FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Drone Flock:Formation Control of micro aerial vehicles

Miguel Rodrigues

Mestrado em Engenharia Eletrotécnica e de Computadores

Supervisor: António Pedro Aguiar

October 4, 2023

Resumo

Controlo em formação de micro veículos aéreos tem sido um tópico cada vez mais discutido, parcialmente devido a avanços tecnológicos que diminuiram o preço dos robôs, enquanto a robustez deles aumentou. As principais aplicações destes robôs no contexto de *swarm robotics*(robótica de "enxames") inclui espetáculos aéreos, segurança e supervisão, limpezas, inspeção e transporte de objetos de grande porte, agricultura e muitos outros.

O fenómeno de agregação ou formação é inspirado na natureza. Durante muitos anos cientistas têm observado este fenómeno em cardumes de peixe que se movem em grupo para se protegerem, pássaros a voar numa certa configuração para minimizar a resistência do ar e até bactérias a mover-se de forma coordenada para atingir zonas ricas em nutriente.

"Swarm robotics", ou robótica de "enxame", é um método inspirado em comportamentos organizados de animais mencionados anteriormente. Ao observar grupos de seres vivos conclui-se que é possível completar certas tarefas que seriam impossíveis de realizar individualmente. Através de regras simples e interações locais, a robótica de enxame visa projetar comportamentos coletivos robustos, escaláveis e flexíveis para a coordenação de grandes quantidades de robôs.

No entanto, *swarm robotics* parte do pressuposto que o sistema é descentralizado, o que significa que não há uma entidade que controla todos os drones, mas, em vez disso, os mesmos tomam decisões sobre qual deve ser a próxima ação. Para além disso, em muitas aplicações, cada agente não tem necessariamente o estado completo do ambiente que o rodeia, em vez disso, é necessário que cada individuo seja capaz de executar bem o seu objetivo com informação incompleta do grupo ou enxame.

A primeira parte da dissertação recai nas noções matemáticas necessárias tal como lyapunov e as diferentes representações e definições em controlo de formação. De seguida, uma arquitetura especifica é definida e implementada para testar os problemas de movimento coletivo anteriormente referidos. Para além disso, é feito um estudo sobre os simuladores de robótica disponíveis e, posteriormente, um deles é selecionado para testar os algoritmos. Já na parte final, os resultados da simulação são apresentados e discutidos.

Abstract

Formation control in micro aerial vehicles (MAVs) is a subject with increasing interest, partially due to technology breakthroughs creating more robust and cheaper robots. Potential applications of these robots in the context of swarm robotics include aerial shows, surveillance, cleaning, inspection and transportation of large objects, farming and many others.

Swarming is inspired in nature. For many years scientist have observed schools of fish moving in an aggregated motion to protect themselves, flocks of birds flying in a certain configuration to minimize air resistance, or even bacteria moving in a coordinated motion to reach nutrient rich zones.

Swarm robotics is an approach to collective robotics that takes inspiration from the self-organized behaviors of social animals mentioned before. By observing living beings' swarms one can conclude that with cooperation between individuals it is possible to complete certain tasks that otherwise would be impossible to achieve. Through simple rules and local interactions, swarm robotics aims at designing robust, scalable, and flexible collective behaviors for the coordination of large numbers of robots.

However, swarm robotics assumes the system is decentralized, meaning there is no process controlling all the drones, but instead all the drones make decisions on what their next action should be, adding to that, in many applications each individual does not a complete state of the environment, instead, they need to perform well with incomplete information of the swarm.

The first part of this dissertation will focus on understanding the mathematical preliminaries needed such as lyapunov and the different types of representations and definitions in swarm robotics. Then a specific architecture is defined and implemented to recreate the motions previously mentioned. In addition, a study on the available robotics simulators is made and one chosen to test our algorithms. In the final part of this dissertation, the simulation results are presented and discussed.

Agradecimentos

I want to thank my family and friends for supporting me along this long journey of my life. I want to thank my parents in particular, for being always there for me, helping go through some of my worst phases. Some friends of mine were also essential and helped me a lot along the way.

I also want to thank some of my professors, especial regards to my supervisor, professor Pedro Aguiar, for being comprehensive and helpful during this work. I want to thank professors Paulo Costa, Vitor Pinto, Paulo Portugal, Mário Sousa and Nuno Cruz as well, for all their help and their input.

Miguel Rodrigues

"O objectivo embora empírico
É contar uma história
Despir-me nesse anfiteatro
Ser criado p'rá glória
Vender a alma numa calma memória
O objectivo é desistir, não resistir
Deixar-me ir
De volta aquela casa onde era fácil rir
Onde do riso vinha arte de fazer sorrir
E ser artista é ser infante
E era tão contagiante
Que hoje eu vou contra gigantes
Sem temer cair
Mas principalmente eu sou quem me viu partir
Vivo ofegante entre avalanches de os desiludir
Enquanto sonho o impossível
Eu sou o fácil e o incrível
Só suponho, já fui mais confiante e risonho
Mas sinto todo o peso quando me oponho
Não quero ser perfeito vejo quando componho mas minto
Eu aprendi sem jeito
Não quero é querer ser perfeito
Mano eu dei-te tudo o que eu tenho no peito
P'ra eu viver direito
P'ra eu viver direito"

Slow J

Contents

List of Figures

List of Tables

Abreviaturas e Símbolos

UAV	Unmanned Aerial Vehicle
MAV	Micro Aerial Vehicle
LED	Light Emitting Diode
TSP	Travelling salesperson problem
RGB	Red green blue
GAS	Globally asymptotically stable
MPC	Model predictive control
FHOCP	Finite Horizon Optimal Control Problem
PFSM	Probabilistic finite state machine
TDoa	Time Difference of Arrival
UWB	Ultra-Wideband
TWR	Two-way ranging
LPS	Local Positioning System
GUI	Graphical user interface
SDF	Simulation description format
IDE	Integrated Development Environment

Chapter 1

Introduction

1.1 Context

The fascination with the emergence of flocking, swarming, and schooling behaviors in groups of agents with local interactions has been a long-standing interest among scientists. Examples of such behavior can be observed in various natural systems, including birds, fish, bacteria, penguins, bees, ants, and more[7] [8] [9] [10]. Understanding how these collective behaviors emerge and are controlled has been a central focus in the field of swarm robotics.

In the past few decades, quadcopters have emerged as valuable tools for studying and implementing such control algorithms. These unmanned aerial vehicles, known for their simplicity in fabrication compared to other vehicles, provide an excellent platform for modeling, simulation, and experimentation[11]. The high degree of autonomy that quadcopters can achieve makes them ideal for exploring various tasks, including formation control, where multiple quadcopters can coordinate their movements to achieve specific spatial configurations or patterns.

Adding up, there has been an increased research effort and an worldwide interest in the use of autonomous robotic vehicles to execute tasks of increasing complexity. An attractive and challenging scenario where considerable effort is now being placed is the deployment of groups of heterogeneous networked vehicle systems (air, surface, and underwater) that can interact autonomously with the environment and other vehicles to perform, in the presence of uncertainty and adversity, tasks beyond the ability of individual vehicles. [9]

Although the scope of this thesis is an homogeneous networked system, since only aerial vehicles are used, this concept is of great interest, where a monolithic structure can be distributed in an inexpensive network of vehicles, resulting in an improvement in efficiency, performance, reconfigurability and robustness. [9]

Finally, a decrease in production costs of this types of robots contributed to a broader interest in swarm robotics. It is extremely easier to replace or repair a single individual of a flock in case of a malfunction than to do the same for a single complex and expensive robot designed to achieve the same goals as the swarm. [9]

1.2 Problem statement

The drone flock motion control problem involves controlling the movement of a group of drones, or unmanned aerial vehicles (*UAVs*), in a coordinated manner. This could mean keeping the drones at a certain distance from each other, maintaining a formation, or following a specific path. The goal of the motion control system is to ensure that the drones move in a safe and efficient manner while also achieving the desired motion behavior.

The goal of this dissertation is to implement and test different motion control algorithms of aggregation, formation, leader-follower, goal location and exploration using the existing literature.

In the first part of this exploration, our focus will be on the development of algorithms using a centralized approach. This approach involves a single controlling entity that manages and coordinates the trajectory and assigned tasks of each agent within the group.

In the second part it will be necessary to select a simulator capable of accurately representing the behaviour of a swarm. There are many options on the market but they require a deep understanding of how the simulator works and how to operate it. Simulating the robot may not be as straightforward as desired. Moreover, it is possible that the simulation may be too simple, not allowing to understand what is happening in the simulation, making it difficult to track each agent behaviour.

In the final part of this thesis, we will focus on testing the algorithms developed and discussing its results compared to the literature and taking into consideration the simulator chosen.

1.3 Motivation

Some tasks can not be performed by a single entity. This is proven to us by the surrounding nature. Since the beginning of times different species made a collective effort in order to achieve some kind of goal. It can be seen in various human situations,ie: the construction of a building, where it would be easier to do it with a large number of people rather than by yourself.Examples can also be found in wildlife such as birds flying in V configuration to minimize the effort of flying, ants working together to carry heavy loads and fish swarms where fish aggregate in a tiny space to confuse predators who would normally attack them if they were isolated.

To achieve a specific goal in robotics like tracking or lifting some structure, or other kind of objective, formation control is often appealing as stated by Cai et al.

«Formation control has many advantages in detection, localization and perception, which may contribute to efficiency task allocation, aerial refueling, self-organizing and reconfiguration for multiple UAVs» ([Cai et al., 2020, p. 1037]([12]).

In the same way *MAVs* can work together in order to complete a certain task that would be impossible to complete alone. Moreover, using smaller cheaper robots improves the robustness of the system, as a malfunction with a single robot should not impact the overall effectiveness of the entire swarm to perform a given task. Adding to this, using cheaper robots makes it easier to replace or repair them, since cheaper robots are usually less complicated.

However, using cheaper robots results in limitations regarding the computational capabilities, energy storage, sensing and hardware performance for each robot.

1.4 Applications

1.4.1 Aerial shows

Micro aerial vehicles have often been employed for entertainment purposes, utilizing *LED* technology to craft captivating 3D images. An illustrative example of this was witnessed during the 2020 Olympics in Tokyo when a swarm of drones expertly positioned themselves to create the iconic image of a globe.[13]. Intel has also explored drone control for entertainment in 2016 when they made 100 drones equipped with *LEDs* fly simultaneously.[14] In Saudi Arabia, Riyadh Season's opening ceremony saw over 2500 crewless aircraft vehicles employed in a display taken charge by the Geoscan Drone Show. They drew remarkable paintings atop Riyadh, the Saudi capital. [15] There are many other examples of drone shows all around the globe as Intelligent control companies try to display their finding over the years and create increasingly complex shapes and formations. Another more recent example was at Spooky Moon Music Festival in El Paso where 200 drones were used to sculpt recognizable shapes like objects or logos in the sky as shown in figure 1.1. [1]

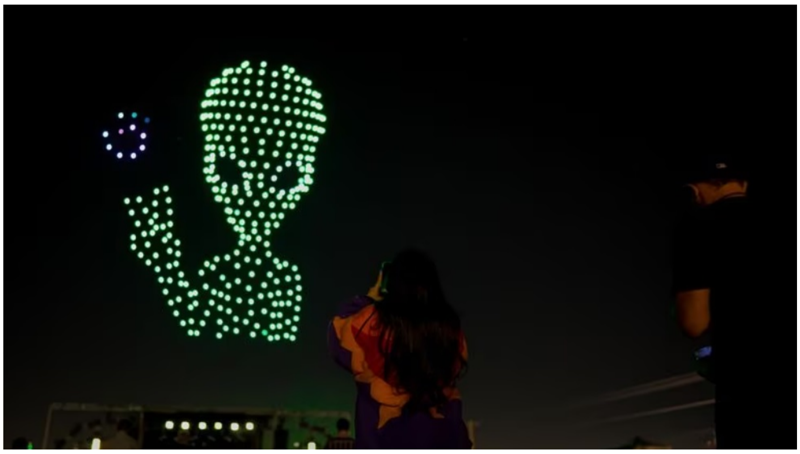


Figure 1.1: Drone Show at Spooky Festival in El Paso. Source:[1]

1.4.2 Security and Surveillance

The area of Security and Surveillance is also evolving towards architectures with incorporated cameras and vision modules to detect different actors in the environment and warn the network's supervisor.

Swarms of drones could also perform convoy protection and path planning for ground vehicles, or even for attacking purposes as seen in the Russia-Ukraine war. There are already many examples of countries developing swarms for military purposes including the previously mentioned, but also countries like the United States and India are pursuing swarm technology. [16] [17] This robots can also be used for environmental surveillance, using sensors to perform air quality measurements.

In [18] the author presents a multiple drone application for rescue missions in flood events. It follows 4 steps:

1. Identify the geographical areas the drone needs to cover
2. Identify the places the drone needs to fly, ie: buildings, streets, etc.
3. Once the places are identified the problem is posed as a Travelling Salesperson problem (*TSP*).
4. Solve the *TSP* using Firefly algorithm.

Travelling Salesperson problem(*TSP*) - tries to minimize the total distance traveled between each pair of given cities. The problem is often represented as a graph, where the cities are nodes, and the distances between them are represented as weighted edges.

Firefly algorithm - Firstly introduced by Xin-She in 2008, it is a metaheuristic algorithm used to solve optimization problems. In the context of optimization, each firefly represents a potential solution to the problem, and the brightness or intensity of the firefly's light represents the quality of the solution. The algorithm aims to move the fireflies in such a way that they converge toward the optimal solution while avoiding getting stuck in local optima.

Drone applications could also be useful in fire situations to help quickly locating any need for help. Furthermore, the usage of inexpensive disposable drones to combat these situations, instead of human force, is a relevant advantage. In [19], the author a self-organization algorithm for swarms coupled with a collective behaviour based in a particle swarm algorithm.

Another interesting application is indoor space navigation, in [20], the author focuses on a decentralized architecture for multiple pocket drones able to navigate through an indoor unknown environment and then return to their initial positions.

It first dives into low-level-navigation capabilities of each individual, namely being able to detect objects and estimate its own relative velocity through a stereo-camera. Then it addresses the necessity for swarm operations of multiple pocket drones to avoid each other.

In the second part, the author investigates alternative methods to conventional navigation strategies like bug algorithms, which does not require a map and minimizes memory usage.

Ultimately, a bug-algorithm-based navigation strategy for pocket drones exploration and homing was developed and tested using 6 pocket drones, being able to achieve its goal for the first time by a swarm with drones of that size.

Drone flocks could also play an important role in environmental monitoring, in [21], the author proposes a heterogeneous architecture composed by wheeled vehicles and *UAVs* designed to map the evolution of pollution using a set of sensors.

1.4.3 Farming

Farming applications have also increased in interest. Farming is a physically demanding activity, with high monitoring necessity. Using drones for farming, or "precision agriculture," can provide many benefits compared to traditional methods of crop cultivation and pest management. For example, drones can quickly and efficiently cover large areas of land, collect high-resolution imagery of crops, and apply pesticides or fertilizers with precise targeting. Overall, using a drone flock for farming can provide a cost-effective and efficient way to perform tasks such as crop mapping, weed detection, and pest control, while also reducing the need for human labor and improving the safety of these tasks.

In [22], a robotic animal herding is proposed based on a network of autonomous barking drones. The goal of this implementation is to replace the usual herding methods like dogs for a drone based control algorithm able to quickly collect sheep from a sparse status and then guide them to a designated location.

In [23], a novel approach to distinguish between different field's plowing techniques by means of am *RGB-D* sensor. Its' system is compatible with many commercial Unmanned Aerial Vehicles. This implementation relies mostly on a computer vision algorithm which provides good classification results of field's plowing depths.

1.4.4 Object Transportation

As online delivery companies like Amazon or AliExpress become a daily tool for most people, the interest in automating deliveries increases. Amazon and Windcopter are examples of companies who already made drone deliveries [24] [25]. Most recently, Walmart joined Wing, an on-demand delivery provider, to offer drone delivery of orders from two Dallas-area stores[26]. One of the most explored and challenging tasks in object transportation is organizing a group of drones in a configuration capable of moving an object too heavy to be carried by a single drone, and doing it in the most cost efficient way possible. In [27], the author studies last-mile delivery problems and presents a system where drones and trucks work in parallel in order to tranport heavier objects.

1.5 Dissertation Structure

Besides the introduction, this dissertation contains more 3 chapters. In chapter 2, the state of the art, fundamentals and important tool for the project are discussed. In chapter 3, the motion control problems' logic is implemented. In chapter 4, the algorithms are simulated and the results discussed. In chapter 5, the main conclusions from this dissertation are presented.

Chapter 2

State of the Art, Fundamentals and Tools

2.1 Introduction

In this chapter it will be revised the state of the art of drone flock motion control as the the main background fundamentals to conduct this project are discussed. Additionally, the tools necessary in this project are also referenced in this chapter.

