NTNU – Trondheim
Norwegian University of
Science and Technology

Implementing Boids behaviors on the ChIRP robots

Hong-Dang Lam

Master of Science in Computer Science
Submission date: June 2015
Supervisor: Keith Downing, IDI

Norwegian University of Science and Technology
Department of Computer and Information Science

Hong-Dang Lam

Implementing Boids behavior on ChIRP robot

Master thesis, Spring 2015
Supervisor: Keith Downing, IDI

Artificial Intelligence Group
Department of Computer and Information Science
Faculty of Information Technology, Mathematics and Electrical Engineering

Abstract

This thesis gives a brief overview to the field of swarm robotics and investigates how robots are able to flock together using the Boids algorithm. Four robots will be used in this thesis. Swarm robotics is an emergent field in artificial intelligence which takes advantage of many smaller cheaper robots instead of using a traditional complex robot. The advantage of flocking robots is that the overall system becomes more stable, more robust because each robot contributes to the overall robustness of the swarm. Boids algorithm is an algorithm meant for computer animation or computer aided design and has been used in various movies.

In this thesis, a Boids simulator will be implemented to get a hands-on experience with the behavior of the Boids. Then the Boids algorithm will be implemented on four differential wheeled robots which have only distance sensors equipped, and a Bluetooth module which they can use to communicate with a Bluetooth-enabled phone or PC. The robots are not able to distinguish obstacles from other robots using their distance sensors only, therefore a centralized computer with an attached camera will be used to provide data to the robots that they do not have access to. The robots used in this thesis is a differential wheeled robot called the ChIRP robot, which is an open source robot made by the CRAB lab at the Artificial Intelligence section of the Department of Computer and Information Science, Norwegian University of Science and Technology.

Preface

First I want to thank my supervisor Professor Keith Downing, at the Department of Computer and Information Science, Norwegian University of Science and Technology, for the guidance through this project. This project would not be possible without him.

I would like to thank Christian Skjetne one of the creators of the ChIRP robot for helping with the technical difficulties that occurred when using the robots on both the Bluetooth communication and the setup of the Arduino libraries. A lot of the library code that the robot's motor and sensors needs to function are written by Christian. The camera tracking software was written by Erik Samuelson, which was an essential part of this project.

I would specially mention the team that the Java gaming library named Slick2D, this gaming library makes it easier to render shapes, and text on the screen. All references are created using citethisforme.com and the software named Mendeley desktop, they have saved me for a lot of work by automating the process of referencing articles.

Hong-Dang Lam
Trondheim, June 11, 2015

Contents

List of Figures

List of Tables

List of Abbreviations

AI Artificial Intelligence

Boids Shorthand for bird-oid objects

cm Centimeter, $\frac{1}{100}$ of a meter

CRAB lab Complex, Robust, Adaptive, Bio-inspired hardware

GPU Graphics processor unit

Hz Hertz, cycles per second

IP Internet Protocol

LED Light-emitting diode

PC personal computer

px Pixels

UDP User Datagram Protocol

Chapter 1

Introduction

This master thesis researches the flocking behavior of the Boids algorithm by implementing it on four differential wheeled robot.

This chapter is the introduction of the thesis, section 1.1 contains background and motivation, section 1.2 contains the research question that this thesis is based on. The research method is described in section 1.3. An overview of the content in this thesis can be found in section 1.4.

1.1 Background and Motivation

Swarm robotics is an emergent field in artificial intelligence. Swarms are inspired by nature, especially animals that work together. There are many reasons for animals to flock together. Fishes flocks together to increase their survivability from being eaten by predators. Ants flock together to find food, and birds flock together to increase survivability and to minimize air resistance.

Usually, flocks do not have a leader, they are able to operate on their own. Each individual in the flock have simple behaviors, but the collective behavior can be quite complex. Swarm robots are decentralized units, meaning that they are not controlled by one centralized unit. Each of them interacts with each other through local interactions, and exchanging information to achieve their goal. Losing one or a few of the robot should not affect the swarm as a whole. In a centralized system, if the centralized unit is lost, damaged or not working anymore, the whole system would not be able to operate anymore.

Robots are becoming more relevant in our lives, already nowadays there are robots helping people mow the lawn, robots that are vacuuming houses etc. Traditional robots are being used for these types of tasks, that is a bigger more expensive robot. Having smaller cheaper robots might be advantageous in some

situations, they are smaller and cheaper. Due to their size, they might be able to reach places that the bigger traditional robots might not be able to reach. Because the swarming robots are usually cheap and interchangeable, they can easily be replaced when they are malfunctioning. The idea is that the small swarm robots will be able to do the same task as the bigger traditional robot. Robot swarm moving in a flock can for example be used to explore new areas. Moving in a swarm makes the exploration more susceptible to failures. Swarm robots can work together to move over rough environments like slippery roads, and they are able to move trough narrow passages by moving through the passage one at a time.

The CRAB lab at the AI department at NTNU has created a robot called the ChIRP robot. These robots are open source and can easily be mass produced. The idea behind these robots was to research swarm robotics. These robots can easily be expanded with other types of sensors, but only distance sensors and light detecting sensors are available at the moment. These robots are still in an early phase of development and has yet to be used to their full potential.

In this project, the Boids algorithm will be implemented on the ChIRP robots to make them flock together.

1.2 Goals and Research Questions

Research question 1: *Can a centralized computer aid a swarm of robots that do not have enough sensors to do the task it was assigned for?*

Swarm robotics is mostly focused on many small cheap robots working together in a swarm to achieve their goal. The swarm robots usually can be swapped out while the system is up and running without affecting the flock. Due to the cheapness of the robots, they might have cheaper equipments and sensors or they might lack the necessary equipment for the robots to be able to do its task. The robots used in this thesis only have distance sensors equipped, but can add more sensors and modules if needed. However there are not other types of sensors available, therefore the robots will communicate with a centralized computer which might be able to aid the robot gain information about its surrounding that it currently lacks. Communication between the robots are limited because the communication between them has to go trough the centralized computer, the robots are not able to communicate directly with each other.

Research question 2: *Will the Boids algorithm make the robots flock together?*

The Boids algorithm uses vectors for each behavior to steer the entities around. The minimal number of behaviors for a typical Boid algorithm are three behaviors. One of them ensures that the entities flock together, the other one makes sure that they move together in the same direction after they have flocked

together. The last behavior ensures that they do not collide with each other, by leaving an open space between the entities. The robots will gain information about their whereabouts and the whereabouts of the other robots from the centralized computer. Using this information the robots will process the information using the Boids algorithm to flock together.

1.3 Research Method

To be able to address the research questions, a simulator was created to run the Boids algorithm in software. The purpose of the simulator was to be able to see how each Boids are supposed to behave, parts of the code from the simulator can be ported to the real robots afterwards. The simulator also serves as a starting point for the real robots, for debugging purpose. The Boids algorithm will be used to make the robots flock and move together. Distance sensors will be used to avoid obstacles. Three scenarios will be used to test if the Boids behaves the way they are supposed to do. The scenarios will be designed to test that the robots are flocking together, and avoiding obstacle by moving around it.

To evaluate the behaviors of the robots, the mean and standard deviation of the robot's angle, velocity and distance between the robot will be graphed.

1.4 Thesis Structure

This thesis is divided into five chapters. Chapter 2 contains research and background information from the field of swarm robotics and the bird flocking algorithm, namely the Boids algorithm. A brief overview of the most common robot control architecture is explained in the same chapter.

In chapter 3 the architecture and the model of the system is explained. The chapter goes in detail on how the system is set up and how the computers are connected. Chapter 4 contains the results, graphs and data gathered from running the experiments and a discussion about them. The last chapter 5 contains the evaluation of the results and a further work section where the details on how to improve the system are explained.

Chapter 2

Background Theory and Motivation

This chapter contains the background research and motivation for this thesis. Section 2.1 contains the background information regarding swarming and more specifically flocking using the Boids algorithm. How the background research have influenced this thesis can be found in section 2.2.

2.1 Background Theory

This section will explain swarming and swarm robots. How to implement bird flocking behavior on particles in a simulation, and the Boids behavior. It will explain how fish schooling and ant swarm works as well to compliment the bird flocking. Swarms can be used to optimize a problem or they can be used to solve complex problems that would cost a lot more using an advanced robot. A brief overview of different common robotics architecture will be presented in subsection 2.1.9

2.1.1 Swarming

Swarm, swarming, swarm intelligence, swarm optimization or swarm robotics are terms used for simple (preferably cheap) agents/robots which can only do simple tasks. However, the power of these robots lies in the numbers [Zhu, 2010; Bonabeau et al., 1999b]. The robots might not be able to do an advanced task alone, but together they might be able to complete advanced tasks. For instance, one robot might not be able to push a heavy box by itself, but with the help of other robots they might be able to push the box. The robots are allowed to

communicate with each other, but they do not necessarily need to. There is no centralized controller that controls the robot, each one needs to find out what it needs to do by itself or by communicating with the other robots. The idea behind these simple and cheap robots are that they can easily be mass produced, and are interchangeable, modularized and disposable. If one robot is malfunctioning or broken, it would not affect the rest of the swarm. It is, therefore, no single point of failure in a swarm, and the system scales well because new robots can easily be added to the system. Swarm robotics is often computational efficient because they each have their own processor. This reduces the computational overhead.

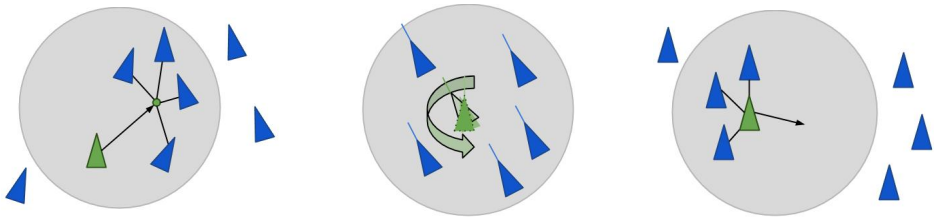
2.1.2 Boids

In [Reynolds, 1999], Craig W. Reynolds created something he called **Boids**, which stands for **B**ird-**oid** object**s**. Boids are particles that would behave like birds, they would try to flock and fly together without colliding. This was done by using three simple behaviors;

Separation Each individual will steer away from the other individual if they are too close to each other. This ensure that they do not collide with their neighbors.

Alignment In a neighborhood (for instance a radius around the individual or the X nearest individuals) find the average angle of the neighborhood and align itself so its angle matches the average angle of the neighborhood.

Cohesion Steer towards the average position of the other individuals in your neighborhood. This makes the Boids stay in the flock.



(a) Cohesion (b) Alignment (c) Separation

Figure 2.1: The three behaviors of the Boids
based on figures from Red3d.com [2015]

The three behavior is per individual Boid, which means that each particle/Boid has to calculate where they are going to fly by checking all the other

Boids position and rotation and then act accordingly. Which makes the complexity of the algorithm $O(n^2)$ for every frame. Reynolds have tried to make the algorithm less computational expensive by putting the Boids into grids with Spatial Hashing. An example of this could be to put all the Boids that has an x-value between 0 and 1 and a y-value between 0 and 1 on the lower left grid, and the ones that has an x-value between 1 and 2 in the next position etc. Using this grid, each Boids in a cell only needed to take the adjacent grids into consideration when checking their neighborhood. That way they do not need to check the position and rotation of all the other Boids. As illustrated in figure 2.2, the

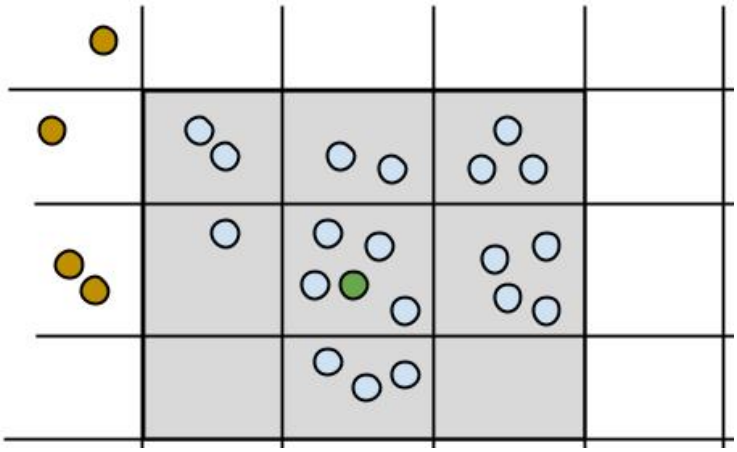


Figure 2.2: Boids in grids using spatial hash

green Boid only needs to check the gray Boids that surrounds its own cell, it does not need to check the rotation and position of the orange/brown Boids outside of the gray highlighted area, because they are too far away to be considered a part of its neighborhood.

In another paper [Joselli et al., 2009] a different technique were used to optimize 3D swarms, a method using neighborhood grids.

Each Boids to cell ratio would be 1 to 1, that is for every cell, there would be at max 1 Boid. Each Boid would have their respective cell based on their position in space, for instance a Boid with low x-value would be on the left of a Boid with higher x-value. Boids who are closer to each other in geometric space would be stored closer to each other in the grid. To obtain this, the Boids needs to be sorted. Odd Even sort and Bitonic sort were used as the sorting algorithm. See appendix 5.3 for more detail.

Of the two sorting algorithm, Odd Even sort was faster, but not very precise. More than 10% of the entities were placed in the wrong cell. An Odd Even sort

algorithm had to be done for each axis, that is one odd even sort for x, y and z. Each axis were sorted in parallel. Bitonic Sort on the other hand was slower, but a lot more precise. Less than 1% of the entities were placed in wrong cell. The reason for placing a Boid in the wrong cell is due to the way the sorting works, it sorts all the entities in one axis first, then a second axis and then the third last axis. For instance sorting on x-values first, for the y-values, then the z-values. When swapping one of the latter axises it might mess up the sorting of one of the other axises.

After the Boids were placed in their corresponding cell, the work could be distributed to the GPU which would calculate where each Boids' new location and rotation would be based on their adjacent neighbor depending on the Moore radius. A Moore radius of 2 would cover the current square, the adjacent squares and their adjacent neighbors as well. The Moore radius is used to determine how many other Boids to consider in its neighborhood. For instance if the Moore radius is 2, the Boids will have a neighborhood as illustrated by the gray color in figure 2.3. Each cell is supposed to contain a Boid.

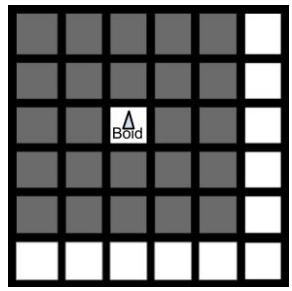


Figure 2.3: A Moore radius of 2 illustrated by the gray squares based on the figure from the paper [Joselli et al., 2009]

In 3D space, extra layers would be added. A Moore radius of 1 would cover all the adjacent grids, which would make the 8 grids that we have in 2D space plus 9 grids above and 9 grids below. That would equal 26 grids, compared to the 8 squares in 2D space. A Moore radius of 2 would cover 74 grids in 3D space (24 + 25 above and 25 below). See figure 2.4 for an illustration of the space covered by a Moore radius of 1 in 3D space, this illustration is only an intersection and does not show all the covered grids.

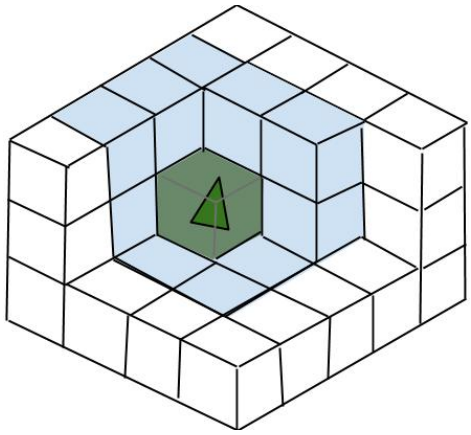


Figure 2.4: An intersection of a Moore radius of 1 in 3D space
based on the figure from the paper [Joselli et al., 2009]

The number of Boids was varied from 1,000 to 1,000,000 and the number of Boid types was varied from 1 to 4. Boids of the same type would try to flock with each other while different types of Boids would try to avoid each other. They were able to see a speedup compared to the spatial hashing method used by Reynolds, but the Boids were not rendered as a bird or an object, only as a primitive shape. They didn't mention if they tried to add non moving obstacles in their test. Due to the high percentage of Boids being placed in the wrong cell when using odd even sort, a lot of Boids would crash into their neighbor during the test run. However using the neighborhood grid method on GPU, real time simulation of 1 million Boids were possible (6-8 fps).

In the paper "steering behaviors for autonomous character" Reynold discusses steering for autonomous character in games and animation, which is a type of autonomous agent that have some ability to improvise their actions. That means that these agents do not have their actions scripted in advance. It is possible for the Boids to have more behaviors, called steering behaviors as explained in [Reynolds, 1999]. The steering behavior decides where the Boids are supposed to steer after their three simple behavior are satisfied. These steering behavior can be seek, flee, pursuit, evade etc.

The seek behavior tells the Boid to seek a goal, it will try to reach the goal/object as fast as possible, but due to its high speed it might have when arriving at the goal, it will fly past the goal. It will then turn around to seek its goal again.

