should be combined more elegantly. How and when do we combine certain sensor data?
        if (SimController.enableWallAv) zDir1 = AvoidWalls(); //This function tries to avoid walls. Modifies x and y.
        if (SimController.enableZSwarm) zDir2 = SwarmingBehaviour(); //This function tries to keep Zebros together, or disperse when too close. Modifies x and y.
        if (SimController.enableZDir) zDir3 = SwarmingDirection();   //This function wants to rotate all local Zebros in the same direction. Modifies x only.
        if (SimController.enableGoal) zDir4 = AttractiveGoal(); //This function makes Zebros attracted to pylons.

        zFinalDir.x = zDir1.x + zDir2.x;

        if (Mathf.Abs(zFinalDir.x) < 0.4f) zFinalDir.x += zDir3.x;
        if (Mathf.Abs(zFinalDir.x) < 0.6f) zFinalDir.x += zDir4.x;

        if (SimController.enableWallAv && SimController.enableZSwarm) zFinalDir.y = Mathf.Min(zDir1.y, zDir2.y);
        else if (SimController.enableWallAv) zFinalDir.y = zDir1.y;
        else if (SimController.enableZSwarm) zFinalDir.y = zDir2.y;
        else zFinalDir.y = 1;

        if (backToBase != 0) zFinalDir = BackToBase();

        Vector2 zStarveReturnResult;
        if (zebroStarveReturn != 0)
        {
            zStarveReturnResult = ZebroStarveReturn();
            if (zStarveReturnResult != Vector2.zero) zFinalDir = zStarveReturnResult;
        }
    }

    // ==== These functions are part of the core controller. They use the received information from the sensors, and process them into a movement vector.
    Vector2 AvoidWalls()
    {
        Vector2 zebroDirection = new Vector2(0, 1);

        //Consider the y as a value of  0 to 1, with 1 default. y should only care about things ahead.
        //Consider the x as a value of -1 to 1, with 0 default. x should only care about things to the sides.
        foreach (SensData wData in wallData)
        {
            //Check if the current checking sensor is in front of the Zebro (0 to 90, 270 to 359 degrees)
            if (wData.Angle <= 90 || wData.Angle >= 270)
            {
                if (wData.Distance < zoneWallMinimum)
                {
                    zebroDirection.x = (Mathf.Abs(zebroDirection.x) < Mathf.Abs(Mathf.Sin(wData.Angle * Mathf.Deg2Rad))) ? Mathf.Sin(wData.Angle * Mathf.Deg2Rad) : zebroDirection.x;
                    zebroDirection.y = Mathf.Min(zebroDirection.y, 1 - Mathf.Cos(wData.Angle * Mathf.Deg2Rad));
                }

                else if (wData.Distance < zoneWallConsider)
                {
                    zebroDirection.x = (Mathf.Abs(zebroDirection.x) < Mathf.Abs(Mathf.Sin(wData.Angle * Mathf.Deg2Rad) * (1 - ((wData.Distance - zoneWallMinimum) / (zoneWallConsider - zoneWallMinimum)))))
                        ? Mathf.Sin(wData.Angle * Mathf.Deg2Rad) * (1 - ((wData.Distance - zoneWallMinimum) / (zoneWallConsider - zoneWallMinimum))) : zebroDirection.x;
                    zebroDirection.y = Mathf.Min(zebroDirection.y, (1 - Mathf.Cos(wData.Angle * Mathf.Deg2Rad) * (1 - ((wData.Distance - zoneWallMinimum) / (zoneWallConsider - zoneWallMinimum)))));
                }
            }
        }

        //Minimum turning value
        zebroDirection.x = (Mathf.Abs(zebroDirection.x) < 0.3f && zebroDirection.x != 0) ? zebroDirection.x = 0.3f * Mathf.Sign(zebroDirection.x) : zebroDirection.x;
        return zebroDirection;
    }


    Vector2 SwarmingBehaviour()
    {
        //This section describes how the Zebro's should behave towards one another.
        //We define a maximum and minimum range. Beyond the maximum range, Zebro's will rotate towards each other (Cohesion).
        //Within minimum range, Zebro's will try to move away from eachother (Separation).
        //In the "Sweet Spot", Zebro's will essentially ignore eachother until they hit the other zones.