2.2 Lyapunov stability theorem

The Lyapunov stability theorem is a fundamental result in the study of dynamical systems that states that if a system has a Lyapunov function, then it is asymptotically stable. More formally:

Let $\dot{x} = f(x)$, the equilibrium point x = 0 is:

- stable if, for each $\varepsilon > 0$, there is $\delta = \varepsilon(\delta) > 0$ such that

$$\|x(0)\| < \delta \Rightarrow \|x(0)\| < \varepsilon, \forall x(t) \geq 0 \tag{2.1}$$

- asymptotically stable if it is stable and δ can be chosen such that

$$\|x(0)\| < \delta \Rightarrow \lim_{x \to \infty} x(t) = 0 \tag{2.2}$$

The Lyapunov stability theorem can also be used to determine if a system is globally asymptotically stable (*GAS*).

Let x = 0 be an equilibrium point for $\dot{x}$ = f(x). Let V : $\Re^n \Rightarrow \Re$ be a continuously differentiable function such that:

$$V(0) = 0, V(x) > 0, \forall x \in D \backslash \{0\} \tag{2.3}$$

and

$$\dot{V}(x) \leq 0, \forall x \in D \tag{2.4}$$

then $x = 0$ is stable.

Moreover, if

$$\dot{V}(x) < 0, \forall x \in D \backslash \{0\} \tag{2.5}$$

then $x = 0$ is asymptotically stable.

2.3 Swarm Motion control

Swarm motion control has remained a critical and engaging topic of study within the field of robotics and computational science. The interest stems from the incredible feat of nature where large numbers of individuals coordinate their movement based on local interactions, producing efficient collective behaviors.

«"...and the thousands off fishes moved as a huge beast, piercing the water. They appeared united, inexorably bound to a common fate. How comes this unity?"» –Anonymous, 17th century (from Shaw)» ([8])

Humans have always been amazed by the great coordination of some species to achieve some kind of goal.

«The basic urge to join a flock seems to be the result of evolutionary pressure from several factors: protection from predators, statistically improving survival of the (shared) gene pool from attacks from predators, profiting from a larger effective search pattern in the quest for food, and advantages for social and mating activities» ([Reynolds, p. 7]([8])

As Reynolds states, intimidating predators, improving survival from attacks of predators or by increasing the chances of locating food are some of the reasons to why certain species developed such amazing coordinated motions.

Several significant models have emerged in the literature which have profoundly influenced the development of unmanned aerial vehicle (*UAV*) networks. These models have provided a solid foundation for understanding and replicating swarm behaviors.

1. Reynolds' Model (1987) - In 1987 Reynolds published the first work on coordinated motion, which aims to develop a distributed behavioral model able to respect the dynamics and interactions of a biological aggregates, such as flocks, herds and schools. He used three simple rules to simulate the movement of flocks: separation, alignment, and cohesion. In this model, each agent moves to avoid crowding local flockmates (separation), towards the average heading of local flockmates (alignment), and towards the average position of local flockmates (cohesion). The Boids model is a local behavior model where every individual only interacts with its close neighbors and does not have a global view of the swarm.

2. Veysel Gazi's Attraction/Repulsion Model (2002) [28] - all agents know the exact position of all the other agents in the system. They implement a cost function based on attraction/repulsion forces where when in large distances from each other aggregation dominates and at short distances repulsion dominates. This mechanism leads the swarm to an equilibrium state where these are balanced.

3. Tamás Vicsek et al. proposed a simple but powerful model in 1995[29]. In Vicsek's model, each agent adjusts its velocity to match the average velocity of its neighbors within a certain radius. This model was unique in that it showed a phase transition from disorder to order as density or speed increased. Despite its simplicity, the model was able to reproduce complex behaviors and phase transitions inherent to real swarms, providing a novel insight into the emergence of order in swarm systems.

In [2], a review on swarm engineering literature, two taxonomies are proposed: methods and collective behaviours. In the first taxonomy, works are classified taking into consideration the design and analysis methods used. In the second taxonomy, works are classified by the type of collective behaviour they implement. The following image shows the two different taxonomies and their classification principles.

This article focuses on swarm robotics, with the idea that robots are autonomous, can act to modify the environment they are in, their sensing and communications capabilities are local and do not have access to centralized control applications or global knowledge. Despite the scope of this work being a centralized application to control the motion and formation of a swarm, [2] presents different techniques to tackle the problem of motion control in swarms that could be useful for this dissertation.

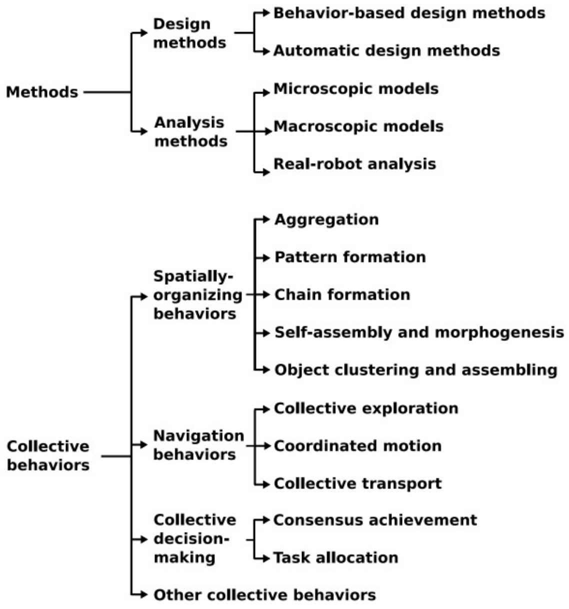


Figure 2.1: Two taxonomies proposed Source:[2]

In Figure 2.1, two taxonomies are presented, illustrating their classifications. This study will primarily concentrate on examining collective behaviors, with a particular emphasis on spatially-organizing behaviors. These include phenomena such as aggregation, pattern formation, goal localization, leader-follower dynamics, and other pertinent challenges related to motion control. However, this paper focuses on decentralized applications, which are not the main scope. Nonetheless, it is important to have a basic understanding of the methods used since a decentralized approach is always more realistic and, in some ways, less computationally heavy for the drones.

«Yet all evidence indicates that flock motion must be merely the aggregate result of the actions of individual animals, each acting solely on the basis of its own local perception of the world.» ([Reynolds, p. 1])([8])

Although the scope of this dissertation is a centralized system with all the data needed to perform its goal, it is important to notice that such is not necessarily the case in the real world. In a school of fish, a single individual does not know the exact velocity, position or orientation of all the individuals.

Nonetheless, one has to consider that in a boid (simulated flock), it is possible to choose the amount of information each individual has access to. This is one of the most important aspects when developing a robust swarm. The dichotomy between each individual having the complete state of the group at every given moment, and the computational effort and speed to process all the information at any given time.

Adding to that, it is intuitive that for a single agent, the nearest individuals' state is much more critical to its dynamics than of the individual the furthest away from it.

So when designing a distributed model it is necessary to take into account if it is not computationally too expensive for each individual to have the complete state of the swarm at any given moment, in the case of not having a complete state of the world, it is important to determine how will the weight of each individuals' information be classified and the number of agents' states each individual will "subscribe" to.

The problem of computational load, communication and battery consumption are well documented and there are many ways to tackle this issue.

In [12], a *MPC* (Model Predictive Control) model is used with an event-triggered mechanism to reduce computational load and communication consumption in formation control. Specifically, the author develops a «distributed *MPC* scheme for *UAV* formation control with event-triggered mechanism, which can enable the Finite Horizon Optimal Control Problem (*FHOCP*) to be solved asynchronously and reduce the computational burden» ([Cai et al., 2020, p. 1038]([12])).

This is also stated by [30], where he presents «a novel event-triggered globalized dual heuristic programming method» ([Yi et al., 2019, p. 1]([30]).

In spite of the advantages of decentralized architectures with specialized mechanisms to tackle energy consumption and computational load, for simple drones and scenarios involving a small number of individuals, a centralized approach often suffices. Therefore, our initial focus will be on exploring and implementing centralized control strategies, mainly the work presented in [28].

Veysel Gazi's attraction and repulsion model allows an implementation of a centralized or decentralized architecture. The only requirement is knowing the position of the agents at all times. They implement two models:

1. *Single Integrator Model* - the control input directly influences the velocity of the agents in the system.

2. *Double Integrator Model* - the control input impacts the acceleration of the agents.

Although the Double Integrator Model is more realistic for physical systems, as changes in velocity usually occur over time and are not instantaneous, the single Integrator Model is simpler to implement and is often accurate enough to meet the requirements of the system.

2.3.1 Single Integrator Model

The following concepts are based on [28]:

The easiest way for mathematical representation of agents in the scope of swarm behavior is the *single integrator model*. In this model the motion of an agent i, i = 1, ..., N is given by:

$$\dot{x}_i = u_i \tag{2.6}$$

where $x_i \in \Re^n$ is the state of the agent i and $u_i \in \Re^n$ is the control input. This model is referred to as higher-level or kinematic agent model since it ignores the low level dynamics of each agent. It is assumed that all individuals in the swarm move simultaneously and know the exact relative position of the other individuals. Let $x^T = [x_1^T, x_2^T, ..., x_N^T] \in \Re^{Nn}$ n denote the vector of concatenated states of all the agents. In this section the control input u_i will have the form:

$$u_i = -\nabla_{x_i} J(x) \tag{2.7}$$

where $J : \Re^{Nn} \rightarrow \Re^n$ is the potential function which represents the interaction between the agents and needs to be chosen by the designer of the system based on the requirements and limitations. The author developed multiple potential functions to model various motion control problems, including aggregation, social foraging, and formation control, as elaborated upon in the subsequent chapters.

2.3.2 Aggregation

The goal of aggregation is to group all the individuals in a region of the environment, without violating a minimum distance usually to avoid collisions. Despite being a relatively simple algorithm, it is often used to exchange information among the drones.

Aggregation is a common behavior in nature, it can be observed in bacteria, cockroaches, bees, fish, penguins and many other living beings.

In swarm robotics, aggregation is usually approached in two ways: probabilistic finite state machines (*PFSMs*) or artificial evolution. The most common approach is based on *PFSMs*: the robots explore an environment and, when they find other robots, they decide stochastically whether to join or leave the aggregate.

However, swarm robotics is defined to be a decentralized system, where the individuals make autonomous decisions based on the environment and the information provided by the other individuals. In this thesis a centralized architecture will be adopted first, to simplify the development of the motion control algorithms. Being so, the aggregation problem decreases in difficulty since the position of each individual is controlled by a single program.

2.3.2.1 Potential Function Design

Since our potential cost function depends on the attraction and repulsion forces, it could be denoted as:

$$J_{aggregation}(x) = \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} [J_a(\|x_i - x_j\|) - J_r(\|x_i - x_j\|)] \tag{2.8}$$

Here, $J_a : \Re^+ \rightarrow \Re$ represents the attraction component whereas $J_r : \Re^+ \rightarrow \Re$ represents the repulsion component of the potential function.

Given the above potential function, the control input can be calculated as:

$$u_i = - \sum_{j=1, j\neq i}^{N} [\nabla_{x_i} J_a(\|x_i - x_j\|) - \nabla_{x_i} J_r(\|x_i - x_j\|)] \tag{2.9}$$

Note that since $\dot{x}_i = u_i$, the motion of the individual is along the negative gradient and leads to a decent motion towards a minimum of the potential function. Moreover, since the functions $J_a(\|y\|)$ and $J_a(\|y\|)$ create a potential field around each individual, the above property restricts the motion of the individuals toward each other along the combined gradient of these potentials. One can show that, because of the chain rule and the definition of the functions J_a and J_r, the equalities

$$\begin{aligned} \nabla_y J_a(\|y\|) &= y g_a(\|y\|) \\ \nabla_y J_a(\|y\|) &= y g_a(\|y\|) \end{aligned} \tag{2.10}$$

are always satisfied for some functions $g_a : \Re^+ \rightarrow \Re$ and $g_r : \Re^+ \rightarrow \Re$. Here, g_a and g_r represent, in a sense of magnitude, the attraction and repulsion terms, respectively. Note that the combined term $y g_a(\|y\|)$ and $y g_a(\|y\|)$ represent the actual attraction and repulsion. The vector y guarantees that the interaction vector is along the line on which y is located as well as affects the magnitude of the attraction and repulsion components. The terms $g_a(\|y\|)$ and $g_r(\|y\|)$ only affect the magnitude of the attraction and repulsion terms.

Let us define function $g(\cdot)$:

$$g_y = -y\,[g_a(\|y\|) - g_r(\|y\|)] \tag{2.11}$$

Function $g(\cdot)$ is an attraction/repulsion function and it is assumed that on large distances attraction dominates, on short distances repulsion dominates and at a unique distance on which both this forces balance. In other words, it is assumed that function $g(\cdot)$ satisfies the following assumptions.

Assumption 1 *The function $g(\cdot)$ in equation 2.11 and the corresponding $g_a(\cdot)$ and $g_r(\cdot)$ are such that there is a unique distance δ at which $g_a(\delta) = g_r(\delta)$. Moreover, for $\|y\| > \delta$, $g_a(\delta) > g_r(\delta)$ and for $\|y\| < \delta$, $g_a(\delta) < g_r(\delta)$.*

Note that the above functions are odd, this is an important feature that leads to reciprocity in the inter-agent relations and interactions.

To satisfy the above assumptions, the designer should choose the attraction and repulsion potentials such that the minimum of $J_a(\|x_i - x_j\|$ occurs on or around $\|x_i - x_j\| = 0$ whereas the maximum of $J_r(\|x_i - x_j\|$ occurs on or around $\|x_i - x_j\| \to \infty$. Furthermore, the minimum of the combined term $J_a(\|x_i - x_j\| - J_r(\|x_i - x_j\|$ occurs at $\|x_i - x_j\| = \delta$

One possible potential function which satisfies the above conditions presented in [28] is:

$$J(x) = \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} \left[\frac{a}{2}\|x_i - x_j\| + \frac{bc}{2}\exp\left(-\frac{\|x_i - x_j\|^2}{c}\right)\right] \tag{2.12}$$

Its corresponding attraction/repulsion function can be calculated as:

$$g(y) = -y\left[a - b\exp\left(-\frac{\|y\|^2}{c}\right)\right] \tag{2.13}$$

The equilibrium distance for this function can be easily calculated as $\delta = \sqrt{cln(b/a)}$ The above potential function based interaction model captures this in its simplest form by having separate equations of motion for each individual that do not depend on an external variable. In contrast, an individual's motion depends only on the position of the individual itself and its observation of the positions (or relative positions) of other individuals.

In addition to this potential function, Gazi proves that the centroid of the swarm is stationary for all t, meaning the velocity of the centroid is equal to zero. Furthermore, as t tends to infinity, the velocity of the individuals will tend to zero and will eventually reach zero when each individual's distance to the centroid is equal to the equilibrium distance.

Afterwards, a swarm cohesion analysis is made where different attraction/repulsions functions for the aggregation method are presented for different bounds in the attration and repulsion terms:

1. Linear Attraction and Bounded Repulsion Case
2. Linearly Bounded from Below Attraction and Unbounded Repulsion
3. Almost Constant Attraction and Unbounded Repulsion
4. Considering Agents with Finite Body Size

2.3.2.2 Linear Attraction and Bounded Repulsion Case

In this section the special case in which the attraction part satisfies

$$g_a(\|y\|) = a \tag{2.14}$$

for some finite positive value $a > 0$ and for all y, whereas the repulsion is constant or bounded as:

$$g_r(\|x_i - x_j\|)\|x_i - x_j\| \leq b \tag{2.15}$$

for some finite constant value $b > 0$. Function 2.13 satisfies both of these conditions.

Gazi then uses Lyapunov stability theorem to prove that a swarm consisting of agents with dynamics respecting equation 2.69 and with control input in 2.9 with an attration/repulsion function g(.) which is odd, satisfies Assumption 1, and has linear attraction and bounded repulsion, satisfying equations 2.14 and 2.15. Furthermore, the author proves that as time progresses, all the members of the swarm will converge into an hyperball:

$$B_\varepsilon(\bar{x}) = \{x : \|x - \bar{x}\| \leq \varepsilon\} \tag{2.16}$$

where

$$\varepsilon = \frac{b}{a} \tag{2.17}$$

Using Lyapunov it is proven that the error $\dot{e}_i = \dot{x}_i - \dot{\bar{x}}$ is bounded by:

$$\|e_i\| \leq \frac{b(N-1)}{aN} < \frac{b}{a} \overset{\triangle}{=} \varepsilon \tag{2.18}$$

There is also a bound on the time it will take to converge:

$$\bar{t} = max_{i \in \{i,\ldots,N\}} \left\{ -\frac{1}{2a} ln \left(\frac{\varepsilon^2}{2V_i(0)} \right) \right\} \tag{2.19}$$

«Having the attraction dominate at large distances prevents the swarm from dispersing, whereas having the repulsion dominate on short distances prevents it from collapsing to a single point, and the equilibrium is established in between.» ([Gazi e Passino, 2011, p. 34]([28])

So it is proven that with the attraction and repulsion function chosen, it is possible to establish a bound on the swarm size.