A seek behavior should have an **arrival** behavior to counteract the fly-by,

that means that when nearing the goal, the Boid will slow down so it stops at the goal in the end, instead of flying past it.

Flee behavior is almost the same as the seek, except in the opposite direction. It will try to turn away from the "goal" and fly in the opposite direction and fly as far away from the "goal" as possible.

Pursuit is the same as seek, except it applies to moving objects. Pursuit requires prediction of the target's future position. The approach is to predict the future position of the target, reevaluate and readjust it each step. A prediction might be wrong at one time step, but this only applies for that single time step, and a new prediction will be made at the next time step which will hopefully be correct.

Offset pursuit behavior behaves almost the same way as the normal pursuit behavior, except that it will not "crash" into the target, but will have an offset R. An example of this could be an aircraft flying near the sensors of a base or something similar.

Wander behavior is a type of random steering, where the particle moves randomly around. An easy way to implement this behavior is to apply a random steering force each frame. But this leads to twitchy movements, which does not look very natural. The proposed method is to have a steering direction which is being displaced each frame with a very small random force. That way, if the particle is move forward, it will still keep moving forward the next frame, but it might turn a little bit to the right. Which makes it seem a lot more natural.

Path following behavior enables, as the name suggest, a character to follow a predefined path. However it is not as strict as a train following the rail tracks, as the character is allowed to deviate a little bit from the track. The implementation involves a spine with a radius, which makes the track. The path makes a tube in 3D or a thick line in 2D. The goal for the path following behavior is to first reach the tube, then stay inside this tube, thus following the path. Variations of path following are wall following, and containment. Wall following ensures that the character follows the wall, while containment refers to motion restricted inside a region.

Leader following behavior makes one character follow another character. The follower wants to stay near the leader, but also stay out of the way. If there is more than one leader, they also want to avoid bumping into each other. The implementation of leader following behavior relies on the arrival behavior where the goal is a point behind the leader. The follower will slow down when drawing near the point behind the leader and eventually stop

before it bumps into the leader. If the entity is in front of the leader, it will fly away and around the leader so it is not in the way.

2.1.3 Collision Avoidance

In the paper [Craig W. Reynolds, 1988], W. Reynolds discusses how to perform obstacle avoidance. That is, obstacles that are placed in the environment which is not a Boid. These obstacles are usually static, non moving obstacles. He starts out with the idea of a force field around the obstacle which he calls the *steer away from surface* approach. The idea is to have every obstacle emit a forcefield around itself which pushed the Boids away. For instance if a Boid is flying toward an obstacle, the obstacle would push the Boid to one side of itself. However this force field method would not work if the Boid flew straight into an obstacle, because the forcefield force would be straight opposite of the direction the Boid is flying thus making the Boid decelerate until it stopped.

The next obstacle avoidance technique Reynold discussed is the curb feeler technique or steer along the wall technique. The idea is to have a feeler that would detect an obstacle before the Boid would crash into it, then turn the Boid away from the obstacle. This can be compared to walking down a dark alleyway where you reach your hand out to feel the walls around you, and navigate through the alleyway just by feeling the wall(s).

The last technique for navigating and avoiding obstacles discussed was image processing. Images could be processed in real time to a gray scale image, where white would signify an obstacle. The algorithm would start with the center of the image, if this was a white pixel it would start to search outwards in a spiral to find either a gray pixel or a black one and then turn the Boid in this direction. This could also be combined with a *Z-buffer image* which gives us a map of the distances to obstacles that lies in front of the Boid, this z-buffer image can be obtained by radar, sonar or similar technology. One interesting way of using the Z-buffer image is to implement a "steer towards the longest clear path". However using this technique without any form of planning or learning might lead the Boid into a local cavity, which might be a dead end.

2.1.4 Physic based control system

In the paper [Spears et al., 2004], a self emergent system was formed using simple attractive and repulsion force for each particle. The idea behind their system was to create an artificial physics framework (AP) that would simulate a physical system. In their paper they had the particles attract other particles that were farther away than distance r and a repulsive force is applied if the particles are closer than distance r. This leads to the particles always being at distance r from each other, which will form a hexagonal lattice. In the hexagonal lattice, each

particle will be at a distance r away from each other, but the next neighbor will be $\sqrt{3}r$ away. Therefore each particle only have a vision range of $1.5r$ so they do not affect the particles that are too far away from it.

In the simulation they spawned all the particles in a cluster, using the two-dimensional Gaussian random variable to initialize the position of the particles. The particles starts with a velocity of 0.0, but their framework does not require so. Due to local forces, the particles will disperse and form local hexagons. To evaluate the quality of the lattice they measured the orientation error of the lattice: they took any particles that were $2r$ apart and formed a line segment, then they took any other particles that were $2r$ apart and formed another line segment. The angle between these to line segments should be measured to be a multiple of 60°. The error would be the absolute value of the difference between the measured angle and the nearest multiple of 60°. The error ranges from 0° to 30°.

They also checked the size of the particle cluster. For each particle i they counted the number of "close" particles, that is particles that are in the range of $0 < r < 0.2r$. The minimum cluster size is 1.0 because they count the particle i as well as its neighbors. The cluster count was averaged for all the particles. At the start there was a high cluster count, but it decreased to roughly 2.5 after 6 time steps.

They also tried out making square patterns, but had to introduce a concept of spin; each particle was either spin "down" or spin "up". Opposite spins would attract each other if the distance was greater than r, and repel each other if the distance is less than r. If the particles had opposite spin the distance would be $\sqrt{2}r$. This means that all the particles on the vertical space would be alternating between spin up and spin down, the same goes for the particles in the horizontal space. The particles on the diagonal will have the same spins as their diagonal neighbors. Sometimes the formation would have "holes" in it, so instead of static spins, the particles were allowed to change their spin. This would fill in the holes or flaws in the square lattice, according to their theory.

A particle would only change its spin if it had a very close neighbor ($r < 1$), and the probability of changing spin was quite small.

2.1.5 V like bird formation

In the paper [Nathan and Barbosa, 2008], bird flocking is discussed, they wanted to simulate the V shape that can be observed when birds fly together. They ran a simulation that where each bird individual had three simple rules:

- **Coalescing rule**:
  The birds should try to seek the proximity of the nearest bird.

- **Gap-seeking rule**:
  If rule 1, the coalescing rule is not applicable anymore the bird should find a position with unobstructed view - that is, the bird should be able to see in front of it without anything being in the way.

- **Stationing rule**:
  Try to stay in place.

These rules would make sure that the birds were able to flock and form different shapes. The only thing that was common between the different runs in this paper was that the bird behind would be a little bit behind and slightly left or right of the bird in front (following rule #2). A lot of different shapes was obtained during the runs, the birds flocked and formed a V-shape, a diagonal line, an inverted V-shape etc.

2.1.6 Family bird: A heterogeneous simulated flock

The paper [Demsar and Bajec, 2013] by Demsar and Bajec says that bird flocks and fish schools seems to be very complex, but the mechanics are very simple as illustrated by Reynold. Only a few simple rules will create flocks that flock together and splits up to avoid obstacles. These flocks are not spectacular or mind blowing compared to the flocks found in nature, due to the rigid motion of each individual. To tackle the artificialness of each individual, Heppner introduced randomness to the motion of the individuals. He defends it by saying that these randomness simulates wind gust, random obstacles and other factors. However, the authors of this paper does not agree with this approach because wind gusts will affect the whole flock, not just a random single individual at random. Even with these randomness added, the flock still does not seem lifelike enough compared to their counterpart found in nature. The reason for the lack of breathtaking in these flocking algorithms are that the individuals all have the same characteristics. In nature each individual will have different size, age, form and shape.

Usually in flocking algorithms, each individual takes into account where all the other entities are and then act accordingly. In nature, each individual might have limited information about the flock. It might only be able to gain so much from its vision due to other flock members being occluded or not able to hear some of the other individuals due to noise or other factors.

This paper runs a simulation with different types of birds, where social relations are a factor and individuals might be solitary or social. Social individuals are entities who would like to stay close to members of its own social group, for instance a social dove will want to stay with other doves. Solitary individuals on the other hand does not care about staying with its own flock and might drift

to another flock. For instance a solitary dove might fly amongst hawks or other types of birds. A flock in this paper are defined as a group where the individuals affect each other in the same group. The simulation they ran varied between all solitary birds to 4 types of different social birds.

2.1.7 Particle Swarm Optimization

Swarms has a lot of potential, not necessary only on robotics but swarms can be used to optimize problem, hence the name **particle swarm optimization (PSO)** [Eberhart and Kennedy, 1995]. PSOs is used to solve problems where optimization are needed. The idea behind PSOs are to have a swarm of particles spawn at random positions in the search space, and then let them fly around searching for solutions. Each particle will know the best solution it have found so far, and the best solution that has been found globally. In some PSO implementations a local best found solution is also known amongst the individual. This local best solution is the best solution amongst a subgroup of individuals and can change for a particle depending on the position of the particle.

If all of the particles were to be attracted to only the global best found solution so far, they might risk to be stuck in a local maxima without being able to find the other local maximas. To avoid being stuck in a local maxima, each particle is also attracted to the best solution it has found, and maybe to the neighborhood's best found solution as well. This ensure that the particles "jiggles" (introduces enough noise) so that the particle might jump to another local maxima.

2.1.8 Other swarming animals

This section will look at other flocking animals or insects and how they have been modeled in simulation.

Fish schools

In the paper named "Artificial Fishes: Autonomous Locomotion, Perception, Behavior, and Learning in a Simulated Physical World" [Demetri et al., 1994] we are given the explanation on how fishes form schools and how different intentions make the fish behave the way they do. He starts out with explaining how a fish and how the simulator is constructed, the math behind it and how the motor controllers work. For the simulation there is different types of fishes, some of them are predators and some of them are preys. Where predators are larger fishes which tries to eat other smaller preys. Each fish has a 300 degrees, where it can see in front of it and has a blind spot directly behind it. These fish uses the containment behavior mentioned earlier, where they swim freely around in the aquarium, but are not allowed to/not able to leave it.

The range of the vision is also limited and might be occluded by other objects. Each fish has a intention generator, which basically is a flowchart of what the fish needs to do. The prey and predator fish have different intention generators. The predators do not get preyed upon, and thus does not need to look out for other predators, therefore are the intentions of escaping, mating and schooling with other fishes of the same species are disabled. The reason for disabling mating is because there is no need for new predator fishes because they do not die in this simulation. They also do not need to school with other predators because they are not in danger, and do not need the extra survivability.

Whenever a predator sees a prey it will chase the prey if the cost of reaching it is low enough. If it is too high, it will not bother chasing.

Preys on the hand needs the extra survivability, and will try to school with the other fishes if it detects a predator nearby. Each fish will then try to stay a certain distance from the others, which is roughly one body length in distance. Then the fishes will try to adjust its speed and direction so it matches the other members. When this school of fish encounter an obstacle, each individual fish will try to avoid this obstacle. This might lead to the school splitting up and rejoining after they have avoided the obstacle.

A third type of fish introduced here are the pacifists fish. This one differs from the other two type in that the intention of mating is activated while escaping and schooling are deactivated. The paper describes that there are male and female fishes, and the two behavior which can occur when the fishes start to mate. A behavior named *nuzzling* where the male fish seeks the female and nudges her abdomen until she's ready to spawn, and *spawning ascent* where the female swims repeatedly to the surface while releases gametes. The paper also describes in detail how the fishes select potential partners and how they try to impress each other for mating purposes.

Ant swarms/colonies

Ant swarms behaves differently than other types of swarms [Blum, 2005], ants do not try to form formations for survival in the same way that birds and fishes do. Ant swarming is mostly about their foraging behavior. That is how they find food for their colony. Each ant's goal is the survival of the colony rather than the survival of each individual. When ants try to find food, they scatter the area by walking in random manner. While exploring the ants leave behind a chemical on the ground. A so called pheromone that the other ants will be able to feel/smell. This pheromone will slowly but surely dissipate. Whenever the explorer ant find a food source, it will evaluate the quality of the food before returning to the anthill. During the return trip, pheromones are reapplied to the path, but the amount is adjusted based on the evaluation of the food. Better food will yield more pheromone on the path. This method will ensure that the

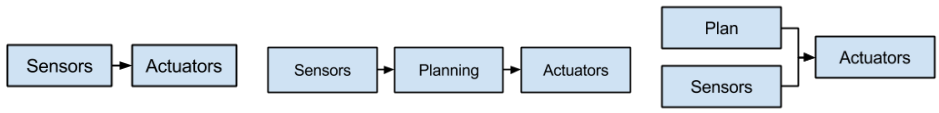
rest of the ants will take the shortest path from the anthill to the food. For the artificial ants, the ant system uses a graph $G = (V, E)$ to model the paths, V are the nodes and E are the edges between the nodes. In the paper [Blum, 2005], they use two nodes: v_s which is the starting node or anthill. The node v_d is the food source. There is two ways to reach the food source from the anthill, e_1 and e_2, which have lengths l_1 and l_2 where $l_2 > l_1$. A value τ_i denoted the artificial pheromone, and it indicates the strength of the pheromone.

An ant will choose a path with the probability $p_i = \frac{\tau_i}{\sum_{n=1}^{k} \tau_n}$ where k denotes the number of paths. In the paper, they only have two paths, so the probability of choosing a path is $p_i = \frac{\tau_i}{\tau_1+\tau_2}, i = 1, 2$. The ant will probably choose e_1 if $\tau_1 > \tau_2$ and vice versa. The ants will return using the same path as the one it took, and reinforce the path with new pheromones using the formula $\tau_i \leftarrow \tau_i + \frac{Q}{l_i}$ where Q is a positive constant. The pheromones that have already been laid out in the path will slowly evaporate, the evaporation formula used is $\tau_i \leftarrow (1 - \rho) \cdot \tau_i, i = 1, 2$, where $\rho \in (0, 1]$. These math formulas will over time make sure that the ants are converging to the short path.

The biggest difference between these artificial ants and real ones are that these move synchronously, while real ants are asynchronous. Real ants leaves pheromones on the ground whenever they move, these artificial ones only leave behind the pheromones on the way back to the anthill. The normal ants' pheromone strength are due to evaporation, while the artificial ones regulates the strength of the pheromones using an evaluation of some quality measure. The ant system can be used for approximate algorithms for combinatorial optimization problem, especially for combinatorial problems which are NP-hard.

2.1.9 Robot architectures

Software architecture is the methodology for structuring the code or algorithm. One software architecture model might work better for one purpose while another architecture works better for a different purpose.



(a) reactive control architecture (b) deliberative control architecture (c) hybrid control architecture

Figure 2.5: Robot control architectures

This subsection will briefly explain common architectures found in robotics.

In robotics, the architectures need to decide how to combine reactive control and model-based deliberative planning. Reactive robots rely mostly on their sensor inputs and reacts accordingly. For example, a robot equipped with distance sensors that usually drive forward might turn if the front distance sensor detects an obstacle.

Deliberative planning on the other hand also uses the available sensors on the robot, but they do not react immediately like the reactive robots would do. Deliberative planning robots uses the information gathered from the sensor(s) to make a plan of where it should head or what it should do, before executing its action(s).

A mix of both planning and reactive robot architecture which uses reactive techniques at the lower level and deliberative planning at the higher levels are called hybrid architectures.

Brooks subsumption

Brooks subsumption architecture is the most common reactive robot architecture. As explained earlier, a reactive architecture is based on a direct sensor to actuator mapping. That is, whenever a sensor measures anything over a predefined threshold or if the sensor is detecting anything, the actuator will react immediately. The idea behind the subsumption architecture is to split the behavior of the robot into sub-behaviors into a vertical hierarchy as illustrated in figure 2.6.

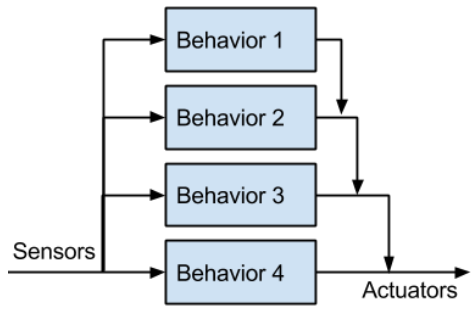


Figure 2.6: Subsumption architecture
basen on a figure in [Yongjie et al., 2006]

As seen in the figure, the sensors input the information it has detected, and each behavior reacts accordingly. A higher level of behavior will have higher priority than the lower ones. For example, behavior 4 in figure 2.6 would have priority over behavior 3, and behavior 3 would have priority over behavior 2 etc.

The advantage of the Brooks subsumption architecture is that the robot using it can have different goal depending on the specific situation. For example, if you have a robot that needs to find a specific item in an area. This robot would first wander around aimlessly, and its sensors would be used to avoid crashing into obstacles or other robots. When the robot has located the item it wants, it would use its sensors to move towards that item instead of using it to avoid it, its sensors could be used to align itself with the item to push it towards a different location.

2.1.10 Deliberative control architecture

Deliberative control architecture are based on Sense-Plan-Act principle as seen in figure 2.5b, the robots uses its sensors to get a full overview of the environment. After it has gained full knowledge of the environment, it will plan solutions then consider them before choosing an action. Deliberative planning robots are usually very dependent on precise sensors to be able to map the environment. It is also assumed that the model of the world the robot will be in, is provided.