        Vector2 zebroDirection = new Vector2(0, 1);

        foreach (SensData sensData in zebroData)
        {
            if (sensData.Collider.tag == "Zebro" || (sensData.Collider.tag == "ZFoundGoal" && sensData.Distance < zebroMinMin))
            {
                if (sensData.Distance > zebroMaxMax)
                {
                    zebroDirection.x += Mathf.Sin(-sensData.Angle * Mathf.Deg2Rad);

                    zebroDirection.y = Mathf.Min(zebroDirection.y, Mathf.Cos(-sensData.Angle * Mathf.Deg2Rad));
                }
                else if (sensData.Distance > zebroMaxRange)
                {
                    zebroDirection.x += Mathf.Sin(-sensData.Angle * Mathf.Deg2Rad) * ((sensData.Distance - zebroMaxRange) / (zebroMaxMax - zebroMaxRange));

                    zebroDirection.y = Mathf.Min(zebroDirection.y, Mathf.Cos(-sensData.Angle * Mathf.Deg2Rad) * (1 - ((zebroMaxMax - sensData.Distance) / (zebroMaxMax - zebroMaxRange))));
                }
                else if (sensData.Distance < zebroMinMin)
                {
                    zebroDirection.x -= Mathf.Sin(-sensData.Angle * Mathf.Deg2Rad);
                    zebroDirection.y = Mathf.Min(zebroDirection.y, -Mathf.Cos(-sensData.Angle * Mathf.Deg2Rad));
                }
                else if (sensData.Distance < zebroMinRange)
                {
                    zebroDirection.x -= Mathf.Sin(-sensData.Angle * Mathf.Deg2Rad) * (1 - ((sensData.Distance - zebroMinMin) / (zebroMinRange - zebroMinMin)));

                    zebroDirection.y = Mathf.Min(zebroDirection.y, -Mathf.Cos(-sensData.Angle * Mathf.Deg2Rad) * (1 - ((sensData.Distance - zebroMinMin) / (zebroMinRange - zebroMinMin))));
                }

            }
        }

        //if(zebroData.Count() != 0) zebroDirection.x /= (0.5f * zebroData.Count());

        zebroDirection.y = 1 + zebroDirection.y;
        if (zebroDirection.y < 0) zebroDirection.y = 0;
        else if (zebroDirection.y > 1) zebroDirection.y = 1;

        return zebroDirection;
    }


    Vector2 SwarmingDirection()
    {
        //This function lets Zebros move in the same direction when they are in each other's sensor range.
        float zebroHeading = 0;
        int numZebros = 0;

        if (zebroData.Count == 0) return Vector3.zero;
        //Rotate to the average direction of all other detected Zebros.
        foreach (SensData sensData in zebroData)
        {
            if (sensData.Collider.tag == "Zebro")
            {
                if (sensData.Collider.transform.eulerAngles.z > 180) zebroHeading += sensData.Collider.transform.eulerAngles.z - 360f;
                else zebroHeading += sensData.Collider.transform.eulerAngles.z;

                if (transform.eulerAngles.z > 180) zebroHeading -= transform.eulerAngles.z - 360f;
                else zebroHeading -= transform.eulerAngles.z;

                numZebros++;
            }
        }

        while (zebroHeading < -180 || zebroHeading > 180)
        {
            if (zebroHeading > 180) zebroHeading -= 180;
            else if (zebroHeading < -180) zebroHeading += 180;
        }

        if (Math.Abs(zebroHeading) < 10) return Vector3.zero;
        else if (Math.Abs(zebroHeading) > 100) return new
            Vector2((1 * Mathf.Sign(zebroHeading)), 0);
        else return new Vector2((zebroHeading / 100f), 0);
    }
```