2.3.2.3 Linearly Bounded from Below Attraction and Unbounded Repulsion

In this section, the attraction function is given by:

$$g_a(\|x_i - x_j\|) \geq a \tag{2.20}$$

for a positive finite constant $a > 0$ and for all $\|x_i - x_j\|$ and the repulsion function is:

$$g_r(\|x_i - x_j\|) \leq \frac{b}{\|x_i - x_j\|^2} \tag{2.21}$$

A simple function satisfying the above equations is:

$$g(x_i - x_j) = \left[a - \frac{b}{\|x_i - x_j\|^2}\right](x_i - x_j) \tag{2.22}$$

which corresponds to the potential function

$$J(x) = -\sum_{i=1}^{N-1} \sum_{j=i+1}^{N} \left[\frac{a}{2}\|x_i - x_j\| - b \ln \|x_i - x_j\|\right] \tag{2.23}$$

As in the previous section, Gazi demonstrates there is a bound on the swarm using Lyapunov proving that the mean squared root of the distances of the swarm members to the centroid will satisfy:

$$e_{rms} \triangleq \sqrt{\frac{1}{N}\sum_{i=1}^{N} \|e_i\|^2} \leq \sqrt{\frac{b}{2a}} \triangleq \varepsilon_{rms} \tag{2.24}$$

So it is also proven that the error e_i will be bounded by:

$$\|e_i\| \leq \sqrt{\frac{bN}{2a}} = \varepsilon_{rms}\sqrt{N} \tag{2.25}$$

Note that the Lyapunov function mentioned chosen $\dot{V} = \dot{e}_i^T e_i$ is just one way to quantify the cohesion/dispersion of the swarm. One could also try to quantify the inter-agent distances instead of the distances to the centroid.

2.3.2.4 Almost Constant Attraction and Unbounded Repulsion

In this section the author discusses the attraction functions that satisfy $g_a(\|x_i - x_j\|) \to 0$ as $\|x_i - x_j\| \to \infty$. However it is assumed that

$$g_a(\|x_i - x_j\|) \geq \frac{a}{\|x_i - x_j\|} \tag{2.26}$$

The repulsion function is the same as in the previous section, resulting in the following attraction/repulsion function:

$$g(x_i - x_j) = \left[\frac{a}{\|x_i - x_j\|} - \frac{b}{\|x_i - x_j\|^2}\right](x_i - x_j) \tag{2.27}$$

The resulting potential function is

$$J(x) = -\sum_{i=1}^{N-1} \sum_{j=i+1}^{N} \left[a\|x_i - x_j\| - b \ln\Big(\|x_i - x_j\|\Big)\right] \tag{2.28}$$

The author then proves using the inequalities

$$a\sum_{i=1}^{N}\sum_{j=1,j\neq i}^{N}\|x_i-x_j\| \leq \sum_{i=1}^{N}\sum_{j=1,j\neq i}^{N} g_r(\|x_i-x_j\|)\|x_i-x_j\|^2 \tag{2.29}$$

and

$$\|e_i\| = \frac{1}{N}\left\|\sum_{i=1}^{N}(x_i-x_j)\right\| \leq \frac{1}{N}\sum_{i=1}^{N}\|x_i-x_j\| \tag{2.30}$$

that the ultimate bound on e_i will be given by

$$\|e_i\| \leq \frac{Nb}{a} \tag{2.31}$$

Gazi concludes this section emphasizing that the bounds found so far are independent of the number of individuals N. He also states that although the above functions guarantee that the agents will not occupy the same spaces, it does not necessarily guarantee uniform swarm density.

2.3.2.5 Agents with Finite Body Size

Until now, the book considers the individuals of the swarm to be a single point. In this section, the agents will be viewed as entities with finite body size instead of points without dimensions. For simplicity purposes, the agents are represented as hyper-spheres in the n-dimensional space.

Assuming all agents are of the same size with radius η. Let x_i be the center of the hyper-sphere of agent i and x_j the center of the hyper-sphere of agent j. Then, to avoid collisions, $\|x_i-x_j\| \geq 2\eta^2$ needs to be satisfied.

In the previous sections, particularly in the unbounded repulsion case, the repulsion function satisfies

$$\lim_{\|x_i-x_j\|\to 0^+} g_r(\|x_i-x_j\|)\|x_i-x_j\| = \infty \tag{2.32}$$

So in order to avoid collisions between rigid bodies, one has to modify the repulsion function to be of "hard-limiting" type satisfying

$$\lim_{\|x_i-x_j\|\to 2\eta} g_r(\|x_i-x_j\|)\|x_i-x_j\| = \infty \tag{2.33}$$

One repulsion function which satisfies the above condition is given by

$$g_r(\|x_i-x_j\|) = \left[\frac{b}{(\|x_i-x_j\|^2-4\eta^2)^2}\right] \tag{2.34}$$

And the corresponding repulsive potential will be

$$J_r(\|x_i-x_j\|) = -\frac{b}{2(\|x_i-x_j\|^2-4\eta^2)} \tag{2.35}$$

Note that this implementation is based on the fact that initially all the agents are sufficiently far apart from each other: $\|x_i(0)-x_j(0)\| \geq 2\eta$ for all $(i,j), j\neq i$.

Morover, it is important to notice that now there is a maximum density, or minimum radius, in the swarm that guarantees there are no collisions. The author proves that for a n-dimensional space, the minimum radius of the **swarm** will be:

$$r_{min} = \eta \sqrt[n]{N} \tag{2.36}$$

It is also proven that there is an upper bound on the swarm density given by:

$$\rho \leq \frac{\beta(n)}{\eta^n} \tag{2.37}$$

where $\beta(n)$ is a constant for a given n, basically meaning the density only depends on the dimension of n of the state space.

«Having relatively uniform swarm density is an important characteristic of real biological swarms and therefore, a desired characteristic of mathematical models of swarms.» ([Gazi e Passino, 2011, p. 40]).[28]

It is possible to implement the rigid body case for all the three types of aggregation previously mentioned and the new potential functions will look like these:

Linear Attraction and Bounded Repulsion Case

$$J(x) = \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} \left[\frac{a}{2} \|x_i - x_j\| + \frac{bc}{2} \exp\left(- \frac{\left(\|x_i - x_j\|^2 - 4\eta^2 \right)}{c} \right) \right] \tag{2.38}$$

Linearly Bounded from Below Attraction and Unbounded Repulsion

$$J(x) = -\sum_{i=1}^{N-1} \sum_{j=i+1}^{N} \left[\frac{a}{2} \|x_i - x_j\| - 2b \ln \left(\|x_i - x_j\|^2 - 4\eta^2 \right) \right] \tag{2.39}$$

Linearly Bounded from Below Attraction and Unbounded Repulsion

$$J(x) = -\sum_{i=1}^{N-1} \sum_{j=i+1}^{N} \left[a \|x_i - x_j\| - 2b \ln \left(\|x_i - x_j\|^2 - 4\eta^2 \right) \right] \tag{2.40}$$

The author ends the chapter with some simulation examples which will be used as a guideline to guarantee the correct implementations of the aggregation methods discussed above.

2.3.3 Social Foraging

Social foraging occurs when agents must make decisions informed by the surrounding environment.

«It is a potential function/profile representing the resources in the environment which can be nutrients or some attractant or repellent substances (e.g., food/nutrients, pheromones laid by other individual, or toxic chemicals in biological swarms or goals, targets, obstacles or threads in engineering swarm applications).» ([Gazi e Passino, 2011, p. 43]([28]).

To represent that additional factor, the book introduces a new potential $\sigma : \Re^n \to \Re$, so that for a point $y \in \Re^n$, $\sigma(y) < 0$ represents attractive nature or nutrient rich, $\sigma(y) = 0$ represents neutral and $\sigma(y) > 0$ represents a noxious environment at y. In this context, the resource profile $\sigma(\dot{)}$ models the environment, obstacles are to be avoided and targets or goals to be moved towards.

The new potential function will now be composed by the attraction/repulsion term previously discussed plus the new resource profile potential.

$$J_{foraging}(x) = \sum_{i=1}^{N} \sigma(x_i) + \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} [J_a(\|x_i - x_j\|) - J_r(\|x_i - x_j\|)] \tag{2.41}$$

and the new control function will be

$$u_i = -\nabla_{x_i}\sigma(x_i) - \sum_{j=1, j\neq i}^{N} [\nabla_{x_i} J_a(\|x_i - x_j\|) - \nabla_{x_i} J_r(\|x_i - x_j\|)] \tag{2.42}$$

Here, it is assumed that the attraction/repulsion function satisfy the conditions stated in the previous section. Namely that function $g(\cdot)$ is odd and the potential satisfies Assumption 1.

Considering first the motion of the centroid one can obtain:

$$\begin{aligned} \dot{\bar{x}} &= -\tfrac{1}{N}\left(\textstyle\sum_{i=1}^{N} \nabla_{x_i}\sigma(x_i) + \sum_{i=1}^{N}\sum_{j\neq i}^{N}\left[g_a(\|x_i - x_j\|) - g_r(\|x_i - x_j\|)\right]\right)(x_i - x_j) \\ &= -\tfrac{1}{N}\textstyle\sum_{i=1}^{N} \nabla_{x_i}\sigma(x_i) \end{aligned} \tag{2.43}$$

This is because function $g(\cdot)$ is odd and $g(x_i - x_j) = -g(x_i - x_j)$ for all pairs (i, j). One can see that the collective behaviour described above has a filtering or smoothing) effect, preventing individuals from getting stuck at a local minima, since they are moving together and other individuals can "pull" them out of such local minima.

For the foraging swarm, the author only considers attraction/repulsion functions with linear attraction as in equation 2.14 and bounded repulsion as in equation 2.15.

Gazi then uses a Lyapunov function to prove that for the error to be bounded, one of the following assumptions need to be made:

Assumption 2 *There is a constant* $\overline{\sigma} > 0$ *such that:*

$$\|\nabla_y \sigma(y)\| \leq \overline{\sigma} \tag{2.44}$$

for all y.

Assumption 3 *There is a constant $A_\sigma > -aN$ such that*

$$\left[\nabla_{x_i}\sigma(x_i) - \frac{1}{N}\sum_{i=1}^{N}\nabla_{x_j}\sigma(x_j)\right] e_i \geq A_\sigma \|e_i\|^2 \tag{2.45}$$

for all x_i and x_j.

Note that Assumption 2 only requires the resource profile function to be bounded, however Assumption 3 requires the gradient of the profile at x_i to have a "large enough" component along e_i so that the effect of the profile does not prevent swarm cohesion. With this in mind, the author presents the following result:

Theorem 1 *Considering a swarm consisting of agents with state equation in 2.69 and with control input function in 2.11, which is odd, satisfies Assumption 1 and has a linear attraction and bounded repulsion. Then, as $t \to \infty$ we have $x_i(t) \to B_\varepsilon(\bar{x}(t))$, where*

$$B_\varepsilon(\bar{x}(t)) = \{y(t) : \|y(t) - \bar{x}(t)\| \leq \varepsilon\} \tag{2.46}$$

and

1. *If Assumption 2 is satisfied, then*

$$\varepsilon = \varepsilon_1 = \frac{N-1}{aN}\left[b + \frac{2\bar{\sigma}}{N}\right] \tag{2.47}$$

2. *If Assumption 3 is satisfied, then*

$$\varepsilon = \varepsilon_2 = \frac{b(N-1)}{aN + A_\sigma} \tag{2.48}$$

Gazi then proves the veracity of these Assumptions and gets to the conclusion that as long as $\|e_i\| > \varepsilon_2$, $\dot{V}_i < 0$. In other words, convergence is guaranteed.

The problem with social foraging is that now there can be conflicting desires in the swarm, the desire to search for food represented by the resource profile potential and the desire to stick together. For this reason the values chosen for the parameter of the attraction/repulsion function and the resource profile function used are intrinsically related to the swarm application purpose and environment where it is placed.

«These concepts are found in foraging theory in biology and model the balance of the desire of the individuals to "stick together" with the desire to "get more food" that was created by evolutionary forces.» ([Gazi e Passino, 2011, p. 47]([28]).

The threshold $A_\sigma = -aN$ is the point at which the inter-agent attraction between individuals is not guaranteed, due to the counterbalance given by the repulsion from the resource potential. If the repulsion term is large enough (desire to keep away from danger), then it is not possible to guarantee the cohesiveness of the swarm.

«Such behavior can be seen in, for example, fish schools when a predator attacks the school.» ([Gazi e Passino, 2011, p. 47]([28]). Here Gazi and Passino give and example of a situation where the desire to get away from danger is larger than the desire to stick together.

In the following section, some resource profile functions will be discussed. In particular, plane, quadratic, Gaussian, and multi-modal Gaussian profiles.

2.3.3.1 Plane Resource Profile

In this section it is assumed that the resource profile function is described by a plane equation of the form:

$$\sigma(y) = a_\sigma^T y + b_\sigma \tag{2.49}$$

where $a_\sigma \in \Re^n$ and $b_\sigma \in \Re$. The gradient of the profile is given by

$$\nabla_t \sigma(y) = a_\sigma \tag{2.50}$$

Assumption 2 holds for $\overline{\sigma} = \|a_\sigma\|$. However, note that:

$$\nabla_{x_i}\sigma(x_i) - \frac{1}{N}\sum_{j=1}^{N}\nabla_{x_j}\sigma(x_j) = a_\sigma - \frac{1}{N}\sum_{j=1}^{N} a_\sigma = 0 \tag{2.51}$$

for all i. Therefore, from Theorem 1 one can deduce that the bound on the swarm size will be:

$$\varepsilon = \varepsilon_p = \frac{b(N-1)}{aN} < \frac{b}{a} \tag{2.52}$$

Note also that

$$\dot{\bar{x}}(t) = -a_\sigma \tag{2.53}$$

Implying that the centroid of the swarm will move with the constant velocity vector $-a_\sigma$. «The motions in this section can be viewed as a model of a foraging swarm that moves in a constant direction (while keeping its cohesiveness) with a constant speed» ([Gazi e Passino, 2011, p. 48]([28])

If the system is transformed into e_i coordinates, it is possible to obtain:

$$\dot{e}_i = \sum_{j=1, j\neq i}^{N} g(e_i - e_j), i = 1, \ldots, N \tag{2.54}$$

which is exactly the model of an aggregating swarm discussed in the previous section.

2.3.3.2 Quadratic Resource Profiles

In this section, a quadratic resource profile will be considered:

$$\sigma(y) = \frac{A_\sigma}{2}\|y - c_\sigma\|^2 + b_\sigma, \tag{2.55}$$

where $A_\sigma \in \Re$, $b_\sigma \in \Re$, and $c_\sigma \in \Re^n$. Note that this profile has a global maximum or minimum, depending on A_σ sign, at $y = c_\sigma$. Its gradient at a point $y \in \Re^n$ is given by:

$$\nabla_y \sigma(y) = A_\sigma(y - c_\sigma) \tag{2.56}$$

Assuming that $A_\sigma > -aN$. Then, with a few manipulations one can show that there is a bound on the swarm.

$$\varepsilon_q = \varepsilon_2 = \frac{b(N-1)}{aN + A_\sigma} \tag{2.57}$$

The motion of the centroid $\bar{x}$ will be given by:

$$\dot{\bar{x}} = -A_\sigma(\bar{x} - c_\sigma) \tag{2.58}$$

Defining the distance between the centroid $\bar{x}$ and the extremum point c_σ (maximum or minimum point) as $e_\sigma = \bar{x} - c_\sigma$ it is possible to demonstrate that:

$$\dot{e_\sigma} = -A_\sigma e_\sigma, \tag{2.59}$$

implying that if $A_\sigma > 0$ then as $t \to \infty$ $e_\sigma(t) \to 0$ and if $A_\sigma > 0$ $e_\sigma(t) \to \infty$.

This means that for swarms with the dynamics described in this chapter, as $t \to \infty$ there are two possibilities:

1. If $A_\sigma > 0$, then $\bar{x}(t) \to c_\sigma$
2. If $A_\sigma < 0$, then $\bar{x}(t) \to \infty$

Note that for the case of $A_\sigma > 0$, it can be shown that $\|\bar{x} - c_\sigma\| < \varepsilon^*$ in a finite time for any $\varepsilon^* > 0$, no matter how small. In other words, the $\bar{x}$ will enter ε^* neighborhood of c_σ in a finite time. In contrast, if $A_\sigma < 0$ and $\bar{x}(0) \neq c_\sigma$ then the swarm will leave any $D > 0$ neighborhood in a finite time. If $\bar{x}(0) = c_\sigma$ and $A_\sigma < 0$, then $\bar{x}(t) = c_\sigma$ for all t, meaning that the swarm centroid will be stuck around the maximum point because of the inter-individual attraction will disperse in all directions if the inter-individual attraction is not strong enough ($A_\sigma < -aN$).