Hybrid control architecture

Hybrid architecture is a mix of deliberative control architecture and reactive architecture. The idea behind the hybrid control architecture is to work around the limitation and drawbacks of both the reactive and the deliberative control architecture.

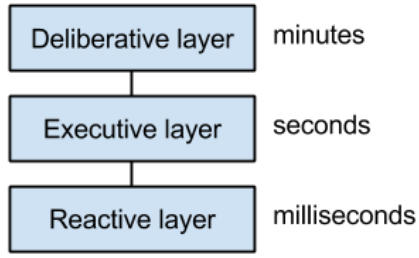


Figure 2.7: Three layer control architecture
based on a figure by Jakimovski [2011]

The hybrid control architecture usually has a deliberative layer to plan and model the environment, but the deliberative controls might be slow and not able to act fast enough in certain situations. Each decision might take minutes. The

environment that the deliberative layer creates or maps out can be learned from data or gathered through the sensors from the reactive layer.

The slowness of the deliberate layer is the reason the reactive layer is added to the control architecture as well. The reactive layer usually has faster reaction time than the deliberate layer, and can help the robot navigate when they need to do an action fast. The reactive layer's decision cycle is often on the order of milliseconds.

To glue together the reactive layer and the deliberative layer, a layer is put in between the lower reactive layer and higher deliberative layer. Namely the executive layer or sequencing layer. The executive layer accepts directives from the deliberative layer and puts them in order for the reactive layer. The executive layer is slower than the reactive layer, it takes seconds to make a decision.

The hybrid architecture explained here is the most common architecture for the hybrid control architecture, which is called the three-layer architecture due to the three layers deliberative, executive and the reactive layer as seen in figure 2.7.

2.1.11 Related systems and projects

This subsection will look at systems and projects where the flocking algorithm have been implemented on a physical robot.

Flocking on wheeled robots

In 1992, Matarić designed a series of behavior based modules for mobile robots, that would make them flock if combined and properly weighted. The wheeled robot were equipped with collision sensors and six distance sensors. By using these sensors combined, the robots would be able to measure the distance to the other robots within a small neighborhood. Matarić implementation uses four behaviors;

collision avoidance makes the robot steer away from objects that are closer than a predefined

following makes the robots follow the other robots, this behavior is implemented by using the two sensors on its side. If there is only one perceived object on either side of the robot, it will turn towards that side. If there are other robots on both its side, it will keep moving forward along with the other ones until there is a robot in front of it blocking its path.

Dispersion and aggregation Steers the robot towards the computed center of mass, this center of mass is computed using the reading from the sensors.

This system uses five robots with the subsumption architecture. If these robots were to flock through a cluttered environment, a more sophisticated sensing ability would be required. The robots had radio transceivers that they could use to distinguish between robots and other objects.

There were other behaviors implemented on these robots as well, but the four behaviors mentioned above are the ones that make the robot flock.

Flocking on quadcopters

In the paper [Csaba et al., 2014] a flocking algorithm were implemented on quadcopters. Ten of these quadcopters were flying around in the air autonomously, they would communicate with each other and all of these quadcopters had access to a GPS. The swarm could form shapes, for example forming circles or lines when instructed to do so. When forming a circle, each quadcopter had to take its place on the circle by evenly spacing themselves on the perimeter of the circle. The quadcopters were also able to do leader following behavior. The implementation uses GPS to determine the position of the quadcopters, but their algorithm works with all other sensory input where relative position, velocity and altitude information can be obtained. Their swarm flock were dependent on sensory errors and delays in the system. The researchers were able to fly this swarm flock autonomously of quadcopters for 20 minutes.

The algorithm were implemented in two and a half dimension, that is the quadcopters would fly up to the correct altitude, and the flocking and formation part of the algorithm would only work in two-dimension after the copters had reached its correct altitude. Each of the quadcopters had to have a 6-10 meters between them due to inaccuracy from the GPS and other disturbances found in the air. If they were to flock at closer than the 6-10 meters range, they would risk crashing into each other.

2.2 Motivation

Most of the flocking systems mentioned in this chapter makes the entities flock together by using some sort of attracting force, and when they are too close to each other, a repellent force acts upon the entities to keep them at a fixed distance from each other.

Swarming is an emergent field in artificial intelligence, nowadays it is becoming more popular with multiple smaller robot than one expensive traditional robot, especially when it comes to AI research. There are some few other projects where flocking behavior has been implemented as mentioned in section 2.1.11.

There are various applications for flocking robots, some of them are mentioned in [Csaba et al., 2015]. Flocking robots can be used for surveillance if they are

equipped with cameras, search and rescue operations. A swarm of flocking robots could fly over a farm to check the health of the soil or the vegetables that are grown there.

This project will make it easier to me to get a better understanding of how swarming and flocking works. It will also give me a hands-on experience with implementing AI algorithms on real life robots.

In this thesis, the Boids algorithm will be used to make the ChIRP robots move together in a flock. A centralized computer will help aid them, by acting as a communication bridge between the robots, and by using a camera it will also act as a GPS by sending coordinate information to the robots. By using distance sensors, the robots might be able to move close to the other robots without crashing, and the centralized computer might help make the robots gain information about its surrounding.

Chapter 3

Architecture/Model

This chapter contains the architecture and model of the system, which tools and methods that has been used, how they all connect together to run the experiments and produce the result. An overview of the system is presented in chapter 3.1. A description of the robot is presented in section 3.2. How the robots calculate where to move is explained in section 3.3. The brief explanation of the simulator is found in section 3.4, and the biggest difference between the physical experiment and the simulator is shown and compared in section 3.5. The math used to plot the graph are shown in section 3.6.

Throughout this chapter the word "Boids" will solely refer to the Boids on the simulator, while entity or entities refer to both the physical robots and the Boids.

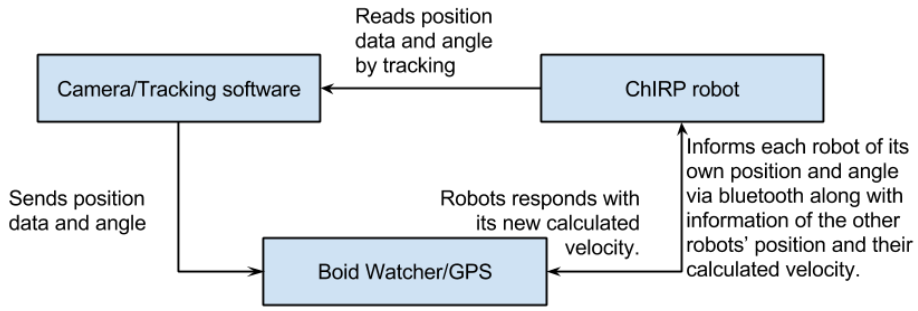


Figure 3.1: Overview of the components of the system

3.1 System overview

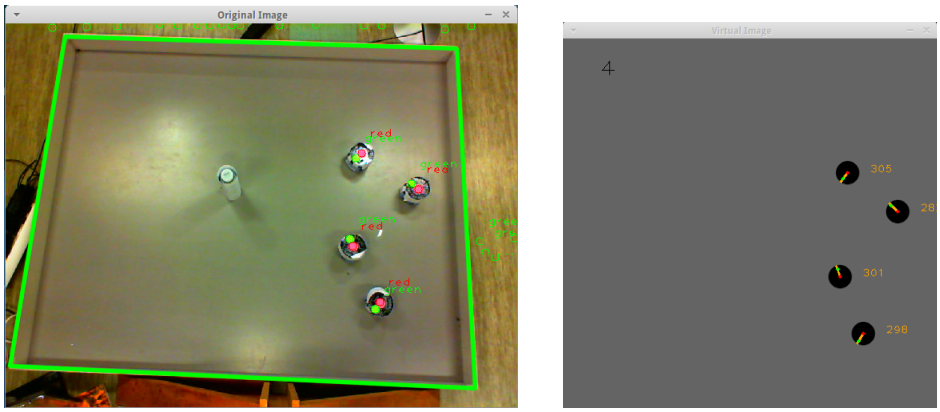
The system used in this experiment consists of three primary components as illustrated in figure 3.1: the camera which is used by the tracking software, the "Boid-watcher" which is a centralized computer that acts as a GPS and the ChIRP robots. The camera tracking software tracks each robot's position and its angle, which is sent to the Boid-watcher. The Boid-watcher then forward this information to the robots via the Bluetooth serial port.

The Boid watcher does not only work as a GPS, but it is also a communication bridge between the robots, it provides information about the position and velocity of each robot to all the other robots. The robots are not able to connect to the other robots directly, that is why the they need to pass information to the other robots through the watcher software via Bluetooth.

Even if it is possible for one computer to run both the camera tracking and the watcher software, two separate computers were used in this setup. The camera tracking software being run on a stationary desktop computer, and the camera is attached to a pole above a sandbox where the robots roam around. The camera is connected to the stationary desktop.

In this experiment, the watcher software runs on a laptop with a built-in Bluetooth adapter. The watcher software needs to communicate with all the robots simultaneously and that requires processing powers. The desktop computer was not able to run both the camera tracking software and the watcher software at the same time while sending Bluetooth data to all four of the robots. When it tried to run both, it was not able to read the data from the camera fast enough and thus the tracking software would crash. The Bluetooth connection between the computer and the robots might not always be stable, if the connection were to be unstable at times, a reboot of the robot or/and a reconnection of the Bluetooth connection would usually fix the problem.

The camera tracks the robots using image recognition, it recognizes the robots by the two circular post-it notes that are placed on top of each robot. To filter out all the noise from the surrounding area and remove the colors that are irrelevant to track, the camera tracking software needs to have a threshold for what it considers to be red and what it considers to be green. This is specified in a configuration file that the camera tracking software loaded whenever it was launched. An example of the configuration file can be seen in the appendix on section 5.3. As seen in the example, each color is defined by six boundaries, a minimum value, and a maximum value for the hue-saturation-value. If a color seen by the camera is inside this boundary it will be considered as the color it is looking for.



(a) Original image (b) Virtual image

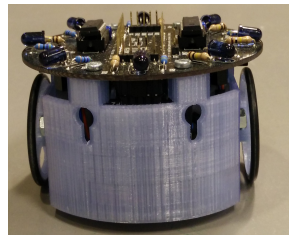
Figure 3.2: Images from the camera tracking software

The camera might detect red or green colors that do not belong to the robots, for example when the sun shines into the room, the green part of the floor around the sandbox might be detected as "green" by the camera tracking software. This is not a problem for the camera tracking software because this green floor is outside of the predefined box and the tracker only tracks colors that are inside of this predefined box, which the user has to specify when starting up the software. This predefined tracking are is the green outline in figure 3.2a. Any robots found outside of this green boundary box are not tracked, only the robots inside this predefined area will be tracked and have a legal position value and an angle. That is why we need a sandbox to contain the robots so they do not wander off outside the range of the green boundary box where they are not tracked anymore. Only the position and the angle of the tracked robots will be sent to the Boid watcher software over UDP.

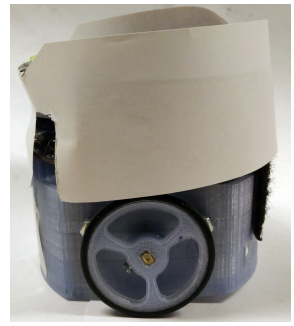
Detecting red or green color inside the sandbox that does not belong to the robots imposes a bigger problem than detecting these colors outside the box. This might happen if there is a green shaded shadow or a reflection of an object reflects onto the sandboxes' surface. If the software detects green or red color that do not belong to the robot, it will simply ignore these colors if the green and the red color are far apart from each other, because the camera tracking software does not do anything unless both these colors are paired. If these "noise" colors do disrupt the movement or angle of the robots, then a recalibration needs to be done to filter out the colors and make the tracking more precise.

3.2 Robots

(a) The robot used in the experiment



(b) Standard ChIRP robot



(c) ChIRP seen from the side

Figure 3.3: ChIRP robot seen from various angles

The robot swarm consists of four ChIRP robots. The ChIRP robot is a circular shaped robot with differential wheel. A differential wheeled robot is a robot with two separate driven wheels on each side, which it can use to move itself. If it wants to change its direction it can vary the relative speed of each wheel or motor. For instance if the right wheel moves faster than the left one, the robot will turn to its left. The advantage of differential wheel is that an additional steering motor is not required for the robot to move around. Usually a caster or additional wheels are added to balance a differential wheeled robot, but the ChIRP robot does not have anything of the sort. Whenever it is moving, the back or the front of the robot is scraping against the floor depending on if it has a backward or forward momentum. Scraping against the floor does not affect the movement of the ChIRP robot, it was designed this way. The robots have a max speed of approximately 13 cm/s, in this experiment the velocity of the robot will not exceed 7.8 cm/s.

Each robot is equipped with eight infrared LED lights and receivers used for measuring distance. Infrared light are emitted from the LEDs, reflected on a surface and received in the infrared receiver. For the robot to know how far from an obstacle it is, it measures the amount of infrared it receives. The higher the amount, the closer to the object it is. This method of measuring distances works very well for bright or colored surfaces, dark surfaces on the other hand do cause problems because the infrared light is not reflected so well.

The distance sensors are spaced evenly around the robot, where one of the sensors are pointing directly in front of the robot. This sensor can be used to

detect whether there is an obstacle directly in front of it. For this experiment, only the three sensors in front are used. Because the robots only moves forward, so it only needs to determine whether there is an obstacle directly in front of it. The two other distance sensors are also needed, because using only the one in front is not sufficient to determine if the robot is going to crash into an object. The other sensors on the side or the back of the robot is not used because the robot does not move in that direction. So it does not matter if there is an obstacle behind or on the side of the robot.

Each of the ChIRP robots is equipped with a Bluetooth module, which it uses to communicate with the watcher computer wirelessly. The robots are very hollow on top as seen in figure 3.3b and the distance sensors have difficulties sensing other robots due to this hollowness. To counteract the hollowness, a white paper strip was taped around the robots. This makes it easier for the robot to detect the other robots nearby because the paper will reflect the infrared light that the robot uses to determine the distance. To be sure that the robot would be able to detect the obstacles placed in the sandbox, the obstacle would have a white paper wrapped around it to reflect the infrared light. The obstacle used for this experiment consist of a bottle filled with water. The water inside the bottle is used to make it heavier, so it does not fall over if a robot were to crash into it. And the paper around the bottle is there to reflect the infrared light that the robot uses for measuring distances.

The camera tracking software uses image recognition to track the robots. That is why each robot needs to have a red and a green post-it note on top of it. The red one determines where in the sandbox the robot is located, and the green one is used to decide which way it is pointing and to determine the angle of the robots.

3.3 Robot controller

This section will go through the controller of the robot, how they work and what steps the robot takes to execute its action.

The robots are implemented using the idea of a hybrid robot control architecture as explained in section 2.1.10. The distances sensors are the reactive layers which will be used to guide the robots away from crashing into obstacles, walls or other robots. The deliberative layer will be the code that processes the data sent from the watcher software on the computer, it calculates where the robot should be heading. The deliberative layer are usually slow, but the robots in this experiment are aided by the centralized computer, it does not need to use a lot of its processing power to map the environment. The robot already knows that they are roaming inside a sandbox, and they already knows the size of this sandbox. The executive layer would be the code running on the robot that decides what

the robot is doing, whether that is reading the sensors and avoiding obstacles or moving towards the planned destination for flocking.

When the robot is turned on and a Bluetooth device is paired with its Bluetooth module, the robot will stand still and wait. The robot is waiting for a command or data from the watcher software. A human can send commands to manually control the robot if needed, this can for example be used for leader following as explained in section 2.1.2.

If the watcher sends data to the robot, it will start to calculate where it should move based on the data it received. The robot takes into consideration the position and the velocity of all the other robots. The positions of the other robots are used to determine the sum of the cohesion vector and the separation vector. The velocity of the other robots are used to determine which way they are pointing, and this is used to find the alignment vector.

A fourth behavior was added for this experiment, which is called the "away from wall" behavior. As the name implies, this is a vector that will lead the robot away from the wall of the sandbox if the robot is too near the wall. After calculating each of these vectors, they are multiplied with a weight depending on how much that specific behavior should impact the movement of the robot. For example, the separation vector should have a higher impact on the movement of the robot because it needs to avoid the other robots when it is too close to one.

Cohesion vector, on the other hand, does not need to affect the robot as much, it is mostly there to guide the robot into the flock. Therefore, the separation vector is multiplied with a higher number than the other ones.