«As mentioned above, such a dispersal behavior can be seen in fish schools when attacked by a predator... the effect of the presence of a predator can be represented by a large intensity quadratic repellent resource profile» ([Gazi e Passino, 2011, p. 50]([28]).

Now, assume that instead of the quadratic resource profile following the function in equation 3.24, a profile which is a sum of quadratic functions will be used. In other words, assume that the resource profile is given by:

$$\sigma(y) = \sum_{i=1}^{M} \frac{A_{\sigma i}}{2}\|y - c_{\sigma i}\|^2 + b_\sigma \tag{2.60}$$

where $A_{\sigma i} \in \Re$, and $c_{\sigma i} \in \Re^n$ for all $i = 1, ..., M$, and $b_\sigma \in \Re$. Its gradient is given by:

$$\nabla_y \sigma(y) = \sum_{i=1}^{M} A_{\sigma i}(y - c_{\sigma i}). \tag{2.61}$$

Defining

$$A_\sigma = \sum_{i=1}^{M} A_{\sigma i} \tag{2.62}$$

and

$$c_\sigma = \frac{\sum_{i=1}^{M} A_{\sigma i} c_{\sigma i}}{\sum_{i=1}^{M} A_{\sigma i}} \tag{2.63}$$

one can obtain:

$$\nabla_y \sigma(y) = A_\sigma(y - c_\sigma) \tag{2.64}$$

which is the same result obtained before.

Quadratic resource profiles are relatively simple to understand, however, it is possible to have more complicated resource profiles, modeled as quadratic in regions near extremum points. In the following sections, other resource profile functions will be studied which are not necessarily quadratic.

2.3.3.3 Gaussian Resource profile

In this section, we consider resource profiles described by a Gaussian-type of equation,

$$\sigma(y) = -\frac{A_\sigma}{2} \exp\left(-\frac{\|y - c_\sigma\|^2}{l_\sigma}\right) + b_\sigma, \tag{2.65}$$

where $A_\sigma \in \Re$, $b_\sigma \in \Re$, $l_\sigma \Re^+$ and $c_\sigma \in \Re^n$. Its gradient is given by:

$$\nabla_y \sigma(y) = \frac{A_\sigma}{l_\sigma}(y - c_\sigma) \exp\left(-\frac{\|y - c_\sigma\|^2}{l_\sigma}\right) \tag{2.66}$$

Using equation 2.43, it is possible to calculate the time derivative of the centroid:

$$\|\nabla_y \sigma(y)\| \leq \frac{\|A_\sigma\|}{\sqrt{2 l_\sigma}} \exp - \left(\frac{1}{2}\right) \tag{2.67}$$

Thus, Theorem 1 holds and it is known that as $t \to \infty$ all the individuals will connverge to the

$$\varepsilon_1 = \frac{(N-1)}{aN}\left[b + \frac{\|A_\sigma\|}{N}\sqrt{\frac{2}{l_\sigma}} \exp\left(-\frac{1}{2}\right)\right] \tag{2.68}$$

neighborhood of the (mobile) centroid $\dot{\bar{x}}$

The author then analyses the motion of the centroid swarm using Lyapunov and is able to reach the following theorem:

Theorem 2 - Consider a foraging swarm consisting of agents with dynamics in equation 2.69 and with control input in function 2.9 with inter-agent attraction/repulsion function $g(\dot{)}$ given by the function 2.11 which is odd, satisfies Assumption 1 and has linear attraction and bounded repulsion. Assuming the resource profile $\sigma(\dot{)}$ of the environment is given by the function in 2.65. Then, as $t \to \infty$ we have:

- If $A_\sigma > 0$, then all individuals will enter $B_{2\varepsilon_G}$.
- If $A_\sigma < 0$ and $\|e_\sigma(0)\| \geq e_m(0)$, then all individuals will exit $B_D(c_\sigma)$ for any fixed $D > 0$

2.3.4 Pattern Formation

In pattern formation one aims to achieve a certain configuration of the individuals in the environment. This can be useful for many situation, such as reducing the area of the environment uncovered by the drones in order to collect important information, avoiding objects or for collective procedures where a certain formation would benefit its execution.

Pattern formation can be found both in biology and in physics. Some biological examples are the spatial disposition of bacterial colonies, the chromatic patterns on some animal's fur and schooling in fish: fish often form schools in which they swim in coordinated patterns.

In centralized pattern formation methods, a computational unit oversees the whole group and plans the motion of the group members accordingly. The command signal with the desired motion is then transmitted through a communication channel.[31]

In [32], a coordination strategy for moving a group of robots in a desired formation over a given path is proposed. Path planning is separated from the path tracking task. They applied this method to coordinate the motion of simulated robots in a triangular formation while being able to avoid obstacles. The paper proves that, if the tracking errors of the robots are bounded or tracking is done perfectly, then the described method stabilizes the formation error.

Belta and Kumar [33] propose a centralized trajectory computation scheme that uses kinetic energy shaping. Instead of using a constant kinetic energy metric, they employ a smooth change in the kinetic energy metric which helps achieving the desired behaviour. The method generates smooth trajectories for a set of robots. The distance between the robots can be controlled via a parameter. However the method does not take obstacle avoidance into consideration and is not scalable.

2.3.5 Motion Control Problems

The concepts presented in this section are based on [9].

This section formulates a collection motion control problems for autonomous robotics vehicles. Consider a framework composed by a group of N autonomous vehicles by general continuous systems of the form:

$$\dot{x}^{[i]} = f^{[i]}(x^{[i]}(t), u^{[i]}(t)), x^{[i]}(0) = x_0^{[i]} \tag{2.69}$$

$$y^{[i]} = h^{[i]}(x^{[i]}(t)) \tag{2.70}$$

2.3.5.1 Point stabilization and way-point tracking

This task refers to the problem of steering and stabilizing a vehicle at a given position with a desired orientation.

Way-point tracking problem: Consider single vehicle described by the state equation in 2.69 and output equation in 2.70 and let p = $\{p_1, p_2, ..., p_n\}$ be a given sequence of n way-points with $p_i \in \Re^q$, with i = 1, 2,..., n be a given desired points. Consider a closed ball B_{ε_i} with center p_i and radius $\varepsilon > 0$ such that:

$$B_{\varepsilon_i} = \{p \in \Re^q : \|p - p_i\| \leq \varepsilon_i\} \tag{2.71}$$

Design a feedback control law such that all the relevant closed-loop signals are bounded and the signal y(t) - p_n converges to zero as t → ∞ after reaching the ordered sequence of the neighborhoods.

Way-point tracking can in principle be done in a number of ways. Most of them lend themselves to intuitive interpretation but may lack a solid theoretical background. Perhaps the most widely known is so-called line-of-sight scheme

Way-point tracking can be done in many ways. However, the most widely known is called line-of-sight scheme. In this case, the vehicle is simply rotated until the main axis of the vehicle is aligned along the line of sight between the present position of the vehicle and the way-point to be reached.

2.3.5.2 Trajectory tracking and path following

Trajectory tracking and path following are typical motion control tasks that autonomous robotic vehicles are required to execute. Trajectory tracking is concerned with with the design of control laws that force a given vehicle to track a desired trajectory. Path-following is only required to steer the vehicle towards a path, without time constraints, and make it follow the path with an assigned speed profile that in some cases may be path dependent.

Trajectory tracking problem: Consider a single vehicle described by the state equation in 2.69 and output function in 2.70 and let $y_d(t) : \Re_{\geq 0} \to \Re^q$ be a given desired reference trajectory. Design a feedback control law such that all the relevant closed-loop signals are bounded and the tracking error is $\|y(t) - y_d(t)\|$ converges to zero as $t \to \infty$.

Path-following problem: Consider a single vehicle described by the state equation in 2.69 and output function in 2.70 and let $y_d(\gamma) : \Re \to \Re^n$ be a given desired path parameterized by $\gamma \in \Re$ and v_d a desired speed assignment. Design a feedback control law such that all the relevant closed-loop signals are bounded and the path following error $\|y(t) - y_d(\gamma(t))\|$ converges to zero as $t \to \infty$, and the parameter γ satisfies the speed assignment $\dot{\gamma} \to v_d$ as $t \to \infty$.

2.4 Crazyflie

2.4.1 Overview

Crazyflie 2.1 (Figure 2.2) is the second version of Crazyflie drones developed by Bitcraze, it is a versatile open source flying development platform that only weighs 27g and fits in the palm of a human hand. Crazyflies are small compact drones that allow the installation of different types of modules, such as the AI deck, the LED-ring deck and Micro SD card.

Figure 2.2: Crazyflie 2.1. Source:[3]

2.4.2 Cascade PID Controller

The concepts presented in this section are based on [4].

Bitcraze offers some controllers for the dynamics of the crazyflie 2.1. There are four levels to control in the crazyflie:

- Attitude rate
- Attitude
- Velocity
- Position

The controller's architecture can be seen in Figure 2.3.

2.4.3 LOCO Positioning System

The concepts presented in this section are based on [34].

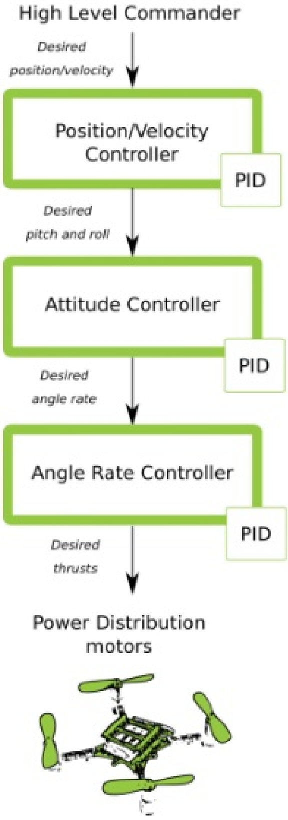


Figure 2.3: Crazyflie's controller architecture. Source:[4]

The Loco Positioning System is an absolute positioning system based on Ultra Wide Band (*UWB*) radios. It has two modes: Two-way ranging (*TWR*) and Time-difference-of-arrival(*TDoA*). Since *TWR* can only be used to calculate the position of a single drone, only *TDoA* method will be discussed.

In *TDoA*, *UWB* tags receive signals from anchors passively and compute the difference in distance between two anchors as Time Difference of arrival measurements. *UWB* tags only listen to the messages from anchors, a *TDoA*-based localization system allows a theoretically unlimited number of robots to localize themselves with a small number of fixed anchors. Two types of *TDoA* protocols (*TDoA2* and *TDoA3*) are implemented in *LPS* system. The main difference between the two *TDoA* protocols is that *TDoA3* protocol achieves scalability at the cost of localization accuracy.

The *TDoa* measurement model is as follows:

$$d_{ij} = dj - di = \sqrt{(x-x_j)^2 + (y-y_j)^2 + (z-z_j)^2} - \sqrt{(x-x_i)^2 + (y-y_i)^2 + (z-z_i)^2} \tag{2.72}$$

where (x,y,z) are the position of the crazyflie and (x_i, y_i, z_i) and (x_j, y_j, z_j) the position of the anchors i and j, respectively.

2.5 Simulators

In robotics, simulation plays a pivotal role in research, development, and testing. Robotics simulators provide a virtual playground where it is possible to experiment, refine, and optimize their systems without the constraints of the physical world. These simulators not only save time and resources but also enable the rapid prototyping of complex robotic applications.

This chapter covers some of the most used simulators in the robotics world, three different types will be discussed:

1. General-purpose platforms like Gazebo and Webots

2. Specialized tools for realistic vision like NVIDIA Isaac and AIRsim

3. Swarm oriented simulators like ArGos and Pybuller Gym

The main focus will be Gazebo and Webots since Bitcraze already has a simulation world implemented and free to use in these platforms. However it is possible to implement and test an architecture in other simulators using both their libraries and Bitcraze's libraries.

2.5.0.1 Gazebo

Gazebo is a 3D simulator able to accurately and efficiently simulate groups of robots in complex indoor and outdoor environments. It also offers physics simulation at a much higher degree of fidelity, a suite of sensors and interfaces for both users and programs.[5]

It is used to test robotics algorithms, designing robots and to perform regression testing with realistic scenarios. The simulator has multiple physics agents, a rich library of robot models and environments, a wide variety of sensors and convenient programmatic and graphical interfaces.

Understanding the GUI

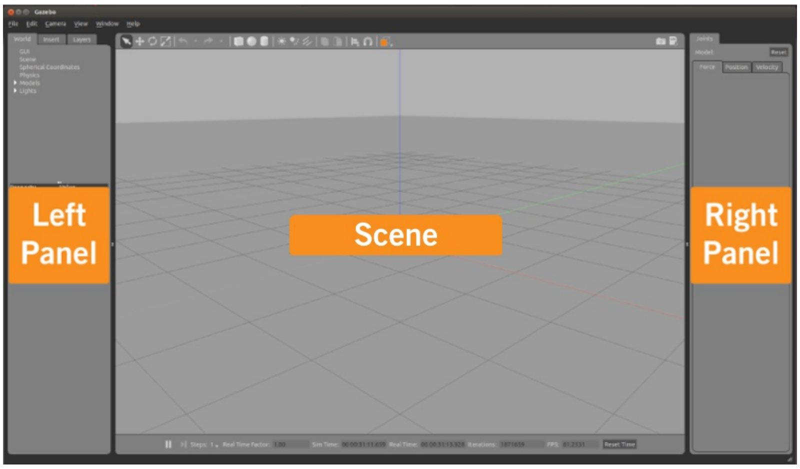


Figure 2.4: Gazebo GUI. Source:[5]

As shown in Figure 2.4, Gazebo's graphical user interface is composed by:

1. **The scene** - Where the simulated objects are animated and it is possible to interact with the environment.

2. **Left Panel**

   (a) **World** - Displays the models that are currently in the scene and allows the user to view and modify model parameters, like their pose. The user can also change the camera view angle by expanding the *GUI* option and tweaking the camera pose.

   (b) **Insert Tab** - Where the user can add models to the simulation.

   (c) **Layers** - Organizes and displays the different visualization groups that are available in the simulation, if any.

3. **Right Panel** - Can be used to interact with the mobile parts of a selected model.

4. **Toolbars**

   (a) **Upper Toolbar** - Includes some of the most used options for interacting with the simulators such as buttons to: select, move, rotate, scale objects and create different shapes.

   (b) **Bottom Toolbar** - Displays data about the simulation like simulation time and its relationship to real-life time.

Since this dissertation is focused on the Crazyflie robot, the model editor user interface is not of great importance. To implement the Crazyflie 2.1 robot it will be necessary to learn how to write a *SDF* (Simulation Description Format) files. However, Bitcraze already implemented a gazebo world to test Crazyflie aplications with a *SDF* file. This constitutes a great advantage to use the gazebo simulator.

Bitcraze offers a github repository [35] where both Gazebo and Webots worlds are implemented. Adding to that, some controllers are already implemented and it is possible to test them in Webots. They also developed a set of instructions for installing gazebo and using the world developed.

The only issue with Gazebo is that Bitcraze's model is not fully implemented and the controllers can only rotate the propellers for now.

2.5.1 Webots

Webots is another general-purpose mobile robot simulation software package designed to help engineers and researchers prototype and experiment with various robotic systems. It has an extensive library, much like gazebo, with pre-built robots that allows the user to create 3D virtual worlds with physics properties such as mass, joints, friction coefficients, etc. These robots can have different locomotion schemes (wheeled robots, legged robots, or flying robots) and may be equipped with a number of sensor and actuator devices, such as distance sensors, drive wheels, cameras, motors, touch sensors, emitters, receivers, etc. [6]

The simulator contains a number of interfaces to real mobile robots like e-puck, DARwIn-OP and Nao. It is also possible to add new interfaces through the related systems.

In addition, Webots supports multiple programming languages, making it accessible to developers from different backgrounds and expertise levels.

2.5.1.1 Introduction to Webots

Webots Simulation

A webots simulation is composed by three elements:

1. A Webots world file (*.wbt*) that defines one or several robots and their environment. The .wbt file does sometimes depend on external *PROTO* files (.proto) and textures, mainly when implementing complex robots like the crazyflie drone.
2. One or several controller programs for the above robots (in C/C++/Java/Python/MATLAB).
3. An optional physics plugin that can be used to modify Webots regular physics behavior (in C/C++).

Thankfully, Bitcraze already developed these three elements and the scope of this dissertation will be changing the controller (second element) developed by bitcraze to a swarm oriented one able to perform some basic flocking tasks. However, there is some interest in understanding these items, since they may be relevant in the future if one decides to increase the complexity of the simulations, such as obstacle avoidance in a path-following problem or a simple aggregation in a building with multiple compartments.

Webots World

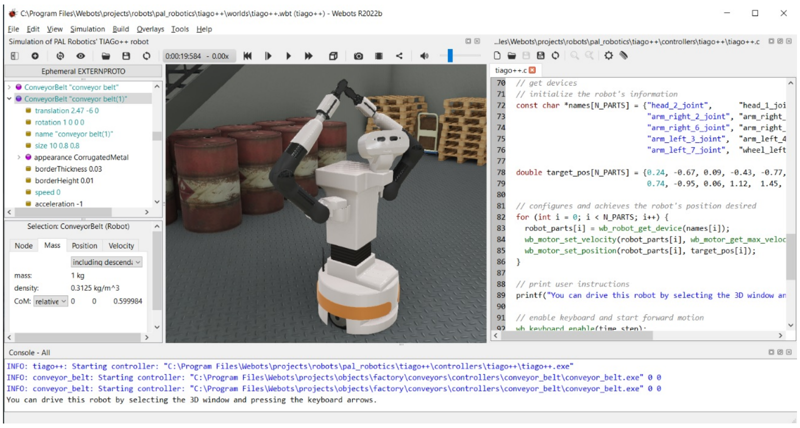
The world file in webots contains a 3D description of the properties of robots and of their surroundings (floor, light, objects, etc). It contains the description of every object: position, orientation, geometry, appearance, physical properties, type of object, etc. Worlds are organized as hierarchical structures where objects can contain other objects, for this application the drone would be equipped with four propellers, a distance sensor, etc. Worlds are saved in ".wbt" files that are stored in the "worlds" subdirectory of each Webots project. [6]

Webots Controller The Webots controller program is supposed to control the robot specified in the world file. It can be written in C, C++, Java, Pyhton or MATLAB. At the start of the simulation, Webots runs the specified controllers in the world file, each as a separate process. It is important to notice that multiple robots may be running the same controller, but a distinct process will be launched for each robot.

The source files and binary files of each controller are stored together in a controller directory that is placed in the "controllers" subdirectory of each Webots project. [6]

Webots GUI

Webots *GUI*, shown in figure 2.5, is composed of four principal windows:


C:\Program Files\Webots\projects\robots\pal_robotics\tiago++\worlds\tiago++.wbt (tiago++) - Webots R2022b
File Edit View Simulation Build Overlays Tools Help
Simulation of PAL Robotics' TIAGo++ robot
0:00:19:584 - 0.00x
Ephemeral EXTERNPROTO
...les\Webots\projects\robots\pal_robotics\tiago++\controllers\tiago++\tiago++.c
tiago++.c
Console - All

```
ConveyorBelt "conveyor belt"
ConveyorBelt "conveyor belt(1)"
  translation 2.47 -6 0
  rotation 1 0 0 0
  name "conveyor belt(1)"
  size 10 0.8 0.8
  appearance CorrugatedMetal
  borderThickness 0.03
  borderHeight 0.01
  speed 0
  acceleration -1
```

Selection: ConveyorBelt (Robot)

Node | Mass | Position | Velocity

including descend:

mass: 1 kg

density: 0.3125 kg/m^3

CoM: relative 0 0 0.599984

```c
// get devices
// initialize the robot's information
const char *names[N_PARTS] = {"head_2_joint",       "head_1_joi
                              "arm_right_2_joint", "arm_right_
                              "arm_right_6_joint", "arm_right_
                              "arm_left_3_joint",  "arm_left_4
                              "arm_left_7_joint",  "wheel_left

double target_pos[N_PARTS] = {0.24, -0.67, 0.09, -0.43, -0.77,
                              0.74, -0.95, 0.06, 1.12,  1.45,

// configures and achieves the robot's position desired
for (int i = 0; i < N_PARTS; i++) {
  robot_parts[i] = wb_robot_get_device(names[i]);
  wb_motor_set_velocity(robot_parts[i], wb_motor_get_max_veloc
  wb_motor_set_position(robot_parts[i], target_pos[i]);
}

// print user instructions
printf("You can drive this robot by selecting the 3D window an

// enable keyboard and start forward motion
wb_keyboard_enable(time_step);
```

```
INFO: tiago++: Starting controller: "C:\Program Files\Webots\projects\robots\pal_robotics\tiago++\controllers\tiago++\tiago++.exe"
INFO: conveyor_belt: Starting controller: "C:\Program Files\Webots\projects\objects\factory\conveyors\controllers\conveyor_belt\conveyor_belt.exe" 0 0
INFO: conveyor_belt: Starting controller: "C:\Program Files\Webots\projects\objects\factory\conveyors\controllers\conveyor_belt\conveyor_belt.exe" 0 0
You can drive this robot by selecting the 3D window and pressing the keyboard arrows.
```

Figure 2.5: Webots GUI. Source:[6]

1. The **3D window** that displays and allows you to interact with the 3D simulation.

2. The **Scene tree** which is a hierarchical representation of the current world.

3. The **Text editor** that allows you to edit source code.

4. The **Console** that displays both compilation and controller outputs.

It has eight menus: **File**, **Edit**, **View**, **Simulation**, **Build**, **Overlays**, **Tools** and **Help**.

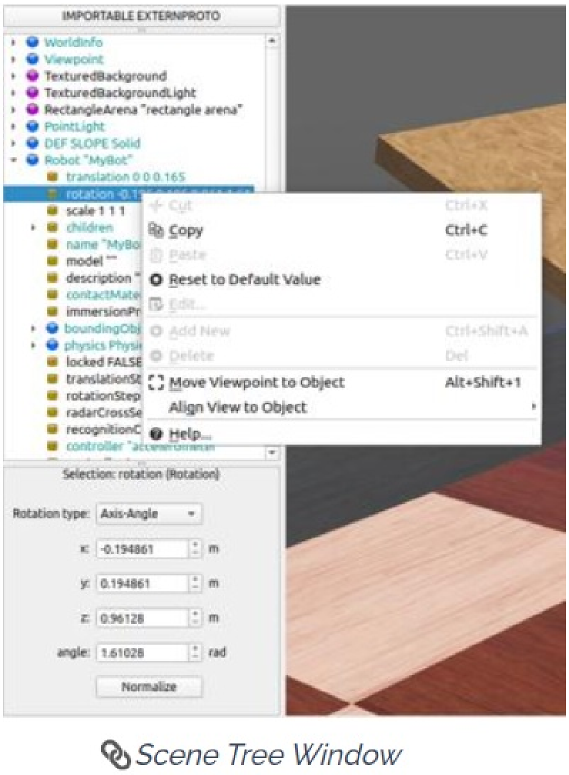
The **File** menu has a submenu called **New** that allows the user to create new world files, new robot controller files, new physics plugin or new *PROTO* files. Besides creating new files, it is possible to save the current world, load other worlds, open text files, take screenshots or film the simulation.

Furthermore, webots allows the user to edit the code in the simulator without with the open editor tab, without the need of an *IDE*. Nonetheless it is possible to integrate your *IDE* with webots, there is a guide in [6] for anyone interested in setting their *IDE* with the controller.

To sum up, webots has all the possible actions the user would have building the world offline in its' interface, making it a lot easier to create objects and specifying their dynamics.

The Scene Tree

The Scene Tree is presented in the left panel of Webots, and it is where the user can navigate through every node of the world currently loaded. It is composed of a list of nodes, each containing fields. Fields can contain values (text strings, numerical values) or other nodes. In the case of being a robot node with a *PROTO* file, it is possible to open its controller in the text editor tab.



Selection: rotation (Rotation)

Rotation type: Axis-Angle

x: -0.194861 m

y: 0.194861 m

z: 0.96128 m

angle: 1.61028 rad

Figure 2.6: Webots Scene Tree. Source:[6]

Figure 2.6 illustrates a Scene Tree of world in Webots.

2.5.1.2 Supervisor Programming

Webots has a special node for robots called Supervisor node, this node is used to control certain aspects that can normally only be carried out by a human operator and not by a real robot.

The Webots supervisor node allows access to a set of exclusive operations, including simulation control tasks such as manipulating objects and recording the simulation. It is able to track the position of any robot in the simulation, set the robots' positions, load or reload the world, quit the simulation, adding or removing nodes to the world, etc.

It is nothing more than a Robot with extra powers, therefore it is able to do anything a Robot instance does. This node will probably come in hand when demonstrating the simulation results, as it can be helpful to illustrate better the motion of the drones.

2.5.1.3 Webots Github

Webots also has several sample applications in their github [36] for different types of worlds such as gantry robot examples, soccer fields, a moon world with some ground vehicles, appartments, factories, among many. They have also examples on how to instantiate devices in the world files and their controller codes too. There are examples for the accelerometer, emitter and receiver, distance sensors and camera, just to name a few.

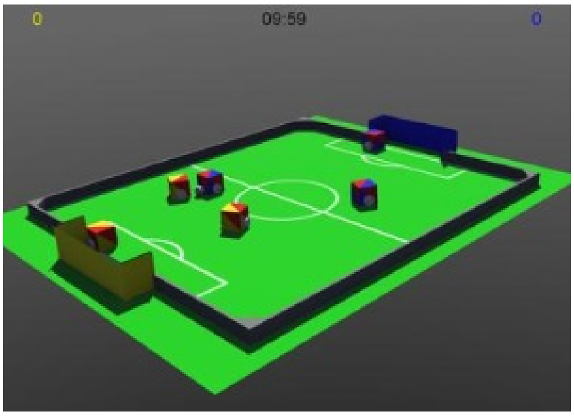


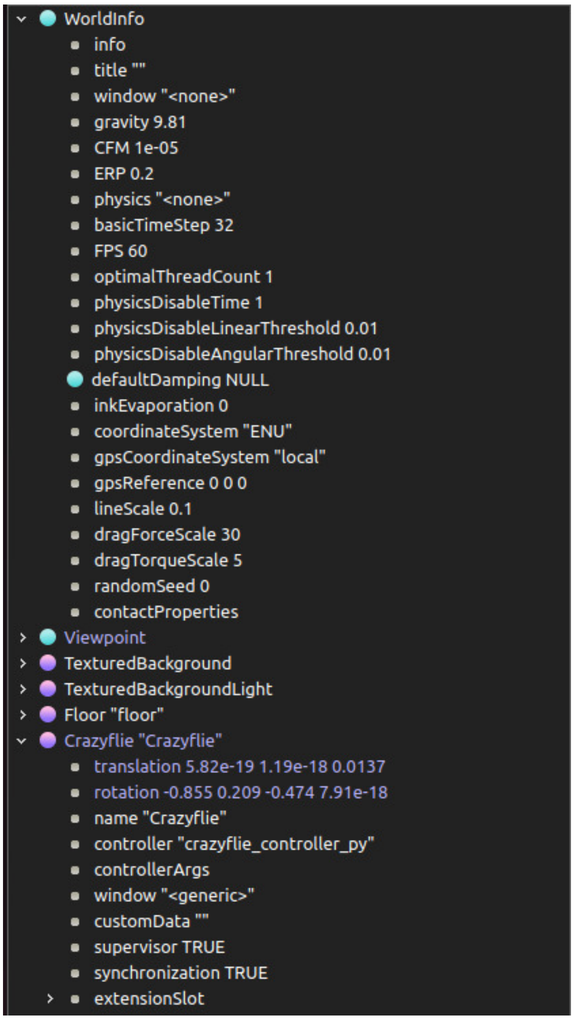
Figure 2.7: Soccer Demo world example. Source:[6]

Figure 2.7 shows a world sample application from Webots with a soccer field. In this implementation, a supervisor node is used as a referee, it counts the goals and displays the current score and the remaining time in the 3D view.

2.5.1.4 Bitzcraze's Webots World

As mentioned befor, bitcraze has already a Gazebo and a Webots world implementation with two simple controllers, a keyboard controller and a wall following controller for an apartment[35]. In order to save time, the world developed by bitcraze will be used and the goal will be changing the existing controller designed for one drone into a swarm controller able to perform the motion problems presented above.

Bitcraze developed a ***PROTO* file** for the crazyflie 2.1, equipped with all the sensors and actuators needed to perform low level control (accelerometer, gyroscope, proppelers, etc) and also a camera which would be interesting to explore for some computer vision algorithms.



```
⌄ ● WorldInfo
    ■ info
    ■ title ""
    ■ window "<none>"
    ■ gravity 9.81
    ■ CFM 1e-05
    ■ ERP 0.2
    ■ physics "<none>"
    ■ basicTimeStep 32
    ■ FPS 60
    ■ optimalThreadCount 1
    ■ physicsDisableTime 1
    ■ physicsDisableLinearThreshold 0.01
    ■ physicsDisableAngularThreshold 0.01
    ● defaultDamping NULL
    ■ inkEvaporation 0
    ■ coordinateSystem "ENU"
    ■ gpsCoordinateSystem "local"
    ■ gpsReference 0 0 0
    ■ lineScale 0.1
    ■ dragForceScale 30
    ■ dragTorqueScale 5
    ■ randomSeed 0
    ■ contactProperties
› ● Viewpoint
› ● TexturedBackground
› ● TexturedBackgroundLight
› ● Floor "floor"
⌄ ● Crazyflie "Crazyflie"
    ■ translation 5.82e-19 1.19e-18 0.0137
    ■ rotation -0.855 0.209 -0.474 7.91e-18
    ■ name "Crazyflie"
    ■ controller "crazyflie_controller_py"
    ■ controllerArgs
    ■ window "<generic>"
    ■ customData ""
    ■ supervisor TRUE
    ■ synchronization TRUE
  › ■ extensionSlot
```

Figure 2.8: Bitcraze's Webots Scene Tree

Figure 2.8 shows the Scene Tree presented if we load Bitcraze's world. As you can see there are very few nodes in the world, it is a pretty simple implementation, but will be more that enough for the motion control problems in hand.

Bitcraze's low level dynamics controller will also be used since the high level swarm motion control is the main focus of this dissertation. However, bitcraze already stated in a meeting that the controller could be improved and it may not work in some real time applications.

All the files mentioned above can be found in their git repository.

2.5.2 ARGoS

ARGoS, [37] is a well know swarm robotics simulator, suitable for multi-robot experiments, where multiple robots can work together to accomplish tasks. This capability is particularly relevant for applications like search and rescue missions, environmental monitoring, and swarm robotics research. Like in webots, ARGoS treats the physical engine as a plug-in, allowing the user to choose which physical engine to use in the simulation.

2.5.3 Pybullet gym

Pybullet gym is another swarm robotics simulator, like ARGos. It is a combination of **OpenAI Gym**[38], an open-source toolkit for developing and comparing reinforcement learning, and **PyBullet** [39], a python module for robotics simulation and machine learning, with a focus on sim-to-real transfer, in other words, it is a Python wrapper around the Bullet Physics Library, which is a widely-used physics simulation engine

2.6 Summary

As described in this chapter, there are many implementations of the problems related to this work in the world. Adding to that, a lot of research is being made in this area in order to achieve more complex and autonomous systems capable of performing tasks in hazardous environments.

While the potential aggregation/repulsion function used by Gazi provides a good starting point, it still can not fully replicate behaviours observed in nature like flocks or schools of fish. This centralized approach relies on the fact that every individual has full knowledge of the environment it is placed to perform any given task. This is neither the case of a biological swarm nor of a idealist swarm robotics application. It is straightforward to notice how an application whose individuals have complete knowledge of the environment will decrease in efficiency as the number of agents increase, it also clear that such does not happen in nature. For that reason it is crucial to explore other methods to better understand their trade-offs and how they tackle scalability issues.

It is necessary to keep in mind that implementing these in the real world with physical robots introduces additional challenges. The assumptions made in the models, such as perfect communication or knowledge of every other agent's position, often do not hold in reality. Hence, it is crucial to develop models and algorithms keeping these practical considerations in mind. To address these issues, different error mitigating techniques and robust control laws will be explored and implemented as a part of this study. The next part of the dissertation will focus on these practical considerations and discuss the implementation details of our chosen models on the Crazyflie 2.1 drones.

Crazyflie 2.1 will be the drone used to implement and test different algorithms mentioned above. It has an embedded controller which will be used to control the attitude rate, the attitude, the velocity and the position of the drone, this way, there is no need to develop a controller for the dynamics of each drone. Moreover, the *LOCO* Positioning System will be used to obtain the drones' position. With this two modules of the Crazyflie it will be possible to implement the high-level control of the drone flock.

Since the algorithms developed in this dissertation will need to be tested, some robotics simulators were studied and presented at the end of this section. The main focus was general purpose platforms like Gazebo and Webots, being the most used and having a world developed by bitcraze with the crazyflie drone played an enormous role in this choice. Both simulators work a in a similar way, it is possible to build robots using their tools or using *SDF* files and test their behaviour. The decisive criteria ended up being the fact that Bitcraze's Gazebo implementation is not fully working, making it more difficult to run and test the algorithms as it would be necessary to complete its implementation before using it.