The equation for how the final acceleration vector which decides the direction the robot is going to move is expressed as:

$$\vec{A} = \Sigma_{i=1}^{N_b}(W_i \vec{B_i}) \tag{3.1}$$

where:
$\vec{A}$ = the acceleration vector
B_i = the behavior i, for example B_1 could be cohesion behavior, B_2 alignment behavior etc.
W_i = the weight for the behavior B_i
N_b = the number of behaviors

The neighborhood distances used on the physical robots are the following ones:

cohesion distance = 800 px

alignment distance = 200 px

separation distance = 150 px

away from wall distance = same as alignment

obstacle distance = same as separation

Each of these behaviors, which is denoted by B_i in equation 3.1, and the following weights W_i have been used:

cohesion vector is multiplied by 2

alignment vector is multiplied by 2

away from wall vector is multiplied by 1

avoid obstacles does not exist

separation vector is multiplied by 4

After calculating the acceleration vector, it will add it to its velocity vector, then use the arctangent function to calculate which direction the robot will turn to. The robot calculates the new direction it needs to face by using the following equations:

$$\vec{V}_{new} = \vec{V}_{old} + \vec{A} \tag{3.2}$$

$$R_{goal} = atan2(\vec{V}_y, \vec{V}_x) \tag{3.3}$$

$$D_{turn} = (R_{goal} - R_{current}) * \frac{180}{\pi} \tag{3.4}$$

$$\vec{A} = [0, 0]^T \tag{3.5}$$

where:
$\vec{V}_{old}$ = previous velocity of the robot
$\vec{V}_{new}$ = new velocity of the robot, the length of $\vec{V}_{new}$ is capped between -40 and 40 or using mathematic notation: $|\vec{V}_{new}|\epsilon[-40, 40]$.
R_{goal} = the angle the robot will be facing after it has turned around, measured in radians, $R_{goal}\epsilon[-\pi, \pi]$
$R_{current}$ = the angle the robot is currently facing, measured in radians, $R_{current}\epsilon[-\pi, \pi]$
D_{turn} = the angle the robot needs to turn to face the correct direction, measured in degrees. $D_{turn}\epsilon[-180, 180]$

The robot needs to know how much it is going to move in the x-direction and how much it will move in the y-direction as shown in equation 3.2. Then by calculating the arctangent of V_x and V_y, it finds out which angle it needs to face to be able to move in the direction of $\vec{V}_{new}$. When the robot has calculated which direction it wants to go, it calculates D_{turn} by using the formula in equation 3.4. D_{turn} is the amount of degrees the robot needs to turn to face the right direction.

The velocity $\vec{V}_{new}$ is only used to determine which direction the robot should be facing, it is not used for determine the velocity of the robot. The velocity of a robot is calculated by calculating how much the robot has been moving since the last time step, or by finding out how much the position of the robot has changed since the last time step:

$$V_{robot} = \Delta P = P_{new} - P_{old} \tag{3.6}$$

When the robot has received the data from the watcher, it will first check measure the distance in front of it, to check if there is an obstacle in front of it. If there is on, it will turn away from the obstacle, and wait for new data from the watcher. If the path is clear, it will do the calculation shown in equation 3.1 trough 3.5. Then the robot will turn D_{turn} degrees so it will face the correct direction.

The robot will then measure the distances once again, in case it has turned towards an obstacle. If there is no obstacle in front of it, it will move forward while waiting for new data from the watcher software. If the robot did find an obstacle in front of it, it will turn 90°to either the right or the left randomly. The robot will stop after turning and wait for new data from the watcher software.

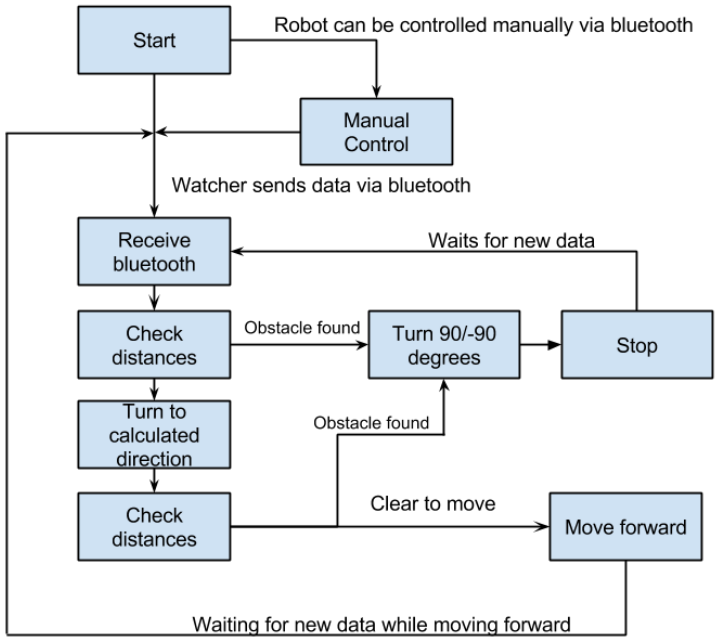


Figure 3.4: Flowchart of the robot's behavior

3.4 Simulator

A simulator was created where the Boids was implemented solely in software, and rendered on screen, that is no physical robots were used in the simulator. The reason to use a simulator was to see how the Boids were supposed to behave and have a working example to compare with. A screenshot of the simulator is shown in figure 3.5.

A typical Boids simulator usually have a wraparound space. If one of the Boids goes outside the window, it will "teleport" to the other side of the window. For example if one of the Boids flies too far to the right and hits the right border of the window, it will loop around and end up on the left side of the window. This is how the typical Boids algorithm usually works on a simulator. But physical robots can not loop around the stage like the Boids on the simulator can. That is why the simulator used in this project stops the Boids from moving beyond the walls of the window.

The same "away from wall" behavior is also implemented in the simulator, so the Boids do not move into the wall aimlessly.

For each frame, the Boids will update it velocity by calculating a vector for each behavior, and then these vectors are added to the acceleration vector by using the formula found in equation 3.1. The acceleration vector is then added to the velocity vector, and the velocity is capped off if the length of the vector exceeds the maximum allowed speed. If there is no velocity cap, the Boids' velocity would increase towards infinity. The velocity then decides where the Boids are going to move. The procedure to calculate the velocity vector of the simulated Boids are the same as the one the robots are using; in equation 3.2.

After calculating the $\vec{V}_{new}$, it needs to cap the max velocity. If the velocity is not capped, the speed of Boids would increase and they would move outside the window. As for the robots, the Boids velocity vector is kept between -40 and 40: $|\vec{V}_{new}| \epsilon [-40, 40]$. The Boids do not need to calculate which way it needs to turn to move in the direction, because it is able to move in all 360° directions without the need to turn. The robot had to turn because it could not move sideways, it could only move either forward or backward.

The Boids simply moves by adding the velocity vector to their position:

$$P_{new} = P_{old} + \vec{V} \tag{3.7}$$

where:
P_{old} = old position of the Boid
P_{new} = new position of the Boid

Before the next time step, the acceleration has to be reset to a null vector for it to work as shown in equation 3.5.

The Boids in the simulator do not have any form of rotation, the angle seen on screen are calculated by taking the arctangent of the velocity vector. That is why the Boids in the simulator do not need to rotate; they change their direction instantly by changing their velocity.

In the simulator, there is no need for any type of sensors. Each Boid has access to the location of all the obstacles and all the other entities.

In the simulator, the following parameters have been used:

cohesion distance = 250 px

alignment distance = 175 px

separation distance = 120 px

away from wall distance = same as alignment

obstacle distance = same as separation

These distances are used by the Boids to determine how near it has to be before it should calculate the vector. For example if the Boid is in the middle of the screen, that means that it is not near a wall, then the "away from wall" function will return the null vector because it does not need the Boid to steer away from the wall.

These behavior distances is the furthest distance that is needed before activating this behavior, for instance, a Boid that have three neighbors, where one of them is 100 px away, the second one is 150 px away while the third one is 300 px away. The first Boid will be taken into consideration when calculating all of the three behaviors; cohesion, alignment and separation because it is very close. The second Boids will not have any influence on the separation behavior, but it will influence the alignment and cohesion vector. The third one is too far away, and will be ignored when calculating the three behaviors. These distances are illustrated in figure 3.5. The Boids in the simulator has a velocity cap of 40 px per frame, moving faster than 40 px would make the Boids move too fast, and it would be hard to clearly see the details of the movements.

Each vector from each behavior is multiplied with a factor that determines how much impact that vector will have on the final acceleration vector. The list provided below is the weight multiplied by the behavior vector. Or the W_i found in equation 3.1.

cohesion vector is multiplied by 2

alignment vector is multiplied by 2

away from wall vector is multiplied by 2

avoid obstacles vector is multiplied by 3

separation vector is multiplied by 3

3.5 Differences between the physical experiment and simulator

Although the robot's behavior are trying to mimic the behaviors found on the simulated Boids, some major differences will still be imminent. This section will explain the biggest differences between the physical experiment and the simulator.

The physical robots are trying to mimic the behavior of the Boids created in the simulator. However, a physical robot is different than the Boids created in the simulator by nature. As discussed in section 3.2, the robots are a type of differential wheeled robots, which means that it can only move forward or backwards, turn on the spot or move and turn at the same time. It can not move sideways.

The Boids in simulator software did not have any direction, they were able to move freely in all 360°direction. Which means that if a Boid in the simulator were to move in one direction, it could change its momentum and move the other direction without having to stop and turn to that direction. The robot, on the other hand, would need to turn 180°before moving forward. The robot is able to reverse its motor to drive backward, but these robot are not allowed to move backward, and thus have to turn around to move in another direction.

In the simulator, each Boid will know exactly where everything is placed. That is, each Boid knows where all the other Boids are, including itself. It also knows where all the obstacles are. The Boids will have real time access to everything that can be seen on screen. Every Boid will update its perception every frame, that is approximately 60 times every second.

The robots, on the other hand, will receive data from the watcher about all the other robots. But due to inaccuracies from the camera and the image processed, the robots only knows vaguely where in the sandbox it is located, and where the others are. The time it takes for a robot to receive new information from the watcher takes roughly one second from the last time it received data from the watcher. If the robots are moving between the time it receives data, it will not know where it is before it receives new data from the watcher software. The camera tracking software is able to see all the robots due to the two post-it notes on top, but obstacles found in the sandbox do not contain any color or anything that the camera can track. Obstacles are therefore ignored by the camera tracker software because it does not support tracking of anything other than the robots. That is why the robots need to use their distance sensors in the front to detect the obstacles.

The biggest most visual difference between the physical experiment in the sandbox and the simulated Boids is the size and speed of the entities. The size of the simulator is 1024 px times 1024 px. While the sandbox is 151.6 cm wide and 123.9 cm long, which corresponds to 800x652 px in the watcher software. The Boids on the simulator moves quite fast and can use the extra space to move around, while the robots are confined inside the sandbox. The biggest reason to use 800x652 px for the watcher software is because the behavior neighborhood distances works well for this size.

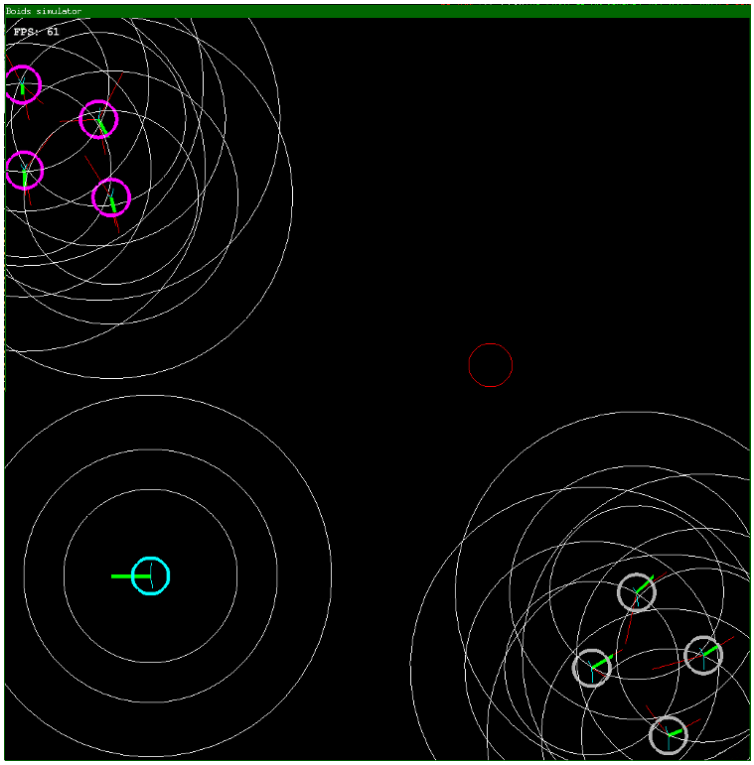


Figure 3.5: Simulated Boids with neighborhood distances visualized

As it will be explained later in section 4.2, the parameters for the simulated Boids and the physical robots are a bit different. The figure 3.5 shows the approximate distances for each behavior, the inner thick colored ring is the outline of the Boids. The inner thin white ring, illustrates the separation distance, the robot do not try to separate itself from the other robots if they are outside this

ring.

The red lines in the figure, are the vectors from each behavior before it is multiplied with the weight. The thickest green line indicates which way the Boids are moving, the longer the line, the faster the Boids are moving.

The second most outer ring indicates the alignment neighborhood, if there is another Boid inside this ring, then both of them will try to align and move in the same direction.

The outer circle is the cohesion distance, the Boids outside of this box will not attract each other.

In the figure 3.5, three types of Boids "family" exist, each one has their own color. Boids will flock together and align themselves only if they have the same color, that is the light gray Boids will flock together with the other light gray Boids. The magenta Boids will only flock together with the other magenta Boids.

When running the simulation for the experiment, all four of the Boids were the same color, therefore just one flock of Boids would emerge instead of forming multiple flocks. The current implementation of the Boids algorithm on the robots does not support multiple groups of Boids. There is no point in implementing this functionality when there are only four robots running at the same time, because it will be very hard to distinguish the difference between a family group flock or if there is just a robot astray from the flock.

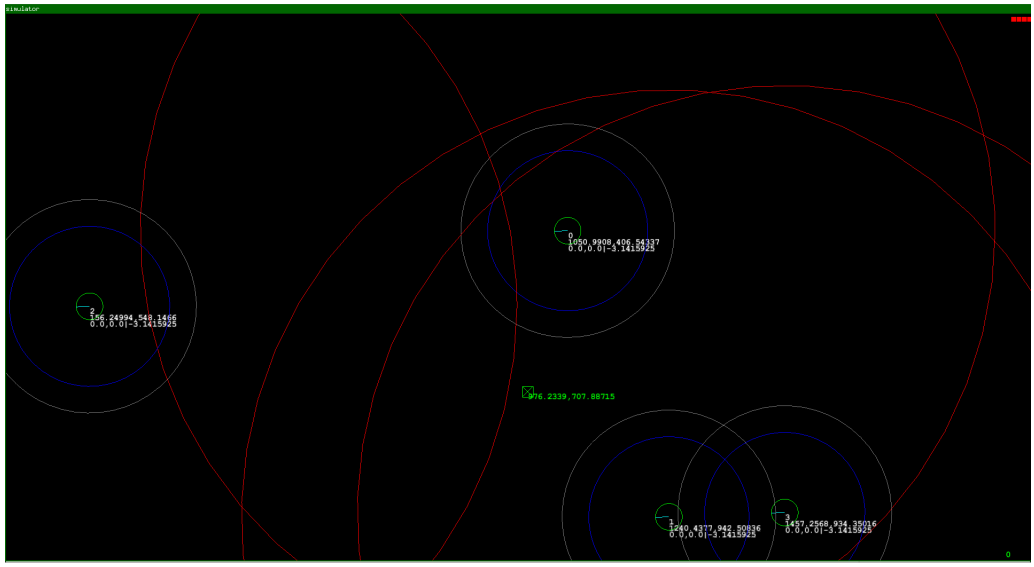


Figure 3.6: Watcher software with neighborhood distances visualized

Figure 3.6 illustrates the distances that the robots use. The exact numbers can be found in section 4.2. The distances have been colored for convenience because it is hard to see which ring represents which behavior. As in the simulator, the most inner ring, which is green in this figure represents the outline of the robot. The figure shown is not the same size as the watcher software that is used to run the experiments, the resolution of the watcher software on the figure has a resolution of 1024x1920 px which is a lot bigger than the resolution on the watcher software used in the experiment. This is to make it easier to see and distinguish the distances.

The next ring, which is illustrated with a blue color, represents the separation distance. And the gray ring shows the alignment neighborhood distance. These two distances are the same in the simulator and on the robot.

The biggest red ring is the one that is the most different from the simulator's. The reason the cohesion distance is so large is mainly to force the robots to flock together even when they are far apart. The distance almost covers the whole sandbox, that way the robots will try to flock from almost anywhere in the sandbox.

Information about each robot is also provided by the numbers next to each robot. The first row shows us the ID of the robot, and the serial port that belongs to that robot if there is a Bluetooth connection and it is assigned. The second row shows us the position of the robot, while the third one shows velocity and the angle of the robot.

The upper right red squares indicate whether the Bluetooth connection is still functional or if the Bluetooth have timed out. The squares in the figure are all red because this is just a debug run where there is no Bluetooth connection.