Chapter 3

Motion Control Methods

This chapter delves into the practical aspects of swarm motion control by implementing some of the models presented earlier in python. There is no particular reason for choosing python besides its broad set of functions and the amount of information that can be found.

3.1 Aggregation

The first motion control problem implemented was aggregation since it is the most straightforward one. The model chosen for the first aggregation algorithms was Veysel Gazi's Attraction/Repulsion Model [28]. In this dissertation the three types of aggregation functions from Gazi's book will be implemented and an additional function for the Rigid Body scenario.

3.1.1 Linear Attraction and Bounded Repulsion Case

In this section we consider the special case in which the attraction part satisfies

$$g_a(\|y\|) = a \tag{3.1}$$

for some finite positive constant $a > 0$ and for all y whereas the repulsion is constant or bounded as

$$g_r(\|y\|)\|y\| \leq b, \tag{3.2}$$

for some finite constant b. Note that the function already referenced in the previous chapters satisfies both of these conditions.

Additionally, it is proven that as time progresses all the members of the swarm will converge to a hyper-ball

$$B_\varepsilon(\bar{x}) = \{x : \|x - \bar{x}\| \leq \varepsilon\}, \tag{3.3}$$

where $\varepsilon = \frac{b}{a}$.

So using function 2.11 a python algorithm was made to test its results. The number of agents in the system chosen was 8, the distances were initialized with random numbers between 0 and 60. The threshold distance chose was 3 and the equilibrium distance 2.

The same values used on the book were chosen so that it is possible to compare the results. The parameter were (a = 1, b = 20, c = 0.2), for this parameters the swarm members are expected to converge to a ball with radius $\varepsilon = 3.8$.

The algorithm runs a loop until all the distances of the individuals is less than the convergence ball radius:

```
while distance > threshold_distance:
    Calculate control forces: $u_i = -\sum_{i=1}^{N-1}\sum_{j=i+1}^{N}(x_i - x_j)\left[a - b\exp\left(-\frac{\|x_i - x_j\|^2}{c}\right)\right]$
    Calculate new positions: x = $\Delta_t \cdot u_i$
    update distance matrix: $d_{ij} = \|x_i - x_j\|$
    check for collisions: if any $d_{ij} = 0$ break;
    Calculate new average distance to centroid: $\frac{1}{N}\sum_{i=1}^{N}\|x_i - \bar{x}\|$
end
```

The results are shown in this following pictures:

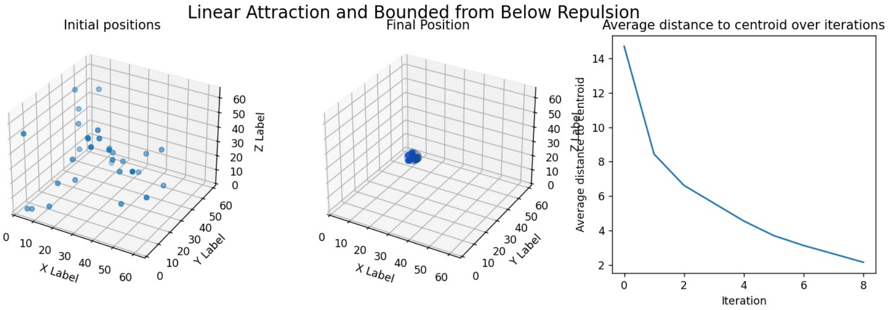


Figure 3.1: Linear Attraction and Bounded Repulsion Case

It is fairly easy to notice in figure 3.1 that the average distance to the centroid doesn't decline linearly just like in the book, that is probably due to the simplistic form the position is updated. Nonetheless, it was possible to achieve convergence and the individuals aggregated around the hyper-ball with radius = 3.8 as intended. To illustrate that better a sphere with the intended radius was drawn on the plot. In figure 3.2 it is possible to visualize the hyperball and the individuals in it.

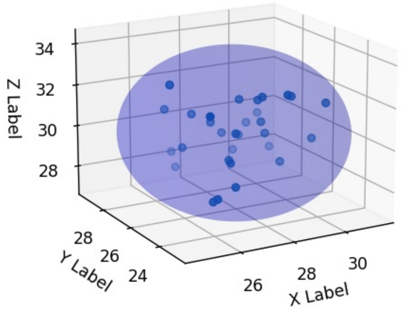


Figure 3.2: Linear Attraction and Bounded Repulsion Case HyperBall

3.1.2 Linearly Bounded from Below Attraction and Unbounded Repulsion

By linearly bounded from below attraction we mean the case in which we have

$$g_a(\|x_i - x_j\|) \geq a \tag{3.4}$$

for some finite constant a and for all $\|x_i - x_j\|$. For the repulsion functions, on the other hand, we will consider the unbounded functions satisfying

$$g_r(\|x_i - x_j\|) \leq \frac{b}{\|x_i - x_j\|^2} \tag{3.5}$$

The simplest example of attraction/repulsion function $g(\cdot)$ satisfying the above assumptions is the case in which the equalities are satisfied it is given by

$$g(x_i - x_j) = \left[a - \frac{b}{\|x_i - x_j\|^2}\right](x_i - x_j) \tag{3.6}$$

which corresponds to the potential function

$$J(x) = -\sum_{i=1}^{N-1} \sum_{j=i+1}^{N} \left[\frac{a}{2}\|x_i - x_j\| - b \ln \|x_i - x_j\|\right] \tag{3.7}$$

The book also proves that the error $e_i = x_i - \bar{x}$ is ultimately bounded by

$$\|e_i\| \leq \sqrt{\frac{bN}{2a}} \tag{3.8}$$

So using this equations it was only necessary to change the control equation in the algorithm for this new case. The parameters a and b were set to 5 and 0.5, respectively, to make the threshold distance small enough. The number of individuals was the same as in the previous case.

Figure 3.3 illustrates the results of the Linearly Bounded from Below Attraction and Unbounded Repulsion case.

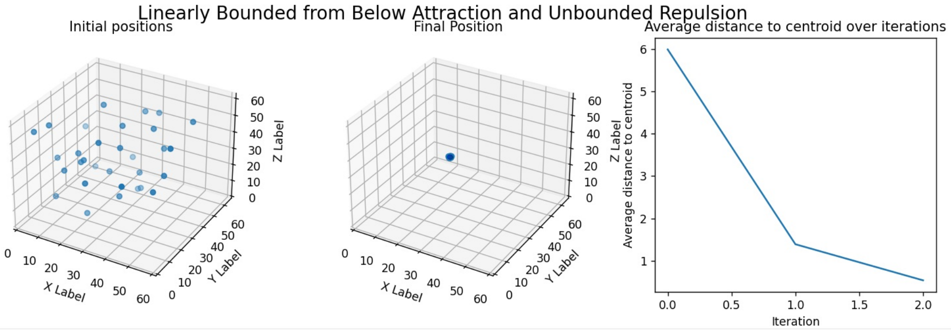


Figure 3.3: Linearly Bounded from Below Attraction and Unbounded Repulsion

3.1.3 Almost Constant Attraction and Unbounded Repulsion

In this section we discuss the attraction functions that satisfy $g_a(\|x_i - x_j\|) \geq \frac{a}{\|x_i - x_j\|}$. For the repulsion function we use the same type of functions as in the previous section. An example of attraction/repulsion function $g(\cdot)$ satisfying the above assumptions could be stated as

$$g(x_i - x_j) = \left[\frac{a}{\|x_i - x_j\|} - \frac{b}{\|x_i - x_j\|^2}\right](x_i - x_j) \tag{3.9}$$

Corresponding to the following potential function:

$$J(x) = -\sum_{i=1}^{N-1}\sum_{j=i+1}^{N}\left[a\|x_i - x_j\| - b\ln\|x_i - x_j\|\right] \tag{3.10}$$

And for this case, the ultimate bound on the swarm size will be

$$\|e_i\| \leq \frac{Nb}{a} \tag{3.11}$$

For this case the parameters chosen were a = 4, b = 0.2, so the threshold distance is close enough to one. The results for the Almost constant attraction and unbounded repulsion case are shown in figure 3.4.

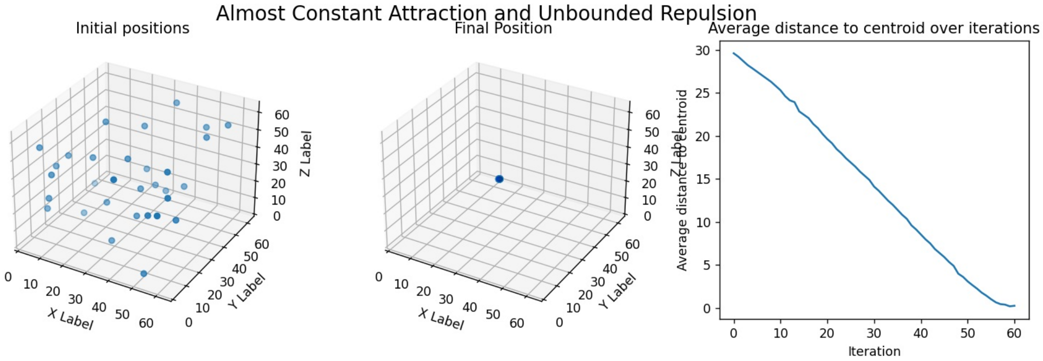


Figure 3.4: Almost Constant Attraction and Unbounded Repulsion

3.1.4 Agents with Finite Body Size

In this section instead of treating each individual as a point, the individuals will occupy space around them. In particular, we will consider individuals which are hyper spheres in the n-dimensional space.

In the previous section, the repulsion function g_r would tend to infinity when the euclidean distance between each individual tends to zero.

$$\lim_{\|x_i - x_j\| \to 0} g_r(\|x_i - x_j\|)\|x_i - x_j\| = \infty \tag{3.12}$$

Now if each individual has a radius η then in order not to collide $\|x_i - x_j\| > 2\eta$ needs to be satisfied. So the repulsion function should tend to infinity when the euclidean distance between each individual tends to two times the radius of each individual.

$$\lim_{\|x_i - x_j\| \to 2\eta} g_r(\|x_i - x_j\|)\|x_i - x_j\| = \infty \tag{3.13}$$

One repulsion function which satisfies the above condition is given by

$$g_r(\|x_i - x_j\|) = \left[\frac{b}{(\|x_i - x_j\|^2 - 4\eta^2)^2}\right] \tag{3.14}$$

Corresponding to the following potential function

$$J_r(\|x_i - x_j\|) = -\frac{b}{2(\|x_i - x_j\|^2 - 4\eta^2)} \tag{3.15}$$

For this function, the minimum radius the swarm can have is

$$r_{min} = \eta \sqrt[n]{N} \tag{3.16}$$

To implement this case the Almost Attraction and Unbounded repulsion function was chosen with the radius term added. The threshold distance chosen was 4, the drone radius 0.3 and the parameters were: a = 1.9 and b = 0.38. Figure 3.5 illustrates the results.

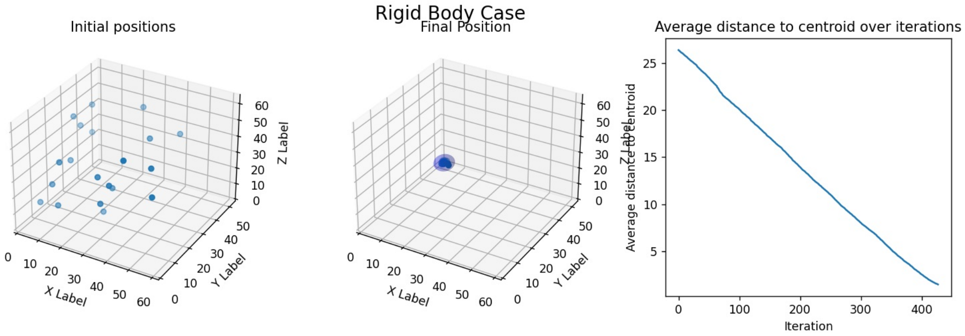


Figure 3.5: Rigid Body Case

During the loop it is checked for any collisions just like in previous examples, but now the distance between the individuals can't be less than two times the radius of each one.

3.2 Social Foraging

This section focuses on modelling different resource profiles to incorporate the effect of the environment within the potential functions framework.

As mentioned in the previous section, the potential function will be

$$J_{foraging}(x) = \sum_{i=1}^{N} \sigma(x_i) + \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} [J_a(\|x_i - x_j\|) - J_r(\|x_i - x_j\|)] \tag{3.17}$$

and the control signal

$$u_i = -\nabla_{x_i}\sigma(x_i) - \sum_{j=1, j\neq i}^{N} [\nabla_{x_i} J_a(\|x_i - x_j\|) - \nabla_{x_i} J_r(\|x_i - x_j\|)] \tag{3.18}$$

Different resource profiles will be discussed such as Plane resource profiles, Quadratic resource profiles and Gaussian resource profiles.

3.2.1 Plane Resource Profile

In this section it is assumed that the resource profile function is described by a plane equation of the form:

$$\sigma(y) = a_\sigma^T y + b_\sigma \tag{3.19}$$

where $a_\sigma \in \Re^n$ and $b_\sigma \in \Re$. The gradient of the profile is given by

$$\nabla_y \sigma(y) = a_\sigma \tag{3.20}$$

The bound on the swarm will be

$$\varepsilon = \varepsilon_p = \frac{b(N-1)}{aN} < \frac{b}{a} \tag{3.21}$$

and the centroid motion will be

$$\dot{\bar{x}}(t) = -a_\sigma \tag{3.22}$$

So the new control signal will be

$$u_i = -a_\sigma - \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} (x_i - x_j) \left[a - b\exp\left(-\frac{(\|x_i - x_j\| - 2\eta)^2}{c} \right) \right] \tag{3.23}$$

To test this algorithm it was only necessary to update the control function used in aggregation for the equation in 3.23. Several simulations were made to understand the limitations of the model.

Figure 3.6 and 3.9 showcase one of the simulations using the following parameters: sampling time = 0.016, a=0.01, N = 30, $\varepsilon = 12$, $\delta = 8$ and $\eta = 0.5$. The minimum radius will be $r_{min} = 0.5\sqrt[3]{12}$, $b = a\varepsilon = 0.12$, $c = \frac{\delta^2}{\ln(\frac{b}{a})} = 25.76$ and $A_\sigma = [0.5, 0, 0]$. The initial positions were randomized for values between [-50, 50], the stopping criteria was any collision at any given time or 12000 iterations.

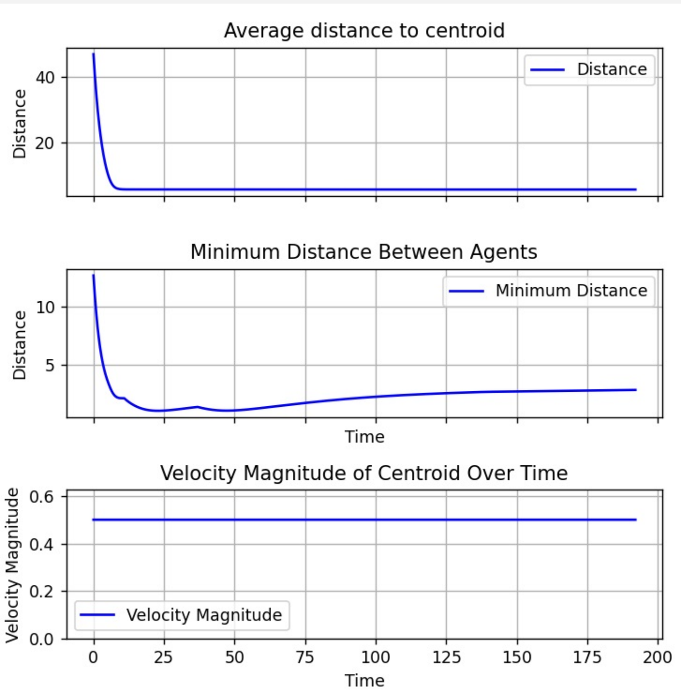


Figure 3.6: Plane Resource Profile

Figure 3.6 is composed by three plots, one for the average distance to the (moving) centroid, one for minimum distance between two agents at any given moment and, finally, the third plot shows the velocity magnitude of the swarm centroid over time. It is clear that the controller converges and is able to stabilize at a satisfactory average distance from the centroid, without collisions and with the expected velocity magnitude. The final velocity was of 0.4999 and the expected centroid velocity was 0.5. The final average distance to the centroid was 5.55 and the minimum distance between two agents was 1.056, meaning that at a certain point in time, two drones were at a distance of 0.056 from colliding. It is important to choose a(attraction term) taking into consideration A_σ, or vice-versa, to avoid collisions it is advisable that their ratio is no more than 10^2.

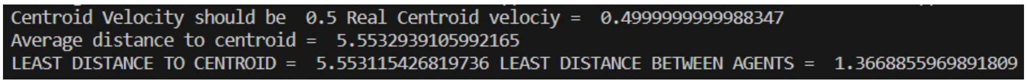

```
Centroid Velocity should be  0.5 Real Centroid velociy =  0.4999999999988347
Average distance to centroid =  5.5532939105992165
LEAST DISTANCE TO CENTROID =  5.553115426819736 LEAST DISTANCE BETWEEN AGENTS =  1.3668855969891809
```

Figure 3.8: Results obtained

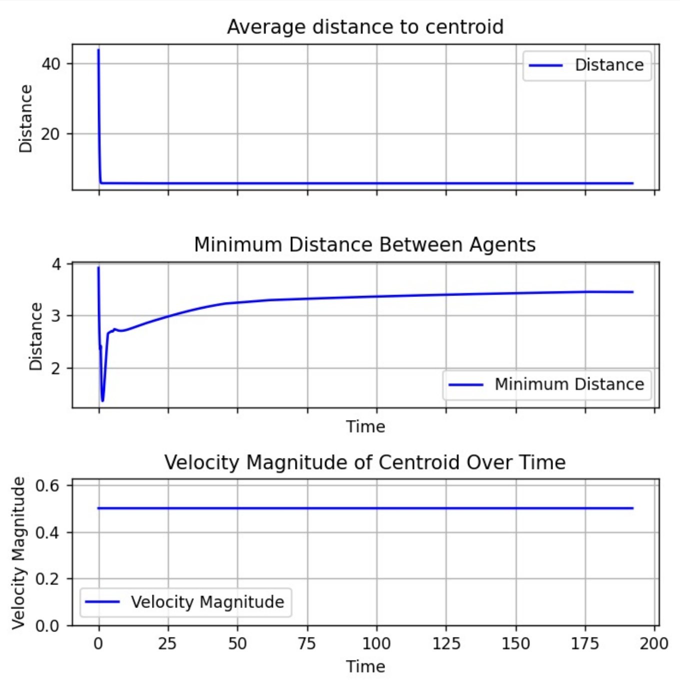


Figure 3.7: Plane Resource profile with larger attraction term

This is proven in figures 3.7 and 3.8 where all the parameters were the same as before, except for the attraction term, which was changed to $a = 0.1$, ten times larger that in the previous simulation. Now the minimum distance between two agents was 1.367 as opposed to 1.056 in the previous example.

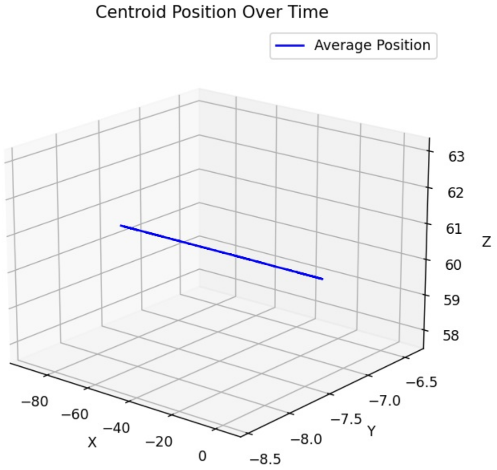


Figure 3.9: Centroid trajectory over time

Figure 3.9 shows the centroid trajectory over time, the centroid is moving along the x axis with negative velocity just as expected.

3.2.2 Quadratic Resource Profile

In this section, quadratic resource profiles given by

$$\sigma(y) = \frac{A_\sigma}{2}\|y - c_\sigma\|^2 + b_\sigma, \tag{3.24}$$

will be considered. Its gradient at a point $y \in \Re^n$ is:

$$\nabla_y \sigma(y) = A_\sigma(y - c_\sigma) \tag{3.25}$$

So now the new control function will be

$$u_i = -A_\sigma(y - c_\sigma) - \sum_{i=1}^{N-1}\sum_{j=i+1}^{N}(x_i - x_j)\left[a - b\exp\left(-\frac{(\|x_i - x_j\| - 2\eta)^2}{c}\right)\right] \tag{3.26}$$

Two situations will be tested, one where $A_\sigma > 0$ and other where $A_\sigma < 0$. In the first situation it is expected that the centroid of the swarm will move towards c_σ and, in the second situation, the swarm is supposed to diverge from c_σ to infinity.

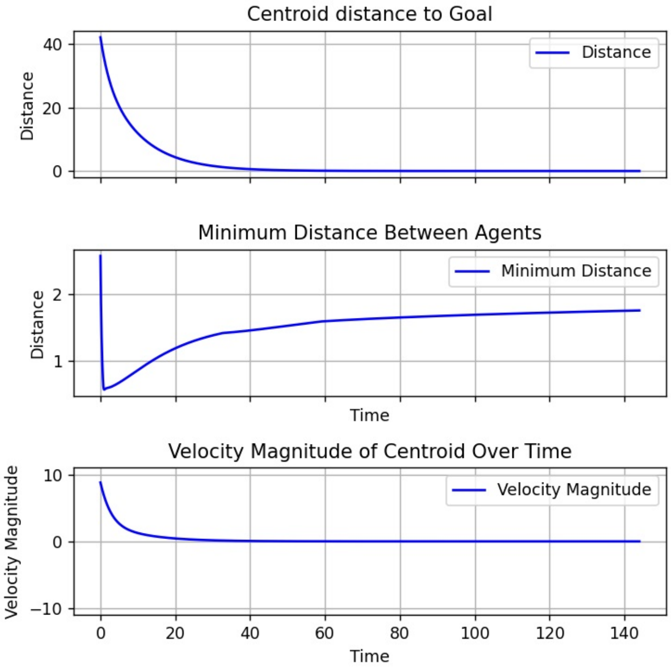


Figure 3.10: Quadratic Resource Profile

In the first simulation, the attraction term was $a = 0.1$, the threshold distance $\varepsilon = 8$, the equilibrium distance was $\delta = 5$, the drone radius $\eta = 0.1$, the number of drones was $N = 30$ and the sampling time was $t_s = 0.016$. For the resource profile, the parameters were $A_{sigma} = [0.1, 0.1, 0.3]$ and $c_\sigma = [22, 22, 22]^T$. The positions of each agent were randomized for values between [-50, 50]. The simulation only stops after 9000 iterations or if there is a collision.

Figure 3.10 displays the centroid distance to c_σ, the minimum distance between agents and the velocity magnitude of the centroid. As seen in the first plot, the centroid eventually reaches its goal, c_σ, and stays around it for $t \to \infty$.

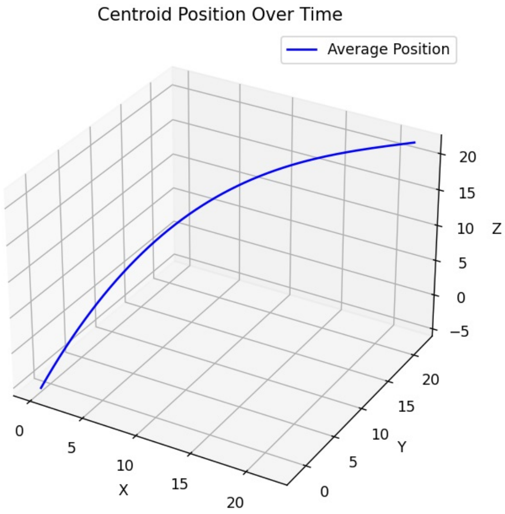


Figure 3.11: Quadratic Resource Profile: Centroid Trajectory over time

In this example, A_σ was defined as a 3D array, with a different scalar value for the z axis. This is the reason why the drone centroid follows the trajectory shown in figure 3.11, if instead a scalar value for A_σ was chosen, the centroid would follow a straight line trajectory to c_σ

In the next example, the algorithm starts with the same parameters as in the previous simulation, but when the distance $||\bar{x} - c_\sigma|| <= 3$, A_σ is assigned -0.1, in order to simulate the repulsive nature of a swarm in the presence of unfavorable situations.

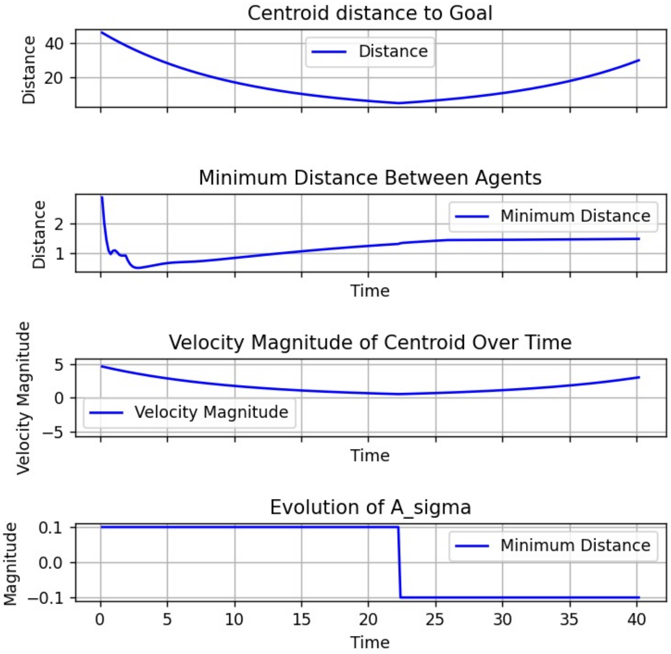


Figure 3.12: Quadratic Resource Profile: Centroid Trajectory over time

In figure 3.12, it is possible to notice that when A_σ is changed, the centroid starts increase its distance to c_σ.

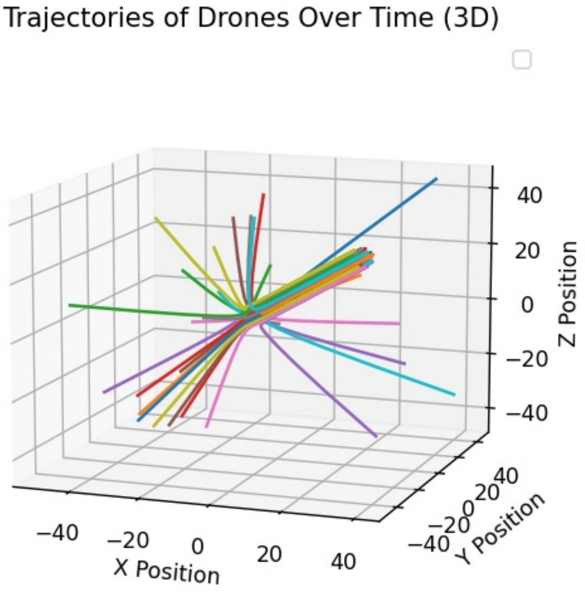


Figure 3.13: Drones' trajectories over time

3.2.2.1 Quadratic Sum Resource Profile

In this section, instead of having one single point of interest, it is possible to define M points of attraction and repulsion. The resource profile function used was:

$$\sigma(y) = \sum_{i=1}^{M} \frac{A_{\sigma i}}{2} \|y - c_{\sigma i}\|^2 + b_\sigma \tag{3.27}$$

where $A_{\sigma i} \in \Re$, and $c_{\sigma i} \in \Re^n$ for all $i = 1, ..., M$, and $b_\sigma \in \Re$. Its gradient is given by:

$$\nabla_y \sigma(y) = \sum_{i=1}^{M} A_{\sigma i}(y - c_{\sigma i}). \tag{3.28}$$

So the control function will be

$$u_i = -\sum_{k=1}^{M} A_{\sigma k}(y - c_{\sigma k}) - \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} \left[\frac{a}{2}\|x_i - x_j\| + \frac{bc}{2} \exp\left(-\frac{\|x_i - x_j\|^2 - 4\eta^2}{c} \right) \right] \tag{3.29}$$

Let A_σ be equal to:

$$A_\sigma = \sum_{i=1}^{M} A_{\sigma i} \tag{3.30}$$

and

$$c_\sigma = \frac{\sum_{i=1}^{M} A_{\sigma i} c_{\sigma i}}{\sum_{i=1}^{M} A_{\sigma i}} \tag{3.31}$$

So now, the swarm centroid will converge or diverge, depending on A_σ, to the new c_σ, shown in equation 3.31.

To test this function, several simulations were made using the parameters shown in table 3.1. The attraction/repulsion parameters were the same for almost all the simulations, whereas different resource profile parameters were chosen for all the simulations. The main focus was trying to understand how the swarm would react to different magnitude values of attraction or repulsion for each c_{σ_i}.

The trajectory of each drone in the first test is shown in figure 3.14, it is possible to notice that the swarm is able to avoid the repulsive point in the environment and converge to the goal. In figure 3.16, it i possible to verify that the distance to the goal converges to zero smoothly.

	TEST 1	TEST 2	TEST 3	TEST 4
ε	8	8	8	8
δ	5	5	5	5
a	0.1	0.1	0.1	0.5
b	0.8	0.8	0.8	4
N	30	30	30	30
η	0.1	0.1	0.1	0.3
M	3	3	3	10
c_σ	$c_\sigma[i][j] \in [-50;50]$	$c_\sigma[i][j] \in [-50;50]$	$c_\sigma[i][j] \in [-50;50]$	$c_\sigma[i][j] \in [-500;500]$
A_σ	$A_\sigma[i] \in [-1;1]$	$A_\sigma[i] \in [-0.5;1.5]$	$A_\sigma[i] \in [-1.5;0.5]$	$A_\sigma[i] \in [-5;5]$
x_{jk}	$x_{jk} \in [-50,50]$	$x_{jk} \in [-50,50]$	$x_{jk} \in [-50,50]$	$x_{jk} \in [-500,500]$
i	∈(1,..., M)			
j	∈(1,..., N)			
k	∈(1,..., 3)			

Table 3.1: Simulation parameters for each test

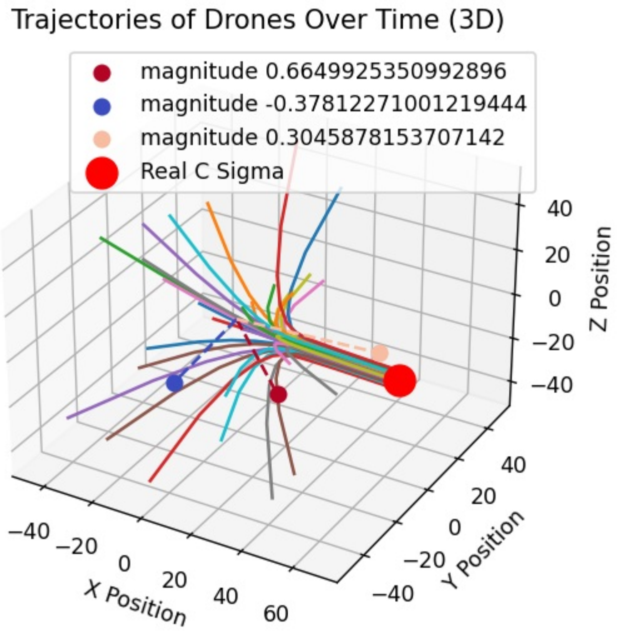
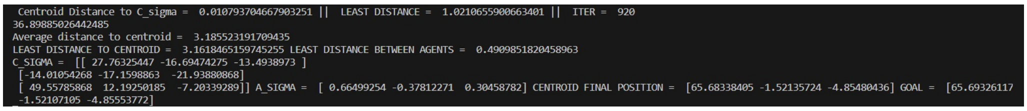


Figure 3.14: Drones' trajectories over time



```
 Centroid Distance to C_sigma =  0.010793704667903251 ||  LEAST DISTANCE =  1.0210655900663401 ||  ITER =  920
36.89885026442485
Average distance to centroid =  3.185523191709435
LEAST DISTANCE TO CENTROID =  3.1618465159745255 LEAST DISTANCE BETWEEN AGENTS =  0.4909851820458963
C_SIGMA =  [[ 27.76325447 -16.69474275 -13.4938973 ]
 [-14.01054268 -17.1598863  -21.93880868]
 [ 49.55785868  12.19250185  -7.20339289]] A_SIGMA =  [ 0.66499254 -0.37812271  0.30458782] CENTROID FINAL POSITION =  [65.68338405 -1.52135724 -4.85480436] GOAL =  [65.69326117
 -1.52107105 -4.85553772]
```

Figure 3.15: Simulation Final Values

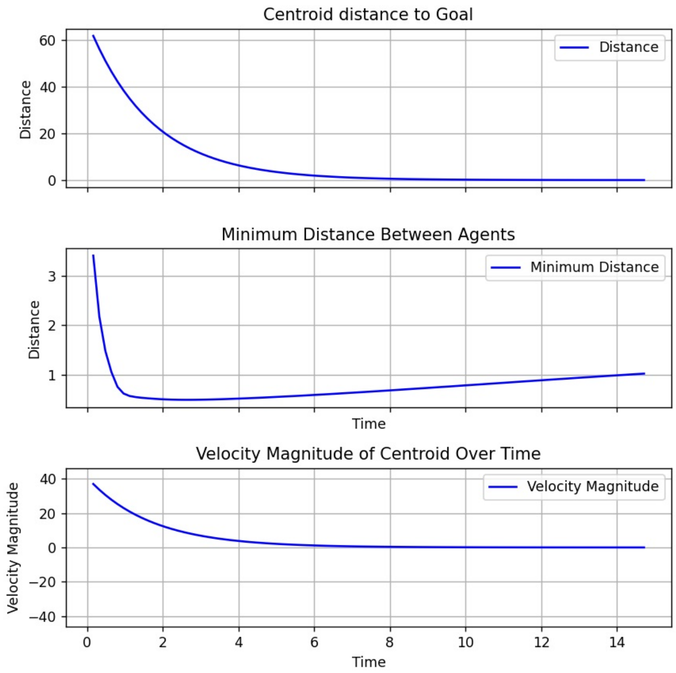


Figure 3.16: Simulation parameters plotted

In the second simulation, a_σ was chosen such that the probability of having more attractive points is higher, in order to test how the swarm reacts to multiple attractive points in the environment. It is possible to observe, in figure 3.17, the swarm aggregating around the new desired point.