Robots will be pushed away if a different robot moves onto it. Two robots can not occupy the same space at the same time. Simulated Boids can overlap without affecting each other. To summarize the difference between the robots and the simulated Boids, a list of the parameters will be shown here:

Neighborhood distances		
Behavior	Simulator	Robots
Cohesion	250 px	800 px
Alignment	175 px	200 px
Separation	120 px	150 px
Away from wall	175 px	200 px
Obstacle	120 px	150 px

Table 3.1: Table comparing neighborhood distances between robots and simulated Boids

Vector weights		
Behavior	Simulator	Robots
Cohesion	2x	2x
Alignment	2x	2x
Separation	3x	4x
Away from wall	2x	1x
Obstacle	3x	Does not exist

Table 3.2: Table comparing weights applied to behaviors between robots and simulated Boids

The reason the cohesion distance for the physical robot are so different than the one in the simulator is because the robots are slow to flock together and they might get stuck at a corner when the sensors, so a big cohesion distance makes the robots flock together faster. If the robots' cohesion distance were to be the same as the cohesion distance for the simulated Boids, they might stay too long in one place, and the robots might not flock. As shown in later sections, the robots flocks with an average distance of 200 px, therefore the cohesion distance has to be sufficient larger than 200 px for it to be effective.

The Boids in the simulator do not need a cohesion distance of 800, because they wander a lot around even if there are no neighboring Boids around them. When they have wandered around for a while, they will eventually find another Boid that is near enough and they will start to flock together.

The weights on the behavior vectors for the robots are a little bit different than the one found in the simulator. The robot's behavior weight are kept as close to the Boids as possible, but some weights had to be tweaked for the experiments.

As seen from the table 3.2, the avoid obstacle vector does not exist for the robots because the watcher and the camera tracking software can not find nor distinguish obstacles from the surroundings. This feature is not implemented in the camera tracking software. The robot needs to know the size of the sandbox beforehand because the camera or the watcher software do not provide this information to the robot. To compensate for the lack of obstacle information, the robot uses their distance sensors to find out where the obstacle are located and avoids them.

As seen in the list above, the two vectors that have the highest influence on the robot is the separation and the obstacle avoidance behavior. These two behaviors needs to influence the robot a lot more than the other behaviors because they are only activated when the robot is very close to another robot or an obstacle.

3.6 How results are made

This section will go through the process of how the results found in section 4.3 are made.

The results are created by writing to a text file in the form of a .csv file, which can be open directly by spreadsheet software. A csv file is simply a text file with comma separated values, a row in the csv file would correspond to a row in the spreadsheet, and a column in the spreadsheet software is separated by a comma in the csv file. The distances between the entities, the difference between their angle and the velocity are the features that the watcher software will be graphing.

The watcher software has no way to find the actual velocity of the robots because the wheels still move when the robot is turning on the same spot. We want to find out if the position of each robot has changed over a given time interval. The way the watcher software calculates the velocity of each robot is to save the old position of the robot, then compare it with the new position and see how far off it is. The formula used to find the mean velocity value of all the robots at a time step is defined as:

$$\mu_{velocity} = \frac{1}{N}\Sigma_{i=1}^{N}|P_{i_{new}} - P_{i_{old}}| \tag{3.8}$$

where:
$p_{i_{new}}$ = the new updated position of robot i
$p_{i_{old}}$ = the old position of robot i from last time step
N = number of robots used, in this experiment $N = 4$
$\mu_{velocity}$ = the mean of the velocities in that time step

The average distance seen in section 4.3 are calculated the same way for both the physical experiment and in the simulator. Each entity's position is found and compared with the position of all the other entities. The length of the distance between each of them are used to find the average and the standard deviation. The formula to find

$$\mu_{distances} = \frac{1}{\binom{N}{R}}\Sigma_{1}^{\binom{N}{R}}|P_i - P_j| \tag{3.9}$$

where:
P_i = the position of robot i, and $i \neq j$
P_j = the position of robot j, and $i \neq j$
N = the number of robots or Boids used
R = the amount of entities that are being compared each time, in this experiment we only measures the distance between two entities at the same time, therefore $R = 2$.
$\binom{N}{R}$ = the combination operator, this corresponds to $\frac{N!}{R!(N-K)!}$

$\mu_{distances}$ = the mean of the compared distances in that time step

The same procedure was applied for the angle of each entity:

$$\mu_{angles} = \frac{1}{\binom{N}{R}} \Sigma_1^{\binom{N}{R}} |A_i - A_j| \tag{3.10}$$

where:
A_i = the angle of robot x, and $i \neq j$
A_j = the angle of robot y, and $i \neq j$
μ_{angles} = the mean of the compared angles in that time step.

To find the standard deviation of the velocity, the formula in equation 3.11 was used, but a modification was done for the standard deviation of the distances and angles as seen in equation 3.12 because we had $\binom{N}{R}$ number of distances and angles. The reason for $\binom{N}{R}$ number of distances is because this is the number of comparisons between each robot. For these experiments, $N = 4$, and $R = 2$ because we use 4 robots and there is a comparison between two robots at the same time.

$$\sigma = \sqrt{\frac{1}{N} \Sigma (X - \mu)^2} \tag{3.11}$$

$$\sigma = \sqrt{\frac{1}{\binom{N}{R}} \Sigma (X - \mu)^2} \tag{3.12}$$

Chapter 4

Experiments and Results

In this chapter, the results from the experiments will be presented and discussed.

Section 4.1 explains how the experiments will be executed. The setup of the experiment is explained in section 4.2, it explains the setup of each scenario. Section 4.3 contains the actual results created by running each scenario. Section 4.4 will discuss the results and will compare the results from the physical experiment with the results from the simulator.

4.1 Experimental Plan

To be able to test whether the robots are behaving like the Boids, results from both the physical and the simulated experiments will be plotted to compare the difference between the Boids and the robots. Three scenarios or three different starting position will be used as explained in section 4.2.

For the physical experiment with the robots, each scenario will be ran ten times to generate the graphs for the results, while only five runs will be used for plotting the behaviors of the simulated Boids. If the simulator runs a scenario twice, that is two different simulations with the same parameters and starting position of the Boids are ran, the Boids in the two simulations would start by moving in the exact same way. All the runs on the simulator are very similar to one another. That is why only five runs were used to generate the results.

The robots did not behave as deterministic as the Boids, there are many factors that can affect the movement of the robots. For example if there are other people moving around in the room where the experiment took place, the light setting could be affected enough so the camera would not be able to recognize the two post-it notes on top of the robot, thus the camera tracking software would not be able to recognize the robots. To be able to remove all this noise and

randomness from the results, ten runs were used to generate the results when running the experiment on the physical robots instead of five runs.

All of the results for each scenario was then added together and divided by the number of runs used on that experiment (five runs on simulator and ten runs on the physical experiment) to make the plot show the average of all the runs. Namely the distance between the entities, the angle difference between the entities and the velocity are the features the software will use when creating the results. The distance between the entities is used to determine if the entities are flocked or not, the lower the distance, the closer they are to one another. The angle difference are used to check if they are facing in the same direction, if all of the entities are looking in the same direction, the angle difference would be 0. Only looking at the plot for the angle difference and the distance between each entity is not sufficient to determine whether the Boids behavior is working or not, because the robots might flock together, face the same direction then stand still there. That is why the average velocity of the entities are plotted as well.

4.2 Experimental Setup

This section explains the setup of each scenario, where each robot is placed in the sandbox, which way it is rotated and where the obstacle is placed. Each scenario has a purpose to demonstrate, which will be explained more in detail. In this experiment, four robots and one obstacle are used inside a sandbox. Each robot needs to have an extra layer on top of them so the red and green post-it note does not fall off, as seen in figure 3.3a. The robots are moving inside a sandbox, which is watched by a web camera from above.

For this project experiment, three scenarios have been used to create the results seen in section 4.3.

4.2.1 Scenario 1

The first scenario is shown in figure 4.1, consists of four robots placed on each corner of the sandbox. The reason for placing each of the robots in each corner of the sandbox is that each robot will be as far from each other as possible. This scenario should demonstrate that the entities are able to flock together, and then stay together as a flock. There is one obstacle placed nearby the middle of the sandbox, the obstacle is placed near the middle to ensure that the robots would encounter it at least once. This scenario would be able to see if the robots actually flocked together like they are supposed to do, and at the same time would be able to avoid the obstacle without bumping or crashing into it. The robots should also move together after they have flocked together.

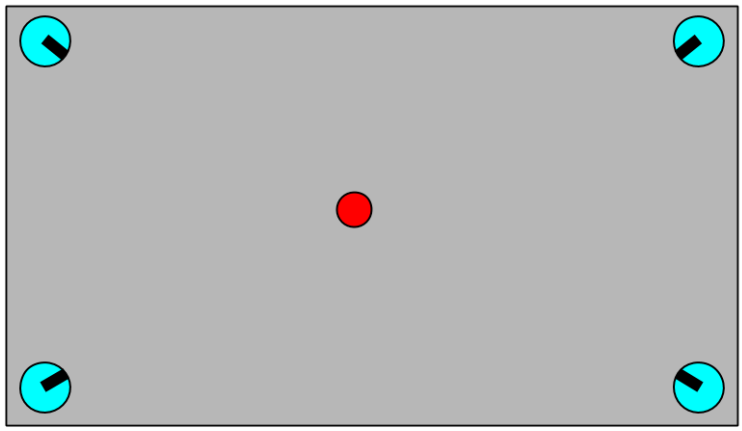


Figure 4.1: Scenario 1, all the robots are placed in each corner with one obstacle

4.2.2 Scenario 2

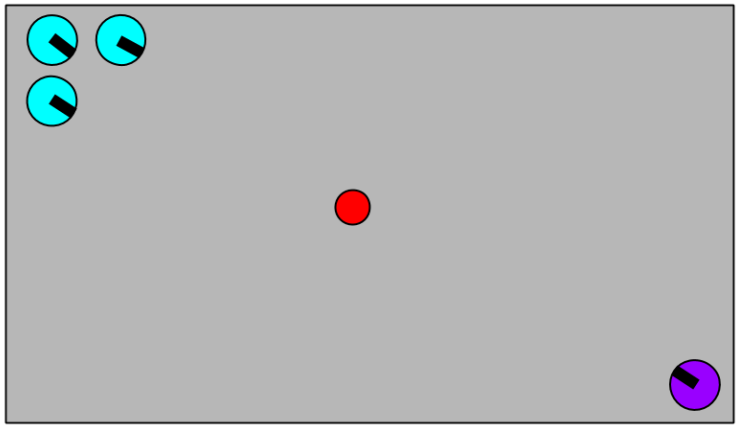


Figure 4.2: Scenario 2, three robots in one corner, and the last one on the opposite corner

The second scenario puts three of entities in one corner, and the last entity is placed on the opposing corner as seen in figure 4.2. However the last entity that is placed on the lower right corner by itself will not move, in the case of the

physical robot, it will not be turned on. The reason that this entity is a sitting duck placed on the opposite corner of all the other entities is that this robot will act as a goal for the other entities. The obstacle is placed between the lonesome robot and the three robots that are clumped up together. The idea behind this setup is that the three robots that start together would try to move to the one that are stationary because they need to flock together. The robot that start by itself does not move because it is not turned on.

The obstacle in the middle will hinder the robots from moving in a straight line to their goal, which forces them to choose a way around it. The robots move around the obstacle when they approach it. As explained in section 2.1.9, the robots will turn either right or left randomly when moving directly towards an obstacle. The robots will probably move around the obstacle on each side of the obstacle, and flock together again on the other side when they have moved past the obstacle.

4.2.3 Scenario 3

The third scenario is a scenario where the robots are placed randomly around in the sandbox. The two first scenarios were designed for a specific purpose. The purpose of the third scenario is created to demonstrate that the Boids behaviors are still intact, even when the robots are placed randomly in the sandbox. The position of the robots, where it should be placed and which way it is pointing was generated randomly by a random number generator. The starting point for this scenario is illustrated in figure 4.3.

This scenario looks a little bit like the first scenario, the robots are laid out in the shape of a square. They are closer to each other than the robots in the first scenario. One of the robots are separated from the other by starting behind the obstacle, the robot can not move directly to the other three robots without first moving around the obstacle.

All the scenarios explained in this thesis uses the same sandbox, which is a sandbox with the size of 151.6 cm wide and 123.9 cm long. The only difference between the scenarios is the placement of the obstacle, the direction the robots are facing and its placement. Each scenario was ran ten times to generate the data seen in section 4.3. In between each run, the robots had to be placed manually back into their starting position before a new run could take place. To keep the data as consistent as possible, everything else would stay exactly the same. For the experiments on the simulator, only five runs will be used. Boids have the same starting position and the same direction each time, each run is therefore not very different from the other ones.

The camera used in this experiment is a web camera. To be able to get clear stable video feed images from the web camera, the settings for the camera had to

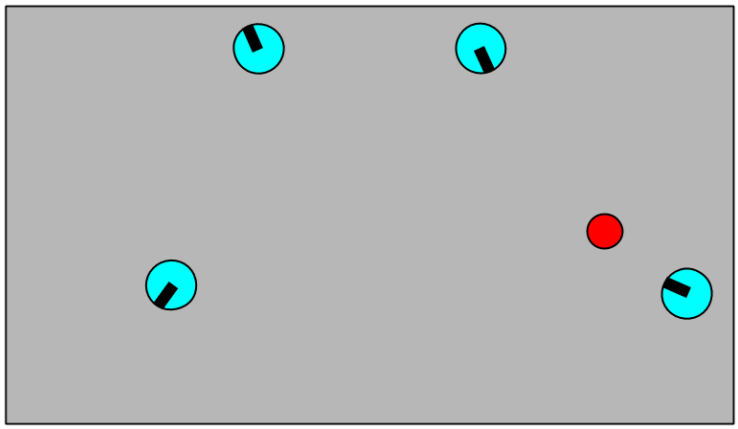


Figure 4.3: Scenario 3, robots and obstacle randomly placed in the sandbox

be manually set up. The most important setting is to disable auto focus. Auto focus makes the images blurry, and the camera tracking software will not be able to detect red and green colors which define the robots. 50 Hz power line frequency was used instead of 60 Hz to eliminate flickering on the camera feed. The other settings are not that important, as long as the web camera is able to provide a decent looking image so the camera tracker software is able to recognize the two colors on top of each robot and thus identify the robots.

4.3 Results

The Boids algorithm are supposed to keep the robots flocked together and preferably they should face the same direction as well. The watcher knows where each robot is, and it knows which direction each robot is facing. The watcher measures the distance between each robot and the angle difference between the robots every five frame or twelve times each second, it then calculates the mean and standard deviation of the distances and angles and saves it to a file. The mean and standard deviation of the velocity is recorded as well, in the simulator the velocity is measured directly by getting the velocity vector on each object, for the physical robot, the change in position is measured instead.

To keep the data as consistent between each run the watcher stops all the robots and saves the data file exactly three minutes or 180,000 milliseconds after the robots have started to move. The distance measured are in pixels. The measurement of the sandbox is 151.6 cm wide and 123.9 cm long. The watcher

software creates a window that has a resolution 800x652 pixels, which means that 1 cm is approximately 5.3 pixels on the screen. The measured angles are shown in radians.

In the upcoming figures, the results from the various runs will be shown. On the x-axis, the time will be shown, and the y-axis displays various types of data depending on the figure. The time shown on the x-axis displays the time iteration, not seconds. One iteration takes five frames, the software runs with a 60 frames per second. Which means that twelve iterations on the x-axis corresponds to one second. The velocity is measured every iteration, the velocity graph is mostly used as an indication to whether the robots are moving or not. The velocity graphs can be used as an indication whether the robots are moving fast or slow, but it can not be used to reliably tell the exact velocity of the robots. The velocity graph for the simulated Boids is more precise because it is displaying the exact velocity of the Boids.

The upcoming figures will show graphs of various types, two similar ones will be displayed for each scenario. One is the results from ten runs on the physical experiment with the robots, and the other one is the results from the experiments ran in simulation. In section 4.4 a discussion of the results will be presented, it will contain an explanation as to why some of the graphs might be different.

4.3.1 Results from scenario 1

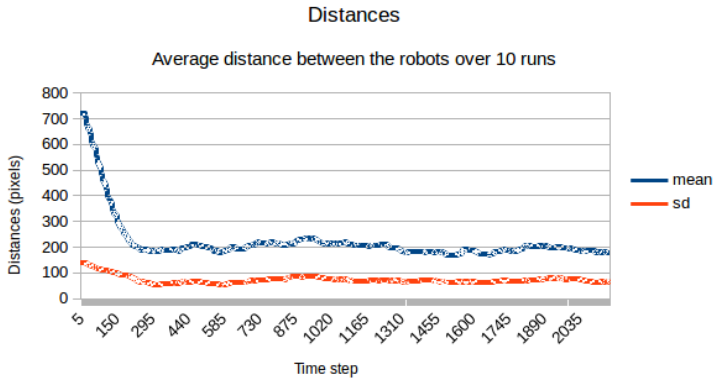


Figure 4.4: Results from 10 runs averaged on scenario 1, robots

The first scenario was designed to check if the robots were able to flock together, the robots were placed on each of the sandbox's corner, as far from each

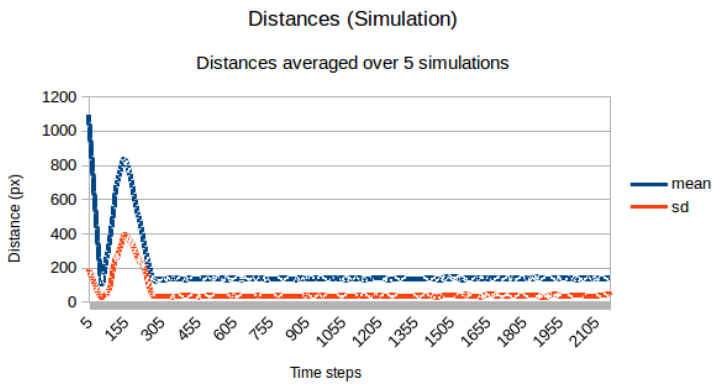


Figure 4.5: Results from 5 runs averaged on scenario 1, simulator

other as possible. From the figure 4.4 we can see that the robots are flocking together quite fast, they start out on each corner of the sandbox then move towards the center of the sandbox, this is shown in the graph by the first drop from 700 px to around 200 px. This corresponds to roughly 132 cm to 37 cm. The time it takes for the graph to drop from 700 px to 200 px is around 300 iterations, which equals 25 seconds. The simulated Boids behave differently in the same scenario, as seen from figure 4.5, between time step 40 and 305, the graph shows a local maxima. Because all of the Boids are moving towards the middle due to the velocity they have at the start, the distance between them will decrease. But one of the Boids is moving directly toward the obstacle and facing it directly, when it comes too close, it will be "pushed" directly in the opposite direction. The Boid that is being "pushed" in the opposite direction by the obstacle is too far away from the other Boids for the cohesion behavior to be active, and therefore moves away from the rest of the flock. This is the reason there is a peak in the graph.

In this scenario, the difference between the angles of the robots starts at 2 radians. This is when all of the robots are facing towards the center of the sandbox. After the robots have moved towards the center, they will start to turn around and the difference between the robots decreases. However, the robots are not able to face the same direction entirely, the lowest difference is still above 1.2 radians, which is approximately 68 degrees. The robots do rotate a lot on the spot, which affects the results seen in the graph. The simulated Boids' angle also starts at approximately 2 radians, then slowly decreasing to 0.3 radians. When the graph has flattened out at 0.3 radians, it seems like all the Boids are facing in the same direction.

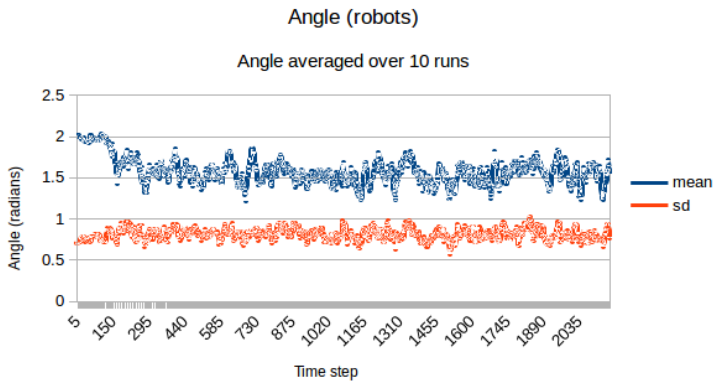


Figure 4.6: Results from 10 runs averaged on scenario 1, robots

Both the robots and the Boids have high velocity at the start when they are moving towards the center. When the Boids reaches the center of the simulated space, they will need to adjust their velocity to change direction, and when all four Boids move together as a flock unit, they need to adjust their flight direction all the time. The velocity graph for the robot in this scenario has the same general outline as the simulated Boids; the graph shows a high velocity at the start, then drops down before stabilizing at a given range. The robots' velocity jiggles a lot, ranging 0.1 to 3.6, there might be various reasons for this result will be discussed more in depth in section 4.4.

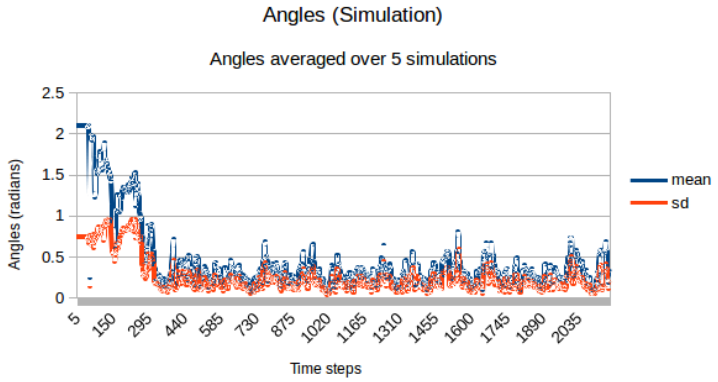


Figure 4.7: Results from 5 runs averaged on scenario 1, simulator

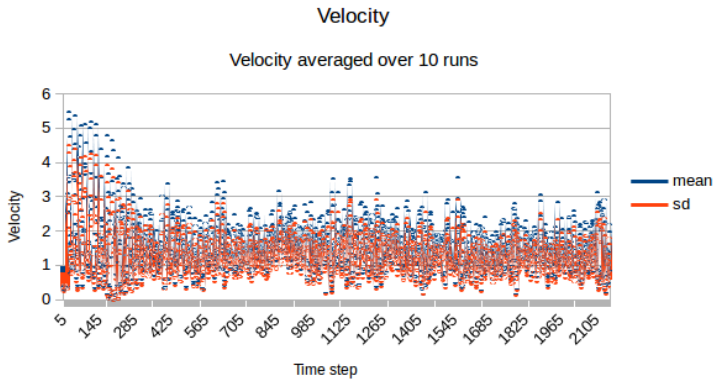


Figure 4.8: Results from 10 runs averaged on scenario 1, robots

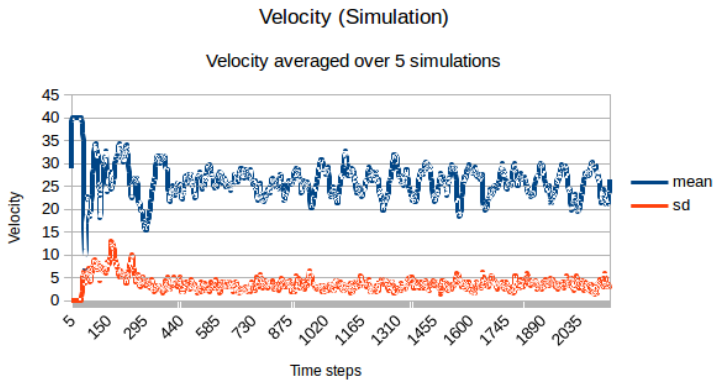


Figure 4.9: Results from 5 runs averaged on scenario 1, simulator

4.3.2 Results from scenario 2

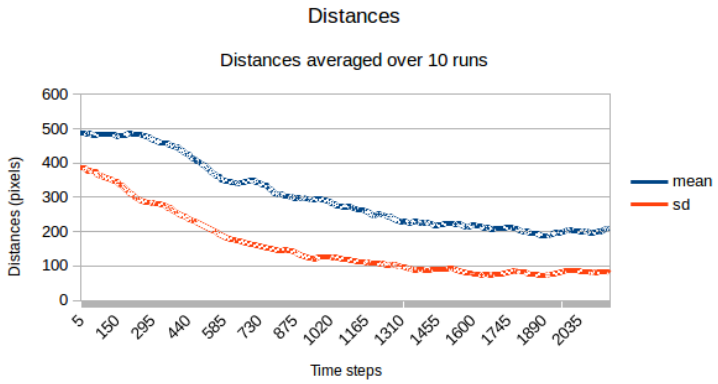


Figure 4.10: Results from 10 runs averaged on scenario 2, robots

Earlier demonstrations of Boids have shown us that when a flock of Boids is moving towards an obstacle, the Boids would split up to sub-groups, fly around the obstacle on both sides before merging together when they have passed the obstacles. This scenario is designed to check if this behavior is still intact on the robots.

In the second scenario, it takes a bit longer for the distance to drop to 200 px, around 1310 time steps, which is approximately 110 seconds. Three of the robots are already in a flock while the fourth one is astray from the flock on the opposite side of the sandbox. The three robots that have already flocked together will try to stay together, they do not want to move all the way to the other side to flock with one single robot. If all of the robots were moving, the single robot would move towards the three robots and they all would flock together faster.

When running scenario 2 on the simulator, the Boids move towards the obstacle and around it. When they are near enough to the non-moving Boids, the cohesion behavior will activate and the Boids will try to flock, but all the other behaviors are trying to push the three moving Boids away from the non-moving Boid. After being pushed away, they will keep moving in the same direction and move all the way around the screen. This movement around the screen creates the oscillated graph plot seen in figure 4.11.

In the second scenario, one of the entity is stationary, and thus its angle does not change, but its angle is still accounted for when plotting the graph. The angles shown in the graphs for the second scenario starts in the lower end, then

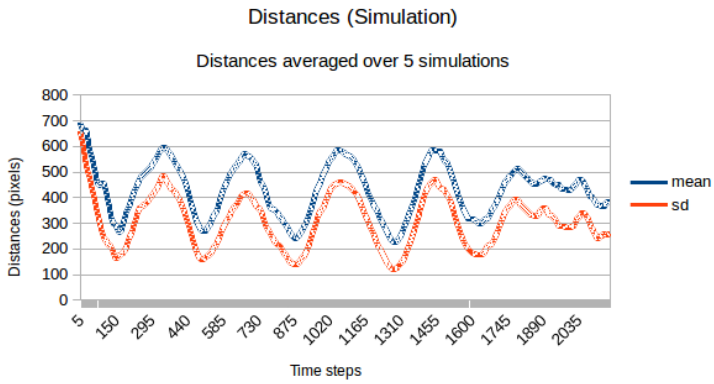


Figure 4.11: Results from 5 runs averaged on scenario 2, simulator

increases. Three of the entities starts by facing the same direction while the last stationary one faces in the opposite direction. That is, all off them faces toward the center as seen in figure 4.2. When starting the runs for the second scenario, the robots started to turn around immediately, thus increasing the difference-angle shown in the graph. One of the robot starts in the upper left corner, it is covered by the two other robots, having no way to move out of this corner without colliding into the other robots, its only option is to turn around on the spot.

The three moving Boids starts off by moving towards the center of the screen, they then move around the obstacle. After reaching the stationary Boid, they are pushed away, which forces them to turn away from the stationary Boid. The Boids angles follow a wave pattern, much like the pattern found in the distance graph for this scenario.

Both of the velocity graphs in the second scenario are considerably lower than the ones found in the other two scenarios. This is expected because one of the four entity is stationary, i.e. non-moving.

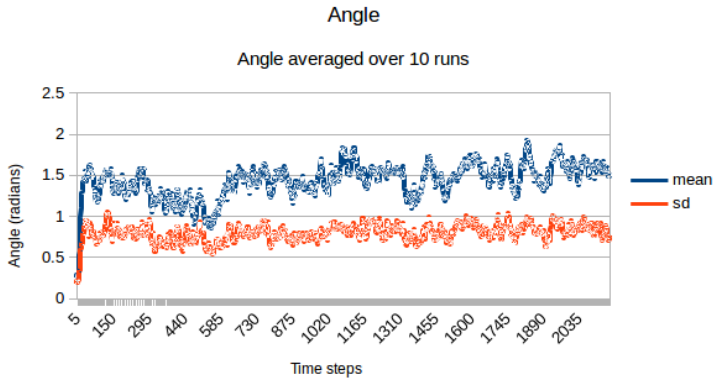


Figure 4.12: Results from 10 runs averaged on scenario 2, robots

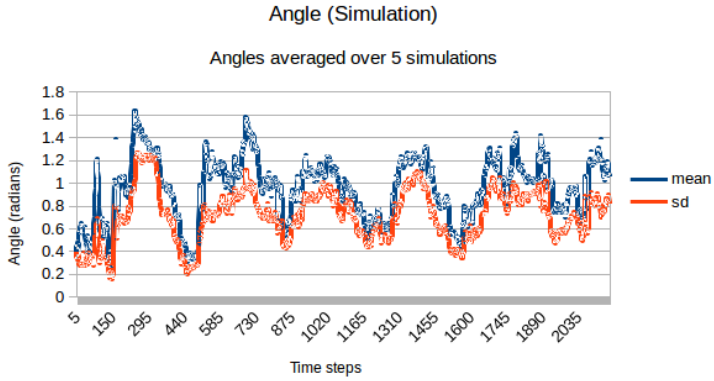


Figure 4.13: Results from 5 runs averaged on scenario 2, simulator

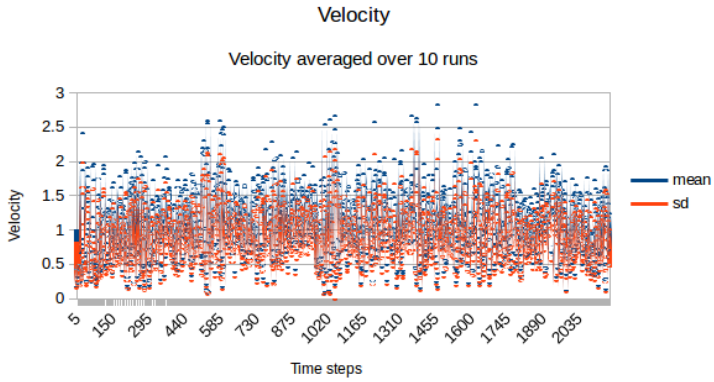


Figure 4.14: Results from 10 runs averaged on scenario 2, robots

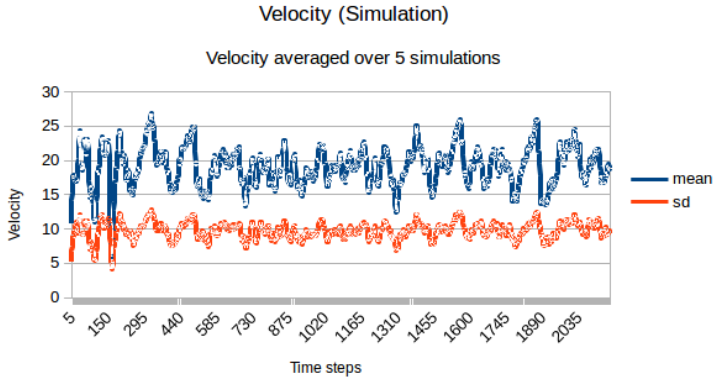


Figure 4.15: Results from 5 runs averaged on scenario 2, simulator

4.3.3 Results from scenario 3

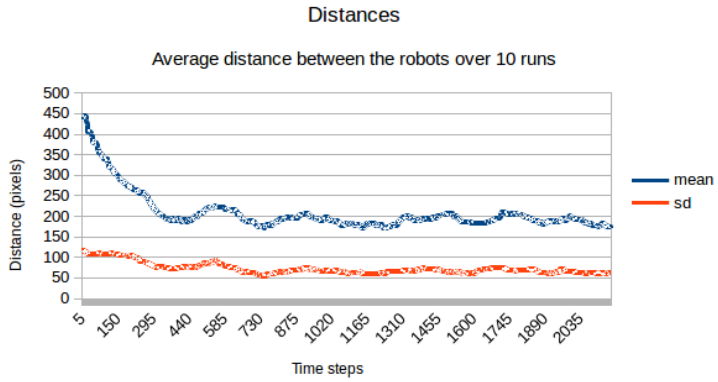


Figure 4.16: Results from 10 runs averaged on scenario 3, robots

The third scenario consists of entities that are placed randomly, the purpose behind this is to check that flocking and collision avoidance are still intact. Both the first and the second scenario were designed to check a specific behavior, this third scenario is not testing any specific behavior, but it will check that the robots are still behaving the way they should be. The robots should still flock and avoid obstacles in this scenario like they have done in the other scenarios.

The results for the distances between the robots in the third scenario is similar to the one in the first. They move closer to each other and then stay as a flock for the rest of the time. In the simulator, the Boids starts by moving further away from each other, before converging together. The Boids that starts on the lower right are behind an obstacle, which pushes it away from the other ones. The lower left Boid has a start velocity away from the robots, and due to the short cohesion distance radius, it will not move towards the other Boids but move away from them instead. The other Boids are not in range for the cohesion behavior to activate. However, it does not take too long before the Boids are able to flock together, as seen from figure 4.17 the Boids are starting to move towards each other after 40 time steps, and are fully flocked together at time step 150.

The robots do not have the same problem as the Boids in this scenario, they have a much wider cohesion distance. Obstacles do not push the robots in the opposite direction either. When a robot tries to figure out which direction it is going to move in, it will ignore the walls and the obstacles. If there is an obstacle in front of a robot, it will turn away from the obstacle. But the next time it

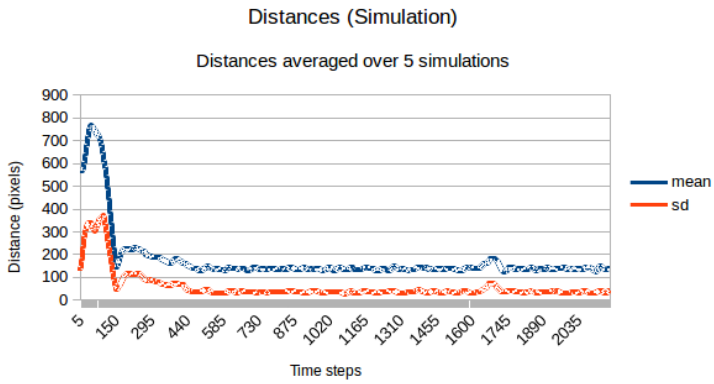


Figure 4.17: Results from 5 runs averaged on scenario 3, simulator

calculates where it wants to go, it might calculate the direction it wants to go is through the obstacle, thus the robot will turn toward the obstacle then realize that there is, in fact, an obstacle in front of it and it has to turn away again. The robot might get stuck on the same spot because of this turning behavior, but it will eventually force itself to move around the obstacle. After turning on the same spot back and forth for a while, the robot is forced to move forward, even if it is currently facing away from the direction it intended to move to. In the meantime, the other robots will move towards the robot that has been stuck behind the obstacle.

The angle and velocity graph of the third scenario is relatively similar to the graphs in the first scenario.

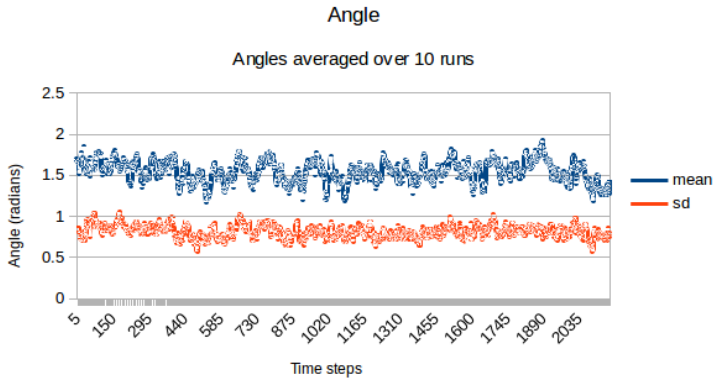


Figure 4.18: Results from 10 runs averaged on scenario 3, robots

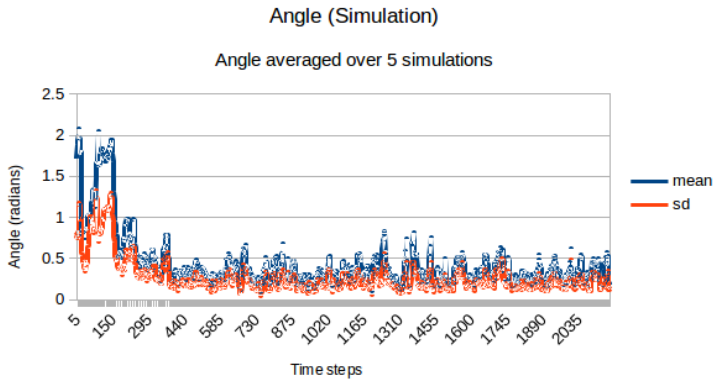


Figure 4.19: Results from 5 runs averaged on scenario 3, simulator

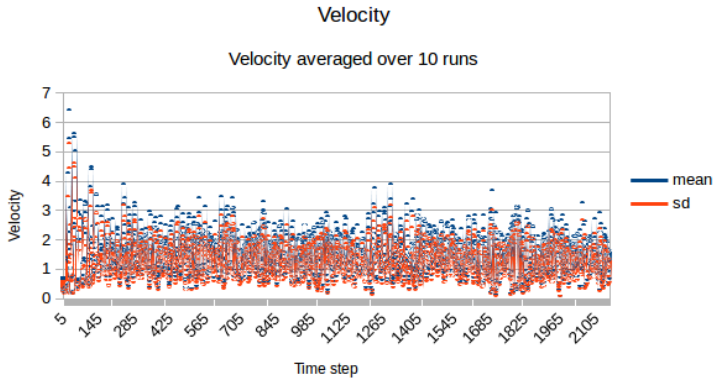


Figure 4.20: Results from 10 runs averaged on scenario 3, robots

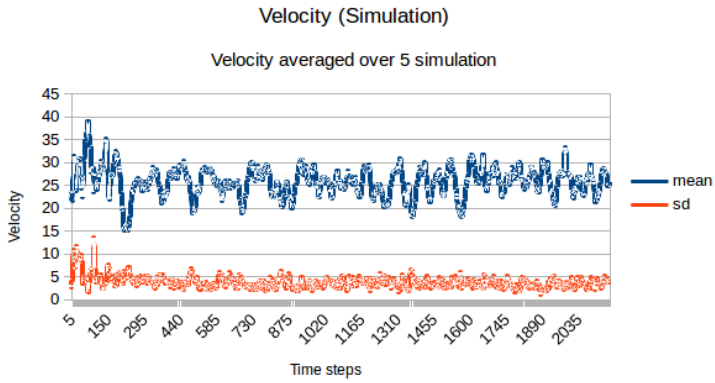


Figure 4.21: Results from 5 runs averaged on scenario 3, simulator

4.4 Discussion

This section will mainly focus on explaining the overall data from the results, why there is a difference between the physical experiment and the experiments done in a simulator.

The robots are able to flock faster in the first and third scenario than the robots in the second scenario. This might be because each of the robot start out by themselves and will therefore seek the other ones. The equilibrium distance which the Boids are flocking at, seems to be around 120 px to 150 px. As mentioned earlier, the separation distance of the Boids is 120 px. The separation distance for the robot is slightly increased compared to the Boids', the distance is 150 px instead of 120 px. The robots are not only influenced by the separation vector for collision avoidance, but they also use their distance sensors as well. So it is expected that the equilibrium distance between the robot is not exactly at 150 px, according to the graphs, the robots' flocking distance stays around 200 px.

The angles measured seems to vary a lot, even if the alignment behavior tries to make all the robots face the same way. But the overall trend of the robots seems to be that the angles lingers around 1.5 radians on average for all three scenarios, which is pretty high.

The simulated Boids are able to face in the same direction as the other Boids, starting with an angle difference of 2 radians and slowly dropping down to 0.5 radians. This holds true on the graphs shown for the first and third scenario. The angles in second scenario are varying a lot, ranging from 0.4 radians to 1.6. This happens because one of the Boid is not moving, its angle is always 0. While the other three is moving around and their will range from $-\pi$ to π depending on the direction they are moving.

If the allowed space to travel were bigger, and there were no obstacles. The Boids would be able to move without needing to turn, and the angle between them would stay consistently low. However the allowed space for the robots to travel is limited and there is an obstacle there as well. Whenever the robots detects an obstacle or move towards the walls, the distance sensors will make the robot turn around so the robot will not crash. Sometimes the robots will think that the other robots around itself are obstacles as well because it has no way to tell the difference between the robots from an actual obstacle. This is because the distance sensors can not distinguish anything, it only measures a distance and the robots will try to avoid anything that is too near it, thinking that the object it detects is an obstacle.

The velocity for the robot varies a lot, we can see from the graph that the velocity varies from 0 to 6.3. The watcher software logs the velocity data by finding the change in position between two time steps, that is it finds ΔP as

shown in equation 3.6 of each robot every iteration. When a robot is turning to change direction or to avoid an obstacle, it is not moving because it is just spinning in place.

The method used to measure the velocity of the robot is very imprecise, when logging the velocity of a robot with constant max velocity, the results did not stay consistently at a given number.

The result from this test is shown in figure 4.22, the robot is moving at max speed the whole time, but there is no consistent line in the graph. The graph consists of "pillars", meaning that the graph goes from 12 to 0 and back up to 12 again. The big gaps in the graph are when the robot has encountered the wall and stops to turn away.

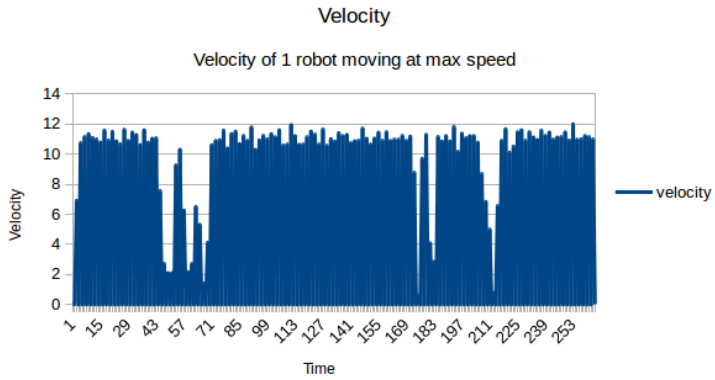


Figure 4.22: One robot moving at max velocity, only turning when facing a wall

The Boids in the simulator never stops, that is why the velocity never drops down to 0, they keep moving in different direction all the time. Being able to move freely in all 360° and they do not need to stop to change direction. The physical ChIRP robot needs to turn around before moving in a new direction. As long as the new direction is off by an angle larger than 1° from the angle the robot is currently facing, then it will stop and turn. All turning takes approximately one second, before the robot is moving again. If the robot will turn a lot, it will only have one second to do so, before it has to move. If it only needs to turn 2°, it will turn first and wait until one second has passed before moving on. The one second delay is introduced to keep the timing somewhat synchronous between the robots, one second delay is long enough for the robot to be able to almost turn all the way around (180°).

The graphs only shows the average distance between the robots, the velocity of

the robots and the difference between their angle. The graphs do not show where the robots are moving or whether they have crashed into anything. Sometimes the robots do bump into the obstacles, or the other robots. The reason is that the robots measures the distances before moving, and not continuously while moving. So if a robot measures the distance in front of it and it does not detect an obstacle in front of it, it will start to move forward. When the robot moves forward it might hit an obstacle or another robot if the distance sensor readings were imprecise. Sometimes robots moves onto the path of another robot while the other robot is moving, the other robot will probably bump into the one blocking its path. The robots do crash into each other sometimes, they do push each other around when they are trying to occupy the same space as the other robots. This might happen if both robots are measuring the distances around them, and either some disturbances make the distance measured inaccurate or the other robots move into the other ones' path between the time they are doing the distance measuring.

When the robots bump into each other, they nudge each other and might scrape the surface of each other. However, none of the paper taped on the robot has been torn apart when the robots were scraping against each other while running the experiment.

The robots only move forward, and they only turn when they need to change direction or if there is an object in front of it that it needs to avoid. Sometimes the robots suddenly stop and rotates on the spot as if there is some sort of object in front of it, even if there is none. Whenever the robot turns, the watcher will not see any change in position, which is the method it uses to log the velocity of the robots. The watcher will therefore log that the velocity of the robot is 0 when the robots turn around on the same spot. The distance measured by the robot's sensors might be imprecise, due to disturbances around in the room that makes the measured distances imprecise. The disturbances can come from the infrared light the other robots sends out when they are measuring distances themselves, which might bounce around in the sandbox and disturb the other robots.

For the camera to see the two colored post-it notes on the robots, the room needs to be well lit. Extra lights were placed around the sandbox to provide enough light. The room where the experiment took place had two large windows, by daytime the sun would shine into the room. The sunlight contains infrared light, if the sun shines into the room and hits the area where the robots roam, the robot will sense the infrared lights from the sunshine and think that there is an object in front of it. The blinds in the room were closed, but some of the sunshine would always shine into the room.

Chapter 5

Evaluation and Conclusion

This chapter will evaluate the results and graphs from the previous chapter. It will mainly focus on the behaviors of the physical robots. Section 5.1 contains evaluation of the results and answers the research questions. How this thesis has contributed to the field of swarming is explained in section 5.2. A suggestion of what needs to be done for future work is presented in 5.3.

5.1 Evaluation

All of the graphs shown in section 4.3 shows that the robots are able to flock together if given enough time. A cohesion distance of 800 px makes the robots move towards one another from almost anywhere on the sandbox, making them flock together. There is a similarity between some of the result data generated from the physical experiment with the robots and the experiments ran on the simulator.

Research question 1: *Can a centralized computer aid a swarm of robots that do not have enough sensors to do the task it was assigned for?*

Each robot was equipped with eight distance sensors and a Bluetooth module for communication. Other types of sensors can be equipped on the robots, but the only other sensor available was light detecting sensors. Using only the distance sensors, the robots will only be able to detect other objects around themselves, but they can not use the distance sensors to distinguish what kind of object it has detected. The structure of the robots makes it hard for the other robots to detect it due to the hollowness of the robot. By using a camera and a centralized computer which the robots can communicate with, the robots gains information about its whereabouts along with its angle and all the other robot's positions.

Research question 2: *Will the Boids algorithm make the robots flock together?*

The robots get the information about its location from the centralized computer and the whereabouts of the other robots. Using this information, the robot will calculate the Boids behavior vectors and move in the direction of the final vector.

Looking at the graphs from section 4.3, we can see that the robots are moving closer to each other, and they stay together with approximately 200 px or 35 cm between each other. In scenario 2, the robots do move on each side of the obstacle due to their placement at the start. If a robot faces an obstacle directly in front of it, they will "flip a coin" to decide whether they are turning left or right, thus a random direction will be chosen around the obstacle if the robot is moving directly toward an obstacle. However, the robots do not move together as one unit towards the obstacle, like the Boids in the simulator do. The most upper left robot seen in figure 4.2 is trapped between the corner and the two robots around it. Having no way to move out at the start, makes it turn around facing the corner of the sandbox. This behavior delays it from moving towards the center and the real obstacle because it has to turn all the way back again. The robots avoids crashing into other objects at almost all cost, it is its highest priority. So when the robots are clumped together, they might spin a lot on the same spot.

5.2 Contributions

This project's main field is artificial intelligence, or more specific the swarm robotics. This thesis contributes to the field by implementing the Boids flocking behavior algorithm on the ChIRP robot, using a few simple behavior makes the robots flock together. By combining the distance sensor readings and the data from a centralized computer, it is able to locate the other robots and move towards them if given enough time. Similar projects had already been done, even without a centralized computer, which is quite interesting. The ChIRP robots can, however, not distinguish between a robot and other objects using only their sensors.

By experimenting on different scenarios, it was possible to see and recognize the typical Boids behavior on the robots. The robots were flocking together, they would spread out and move around obstacles and then flock together again on the other side. Leader following was also possible by manually controlling one of the robots.

This project also helps further explore the limitation of the ChIRP robots, by connecting together the camera tracking software, Bluetooth, and the ChIRP robot. Most of the project involving the ChIRP and the Bluetooth module, only uses the Bluetooth to send direct commands to the robot, telling them exactly

how much each motor should move. By using the camera and sending this data to the robot, it shows that the robots are able to process the data themselves to figure out where they should move.

This experiment also shows that a flock of robots and the whole system involving the camera and the watcher software is able to run autonomously after the setup. It does not need human involvement to work.

5.3 Future Work

This section will look further upon what research question is still unanswered, what can be improved, and what tweaks and improvements needs to be done on this project for the robots to be able to fully mimic the behavior of the Boids.

More robots

The current implementation of the system only supports four simultaneous robots at the same time. A new robot can not be added to the system without uploading new code to each robot, nor can a robot disappear from the system. The camera tracking software loads a configuration file at startup that specifies how many robots will be tracked, this value has to be manually changed by the user and reloaded. The watcher software needs to know which robot ID belongs to which Bluetooth serial port. If one of the robots' Bluetooth connection disconnects or times out, everything has to stop and reconnected again for the system to work.

The main idea behind swarm robots is that it should be able to run continuously without having to reboot or stop the swarm to add a new robot or remove one. If one of the entities in a swarm is defected, injured or not working, the other ones should still be operable.

The current implementation of this project relies on the robot needing to know how many other robots there will be in the sandbox at the same time, because it will need to know how much data it is going to store before processing it. This has to be changed manually in the code and then uploaded to the robot again. A functional robot swarm flock should have more than four robots running simultaneously, and we should be able to add a new robot without re-uploading the code to the robots.

In simulation, it is easy to add new Boids of different type or family as seen in figure 3.5 by the different colors on the Boids. With only four robots running at the same time, there is no point in trying to implement different types of Boids families so subgroups of Boids flocks would emerge. The reason is that it will be hard to see if there is any family groups with only four robots. With more robots, it would be easier and more obvious to see groups of families emerge.

More fluid movement

The current implementation does not punish the robot for standing still nor for only rotating on the same spot. The only escape from continuously spinning on the same spot is a spin counter, which forces the robot to move forward if the robot has been stuck on a spot for five iterations and if there is nothing in front of it.

Each robot also stops when it turns around and whenever they need to change direction. When birds fly together as a flock they do not stop when they need to turn. The robots should be able to move and rotate without stopping.

In the simulator, the Boids held its formation whenever they flocked until they had to change their direction because they were too near an obstacle. The robots do stay near each other to a certain degree. Whenever the robots flock, the distances between each robot varies, they do not have a consistent spacing which the Boids on the simulator do have. The reason this happen might be because the robots are not fully synchronous like the simulator Boids are, that is, each Boids calculates their new acceleration vector direction every frame at the same time, while the robots do not calculate their new direction at the same time as the other robots.

The robots still bump into the obstacle and they still bump into each other. A rewrite of the executive layer should be able to fix this problem. Instead of checking the distances to objects in front of it before it is about to move in a direction, the robots should check and measure the distances continuously. Checking the distance all the time might affect movement of the robot. But if the sensors could be more precise, then the disturbances and noises would not be a problem for the movement of the robot. However, the Boids on the simulator do move through each other if the velocity is high enough.

In scenario 2, the robots do move around the obstacle, but this does not happen at the same time. The three robots stays a while in the upper left corner before moving, and they do not move as a unit to the other side of the sandbox. Some of the runs, they were moving one robot at a time. Preferably these three robots should move as one unit.

After the robots have been flocked together, they would most of the time just move back and forth. The overall movement of the whole swarm did not move a lot, the graphs only show that the robots still move individually back and forth. Sometimes, one of the robots would move out of the swarm, away from the rest. This is most likely a bug because the robot's intended direction is toward the rest of the robots, but it is still moving away from them. This bug, do help the swarm move around in the sandbox as a whole swarm because the robot is forcing the other robots to follow it.

Bibliography

Bai, L., Eyiyurekli, M., and Breen, D. E. (2008). An Emergent System for Self-Aligning and Self-Organizing Shape Primitives. *2008 Second IEEE International Conference on Self-Adaptive and Self-Organizing Systems*, pages 445–454. http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4663447.

Behaviors, S., Green, R., and Reynolds, C. (2001). Steering Behaviors.

Beni, G. (2005). LNCS 3342 - From Swarm Intelligence to Swarm Robotics. pages 1–9.

Blum, C. (2005). Ant colony optimization: Introduction and recent trends. *Physics of Life Reviews*, 2(4):353–373. http://linkinghub.elsevier.com/retrieve/pii/S1571064505000333.

Bonabeau, E., Dorigo, M., and Theraulaz, G. (1999a). *Swarm intelligence*. Oxford University Press.

Bonabeau, E., Dorigo, M., and Theraulaz, G. (1999b). *Swarm intelligence: from natural to artificial systems*. Oxford University Press. http://books.google.com/books?hl=en&lr=&id=PvTDhzqMr7cC&oi=fnd&pg=PR11&dq=Swarm+Intelligence+From+natural+to+Artificial+Systems&ots=yjdmAOyyMI&sig=7KXlsuUObbUmHqQ_4Cy3x5TyVoU.

Brundage, H. (2011). Neat Algorithms - Flocking - Will You Harry Me. http://harry.me/blog/2011/02/17/neat-algorithms-flocking/.

Craig W. Reynolds (1988). Not Bumping Into Things. http://www.red3d.com/cwr/nobump/nobump.html.

Csaba, V., Tamás, N., Gábor, V., Tamás, V., Norbert, T., and Gergő, S. (2014). Outdoor flocking and formation flight with autonomoush aerial robots.

Csaba, V., Tamás, N., Gábor, V., Tamás, V., Norbert, T., and Gergő, S. (2015). Collective Motion of Flying Robots (Drones). https://hal.elte.hu/flocking/wiki/public/en/projects/CollectiveMotionOfFlyingRobots.

Demetri, T., Xiaoyuan, T., and Radek, Grzeszczuk (Department of Computer Science, U. o. T. (1994). Artificial Fshes: Autonomous Locomotion, Perception, Behavior, and Learning in a Simulated Physical World.

Demsar, J. and Bajec, I. (2013). Family Bird: A Heterogeneous Simulated Flock. *Advances in Artificial Life, ECAL 2013*, pages 1114–1115. http://mitpress.mit.edu/sites/default/files/titles/content/ecal13/978-0-262-31709-2-ch167.pdf.

Eberhart, R. and Kennedy, J. (1995). A new optimizer using particle swarm theory. *MHS'95. Proceedings of the Sixth International Symposium on Micro Machine and Human Science*, pages 39–43. http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=494215.

Edwards, C. H. and Penney, D. E. (1988). *Elementary linear algebra*. Prentice Hall International.

Floreano, D. and Mattiussi, C. (2008). *Bio-inspired artificial intelligence*. MIT Press.

Hashimoto, H., Yokota, S., Sasaki, A., Ohyama, Y., and Kobayashi, H. (2008). Cooperative movement of human and swarm robot maintaining stability of swarm. *RO-MAN 2008 - The 17th IEEE International Symposium on Robot and Human Interactive Communication*, pages 249–254. http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4600674.

Jakimovski, B. (2011). *Biologically Inspired Approaches for Locomotion, Anomaly Detection and Reconfiguration for Walking Robots*, volume 14 of *Cognitive Systems Monographs*. Springer Berlin Heidelberg, Berlin, Heidelberg. http://link.springer.com/10.1007/978-3-642-22505-5.

Joselli, M., Passos, E. B., Zamith, M., Clua, E., Montenegro, A., and Feijó, B. (2009). A Neighborhood Grid Data Structure for Massive 3D Crowd Simulation on GPU. *2009 VIII Brazilian Symposium on Games and Digital Entertainment*, pages 121–131. http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5479102.

Joselli, M., Silva, J. R. D., and Clua, E. (2014). An Architecture for Real Time Crowd Simulation Using Multiple GPUs. *2014 Brazilian Symposium on Computer Games and Digital Entertainment*, pages 1–10. http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7000027.

Journal, I. and Robotics, O. F. (1986). Robust Layered Control Syste For A Mobile Robot. (I):14–23.

Kennedy, J. and Eberhart, R. (1948). Proc. IEEE Int'l. Conf. on Neural Networks. pages 1942–1948.

LaValle, S. M. (2006). *Planning algorithms*. Cambridge University Press.

Mataric, M. A. I. L. (1992). designing emergent behaviors: from local interactions to collective intelligence mataric 1992.

Nagy, M., Akos, Z., Biro, D., and Vicsek, T. (2010). Hierarchical group dynamics in pigeon flocks. *Nature*, 464(7290):890–3. `http://www.ncbi.nlm.nih.gov/pubmed/20376149`.

Nathan, A. and Barbosa, V. C. (2008). V-like formations in flocks of artificial birds. *Artificial life*, 14(2):179–88. `http://www.ncbi.nlm.nih.gov/pubmed/18331189`.

Olfati-Saber, R. (2006). Flocking for Multi-Agent Dynamic Systems: Algorithms and Theory. *IEEE Transactions on Automatic Control*, 51(3):401–420. `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1605401`.

Red3d.com (2015). Boids (Flocks, Herds, and Schools: a Distributed BehavioralModel). `http://www.red3d.com/cwr/boids/`.

Reynolds, C. (1999). Steering behaviors for autonomous characters. *Game developers conference*. `http://www.red3d.com/cwr/steer/gdc99/`.

Reynolds, C. W. and Division, S. G. (1987). Flocks, Herds, and Schools: A Distributed Behavioral Model 1. `http://www.red3d.com/cwr/papers/1987/boids.html`.

Russell, S. J., Norvig, P., and Davis, E. (2010). *Artificial intelligence*. Prentice Hall.

Skjetne, C., Haddow, P. C., Rye, A., and Montanier, J.-m. (2014). Robot Intelligence Technology and Applications 2. 274. `http://link.springer.com/10.1007/978-3-319-05582-4`.

Skjetne, Christian (IDI, N. and Montanier, Jean-Marc (IDI, N. (2014a). ChIRP. `http://chirp.idi.ntnu.no/doc/`.

Skjetne, Christian (IDI, N. and Montanier, Jean-Marc (IDI, N. (2014b). ChIRP rev 109 documentation. `http://chirp.idi.ntnu.no/doc/`.

Spears, W., Spears, D., Hamann, J., and Heil, R. (2004). Distributed, physics-based control of swarms of vehicles. *Autonomous Robots*, pages 137–162. http://link.springer.com/article/10.1023/B:AURO.0000033970.96785.f2.

Wagner, M., Cai, W., and Lees, M. H. (2013). Emergence by strategy: flocking boids and their fitness in relation to model complexity. pages 1479–1490.

Walpole, R. E. (2007). *Probability & statistics for engineers & scientists*. Pearson Prentice Hall.

Yongjie, Y., Qidan, Z., and Chengtao, C. (2006). Hybrid Control Architecture of Mobile Robot Based on Subsumption Architecture. *2006 International Conference on Mechatronics and Automation*, pages 2168–2172. http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4026433.

Zhu, Y.-f. (2010). Overview of swarm intelligence. *2010 International Conference on Computer Application and System Modeling (ICCASM 2010)*, (Iccasm):V9–400–V9–403. http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5623005.

Camera tracking configuration file

This file is named chirpObserverConfig.cfg and needs to be loaded by the camera tracking software at startup.

```
# [comPort]
# the id of the comm port to use.
# 0 corresponds to /dev/ttyS0
# 1 corresponds to /dev/ttyS1
# 2 corresponds to /dev/ttyS2
# 3 corresponds to /dev/ttyS3
# 4 corresponds to /dev/ttyS4
# 5 corresponds to /dev/ttyS5
# 6 corresponds to /dev/ttyS6
# 7 corresponds to /dev/ttyS7
# 8 corresponds to /dev/ttyS8
# 9 corresponds to /dev/ttyS9
# 10 corresponds to /dev/ttyS10
# 11 corresponds to /dev/ttyS11
# 12 corresponds to /dev/ttyS12
# 13 corresponds to /dev/ttyS13
# 14 corresponds to /dev/ttyS14
# 15 corresponds to /dev/ttyS15
# 16 corresponds to /dev/ttyUSB0
# 17 corresponds to /dev/ttyUSB1
# 18 corresponds to /dev/ttyUSB2
# 19 corresponds to /dev/ttyUSB3
# 20 corresponds to /dev/ttyUSB4
# 21 corresponds to /dev/ttyUSB5
# 22 corresponds to /dev/ttyAMA0
```

```
# 23 corresponds to /dev/ttyAMA1
# 24 corresponds to /dev/ttyACM0
# 25 corresponds to /dev/ttyACM1
# 26 corresponds to /dev/rfcomm0
# 27 corresponds to /dev/rfcomm1
# 28 corresponds to /dev/ircomm0
# 29 corresponds to /dev/ircomm1
# id = 27

[camera]
# the id of the camera device to use, numbered from 0 and upwards
# deviceId = 1
deviceId = 0
# the width and height of the captured image, denoted in pixels
width = 720
height = 500
# the maximum number of objects to track.
#All objects are ignored if the number of objects
# discovered exceeds this number
maxNumObjects = 1000
#the minimum object area to recognize (in pixels).
#Any object smaller than this is ignored.
minObjectArea = 10
# upper limit of the size of leds compared to
# min(frame width, frame height)
ledSize = 0.01

# settings governing the colors in the captured frame
# the settings are potentially specific to each camera
# linux application "guvcview" provides details
# the settings appear to be sticky, so an application like
#"guvcview" may be necessary to revert to default state
#
# if the value is less than 0, the application won't change
# the camera's setting for that property
# the brigtness of the image
brightness = 0.1
# the contrast of the image (0 - 10 with default of 5)
contrast = 5
# the saturation of the image (0 - 200 with default of 83)
saturation = 30
```

```
# each color is defined by six boundries,
#and is only recognized if the HSV value is within the bounds
[red]
hmin = 120
hmax = 220
smin = 100
smax = 256
vmin = 130
vmax = 256

#hmin = 0
#hmax = 45
#smin= 90
#smax = 256
#vmin = 170
#vmax = 256

[blue]
hmin = 100
hmax = 120
smin = 250
smax = 256
vmin = 250
vmax = 256

[green]
hmin = 31
hmax = 90
smin = 100
smax = 256
vmin = 135
vmax = 256

#hmin = 21
#hmax = 86
#smin= 50
#smax = 256
#vmin = 205
#vmax = 256
```

```
# the size of the virtual feed depicting the
#scene as interpreted by the tracker
[virtualFeed]
width = 500
height = 500

# settings for the UDP network sending
#information about tracked robots
[network]
enabled = true
address = localhost
port = 52346

[general]
# whether to calibrate colors, this allows one to
#find values for the color definitions
calibrationEnabled = false
# the time the system waits after a frame
#has been processed before it fetches a new one
# if this number is set too low,
#the camera won't be able to finish capturing a frame
frameDelay = 10

[tracking]
# number of robots to track
numRobots = 4
# robots are removed if this number of steps pass
#without a pinpointing of the robot's position
evictionLimit = 6
# the maximum speed (distance / frame) of robots in any
#direction compared to the width and height of the bounding box
 robotSpeed = 0.01
# robotSpeed = 0.065
# the maximum rotational speed (radians / frame) of
# the robot compared to a full circle
robotRotationalSpeed = 0.05
# the diameter of the robots compared to the bounding box
robotDiameter = 0.06
#0.059
# the number of recent values to use when smoothing
# the position and rotation of the robot
```

```
historyLength = 3

[controls]
# key to reload the configuration at run time.
#Not all changes will take effect
loadConfig = l
```

Sorting algorithms

This section will explain some of the sorting algorithms that are used in some of the papers mentioned in chapter 2.1.

.1 Odd even sort

Odd even sort is a sorting algorithm which resembles bubble sort. The algorithm takes a list of elements, then compares all the odd placed element with the element next to it. That is; compare list[i] with list[i+1] where i is an odd number. Then the algorithm does the same for all i even number. This is done repeatedly until we get a sorted list.

.2 Bitonic sort

Bitonic sort is a highly parallel sorting algorithm, the idea is to have the elements in the list in a bitonic sequence. A bitonic sequence is a sequence where the list is increasing, then decreasing and then eventually increasing again. However the last increasing part are not allowed to increase past the first element in the list.

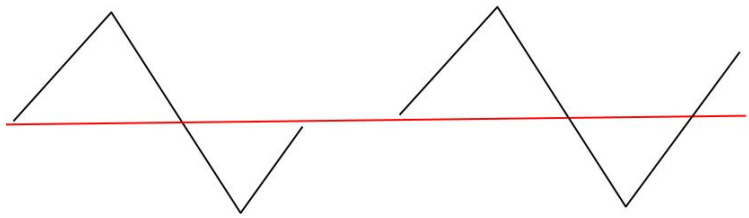


Figure 1: Left figure is a bitonic sequence, right figure is not a bitonic sequence because the last increasing part is higher than the start

After the algorithm have obtained a bitonic sequence, it will start to compare the elements with certain distance from each other and swap these elements if they are in the wrong order.

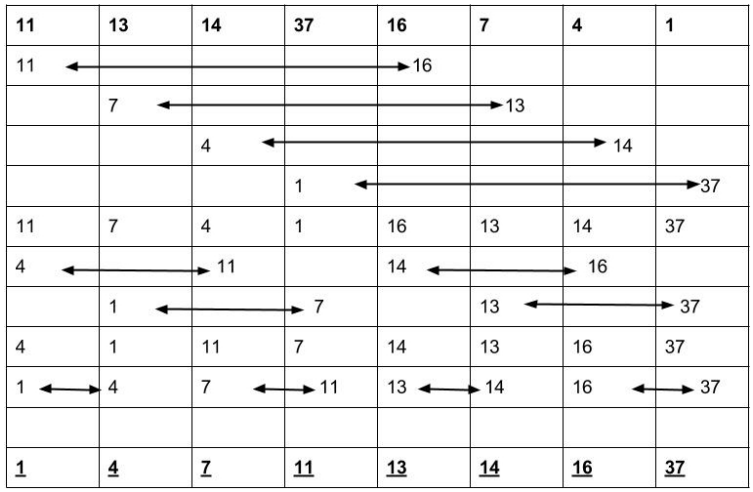


Figure 2: Example of a bitonic sort run

In the example we have the numbers 11,13,14,37,16,7,4,1. Which is initially in a bitonic sequence because the first half is increasing and the last half is decreasing. The algorithm now compares the first and the fifth element, as indicated by the arrows. All of these comparisons can be done in parallel. The algorithm compares the elements at different position, in the example it start with a comparison of elements that are distance 4 away from each other, that is element at position 1 and 5, 2 and 6 and so on. Then elements with distance 2 is compared, and then elements with distance 1 are compared. The runtime of the bitonic sort algorithm is $O(nlog(n)^2)$ but the idea is to run it on n Cuda threads, reducing the runtime to $O(log(n)^2)$ for each thread.

System dependencies

The system requires OpenCV and Boost to run the camera tracking software. An internet connection is also required for the camera tracking software to run because the information of each robot is sent via UDP to the IP address specified in the configuration file. The internet connection is required even if the UDP packets are sent to itself via loopback IP, without a valid internet connection, the camera tracking software will crash. Slick2D java game library is used to run the simulator and to run the watcher software. The watcher software also needs RXTXComm to be able to communicate with the robot using the serial port.

The robots' code runs on an Arduino micro and are therefore programmed using the Arduino language and C++. The ChIRP library files are required to communicate with the sensors and the motors. An Arduino Uno can be used to program the microcontrollers on the robot.

Video results

It is hard to tell if the robots are behaving like they are supposed to by only looking at graphs and interpreting them. Videos of a random run per scenario was created because of this. All the videos were filmed using a mobile phone. The QR-code can be used to get access to the links without the need to type them out. All of the QR-code were generated using http://goqr.me/. All of the videos were filmed from the other side of the room, and will therefore be mirrored and upside down compared to the images shown in 4.2.

.3 Scenario 1

https://www.youtube.com/watch?v=bM8l8bkL49I

.4 Scenario 2

https://www.youtube.com/watch?v=yAYImaH_spA

.5 Scenario 3

https://www.youtube.com/watch?v=AgzKlR0NMgI

This video is not three minutes long, because someone was calling the phone, therefore the video got cut short.