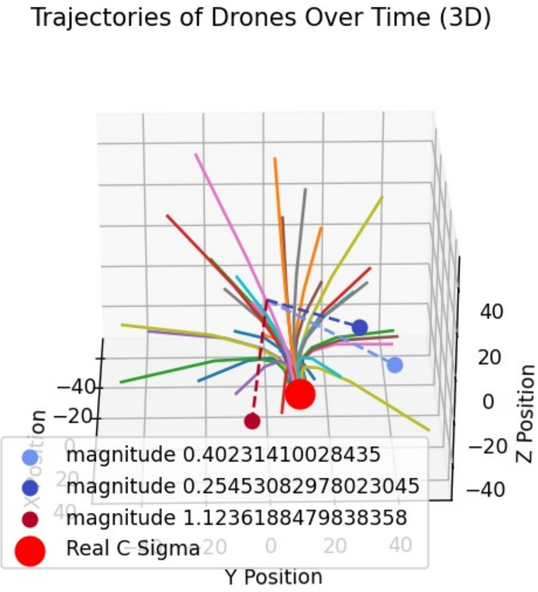
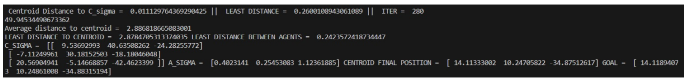


Figure 3.17: Drones' trajectories over time



```
 Centroid Distance to C_sigma =  0.011129764369290425 ||  LEAST DISTANCE =  0.2600108943061089 ||  ITER =  280
49.94534490673362
Average distance to centroid =  2.886818665083001
LEAST DISTANCE TO CENTROID =  2.8784705313374035 LEAST DISTANCE BETWEEN AGENTS =  0.2423572418734447
C_SIGMA =  [[  9.53692993  40.63508262 -24.28255772]
 [ -7.11249961  30.18152503 -18.18046048]
 [ 20.56904941  -5.14668857 -42.4623399 ]] A_SIGMA =  [0.4023141  0.25453083 1.12361885] CENTROID FINAL POSITION =  [ 14.11333002  10.24705822 -34.87512617] GOAL =  [ 14.1189407
3  10.24861008 -34.88315194]
```

Figure 3.18: Important simulation variables

In test 3, the main focus was analysing the swarm behaviour in the presence of more repulsive points than attractive points, and the results can be seen in figure 3.19. Since the repulsion forces dominate the environment, the swarm will diverge to infinity in the opposite direction of the c_σ calculated using equation in 3.31.

In figure 3.13, it is readily observed how the distance to the goal and the velocity of the centroid will continue increasing to infinity.

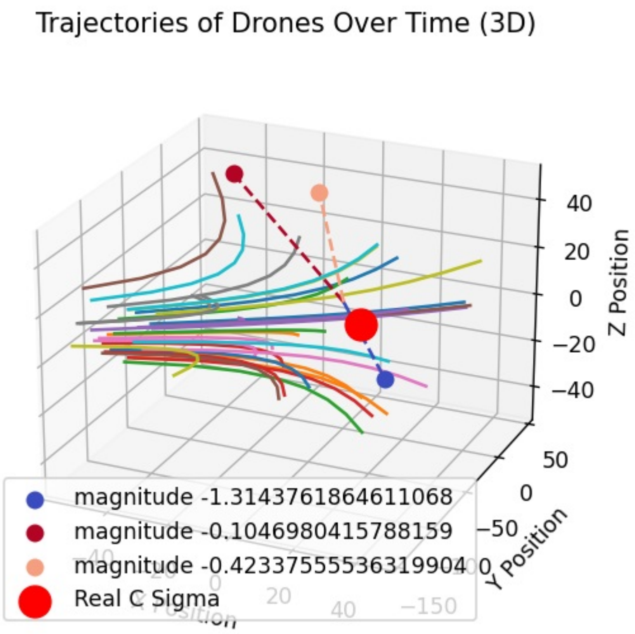


Figure 3.19: Drones' trajectories over time

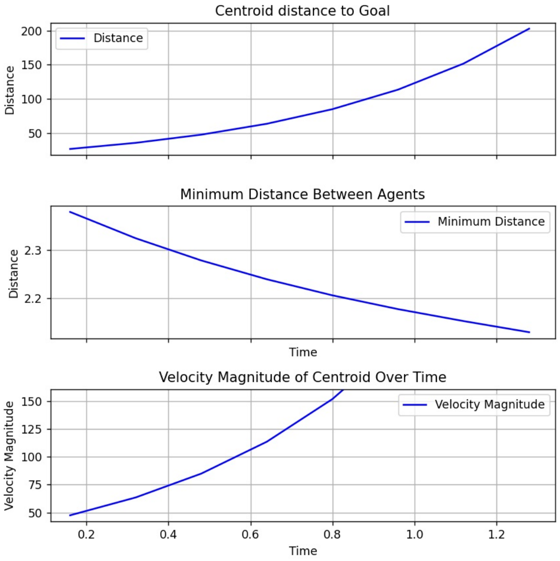


Figure 3.20: Drones' trajectories over time

Finally, in test 4 the environment is overloaded with more attraction/repulsion points, the results are shown in figure 3.21.

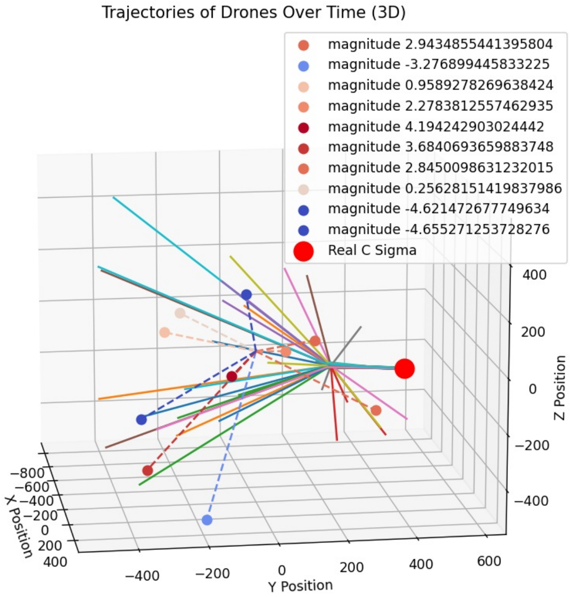


Figure 3.21: Drones' trajectories over time

The drones were able to converge to the goal with an extremely quick response, partially due to the increase in the attraction term (a). Figure 3.21 and figure 3.22 show the drones' trajectory, as well as the centroid's distance to the goal and other important variables.

Quadratic resource profiles are relatively simple to implement, and can be extremely useful to model static and non static attraction/repulstion forces in the surrounding environment of the swarm.

3.3 Motion Control Problems in Webots

After developing and testing the logic behind the algortihms mentioned above, it was necessary to adapt the code for Webots. Since the position update was made using an approximation on the loop and now it will be given by the simulator himself. Moreover it will be necessary to have some kind of communication, since the drones do not know the other drones' position beforehand.

There were to possible ways communicate their positions:

1. All drones communicate with a centralized program whose task is to calculate all the control inputs for all the drones and communicate them to each drone.

2. All drones communicate with each other, there is not centralized program and all the drones make their own calculations on the new control inputs for themselves.

In this implementation, the second option was chosen, mainly because the degree of difficulty between both implementations is not large and the second implementation is closer to a decentralized approach, making the application more realistic. This also takes advantage of the fact that the

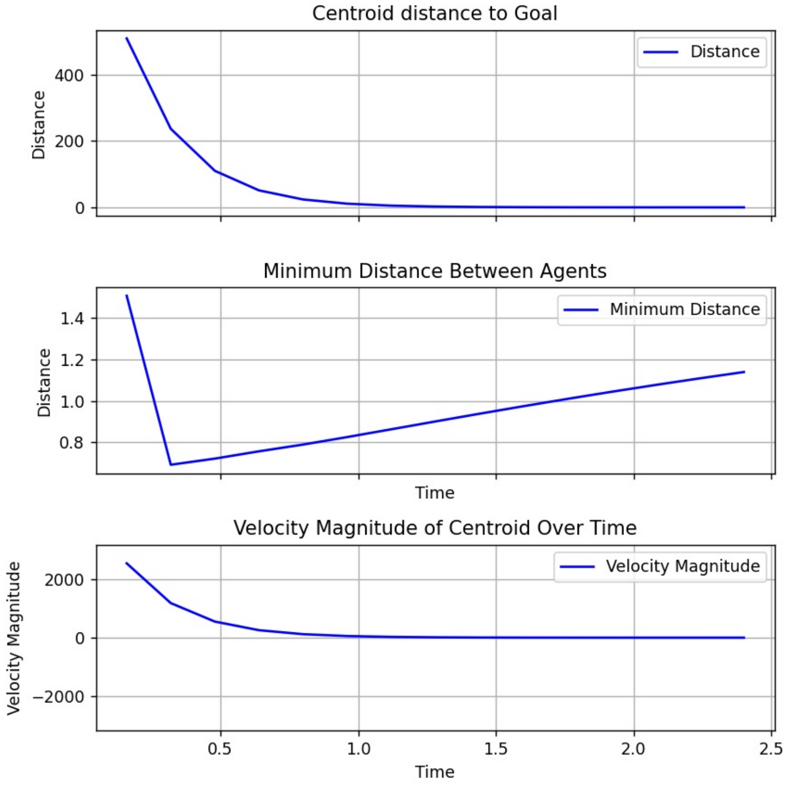


Figure 3.22: Drones' trajectories over time

controller program is incorporated inside the drone, so every drone has an individual controller working in parallel with the other drones, which is already close to the goal in mind.

To the Bitcraze implementation, the following elements were added:

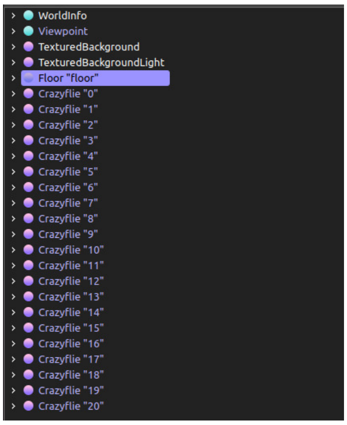
1. N instances of the drone in the world file (.wbt) with random initial positions, since the Bitcraze's controller was for a single drone. The names were instatiated as strings containing numbers from 1 to N, making it easier to identify them.

2. An emitter and receiver device were added to Crazyflie's PROTO file so that the drones could communicate with each other.

The new Scene Tree will look like this:

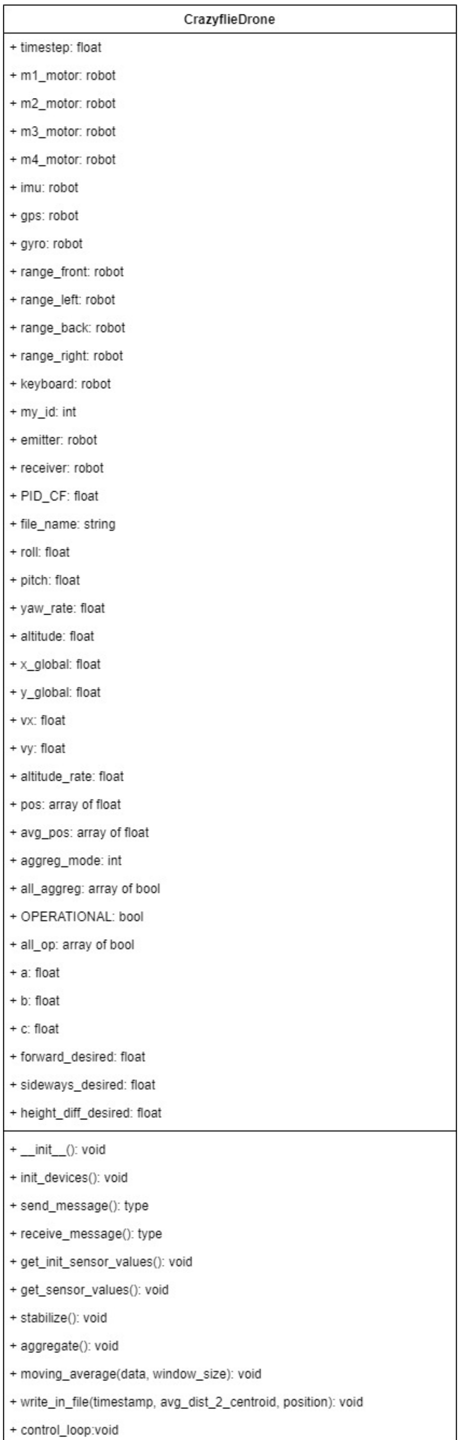
As it is observed from figure 3.23 the only changes are in the number of drones in the simulator. Apart from that, and as mentioned above, two devices were added to the robot node, emitter and receiver. Figure 3.24 shows these two devices added to the Scene Tree.

In this phase, a class for the drone was created containing all the methods necessary.



WorldInfo
Viewpoint
TexturedBackground
TexturedBackgroundLight
Floor "floor"
Crazyflie "0"
Crazyflie "1"
Crazyflie "2"
Crazyflie "3"
Crazyflie "4"
Crazyflie "5"
Crazyflie "6"
Crazyflie "7"
Crazyflie "8"
Crazyflie "9"
Crazyflie "10"
Crazyflie "11"
Crazyflie "12"
Crazyflie "13"
Crazyflie "14"
Crazyflie "15"
Crazyflie "16"
Crazyflie "17"
Crazyflie "18"
Crazyflie "19"
Crazyflie "20"

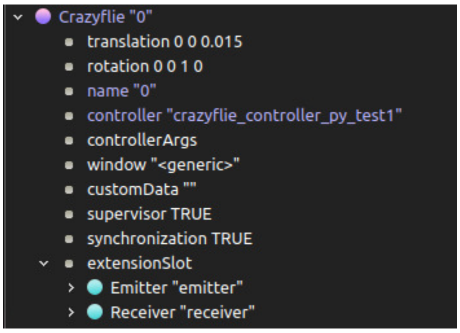
Figure 3.23: New Scene Tree



CrazyflieDrone

+ timestep: float
+ m1_motor: robot
+ m2_motor: robot
+ m3_motor: robot
+ m4_motor: robot
+ imu: robot
+ gps: robot
+ gyro: robot
+ range_front: robot
+ range_left: robot
+ range_back: robot
+ range_right: robot
+ keyboard: robot
+ my_id: int
+ emitter: robot
+ receiver: robot
+ PID_CF: float
+ file_name: string
+ roll: float
+ pitch: float
+ yaw_rate: float
+ altitude: float
+ x_global: float
+ y_global: float
+ vx: float
+ vy: float
+ altitude_rate: float
+ pos: array of float
+ avg_pos: array of float
+ aggreg_mode: int
+ all_aggreg: array of bool
+ OPERATIONAL: bool
+ all_op: array of bool
+ a: float
+ b: float
+ c: float
+ forward_desired: float
+ sideways_desired: float
+ height_diff_desired: float

+ __init__(): void
+ init_devices(): void
+ send_message(): type
+ receive_message(): type
+ get_init_sensor_values(): void
+ get_sensor_values(): void
+ stabilize(): void
+ aggregate(): void
+ moving_average(data, window_size): void
+ write_in_file(timestamp, avg_dist_2_centroid, position): void
+ control_loop:void



```
Crazyflie "0"
    translation 0 0 0.015
    rotation 0 0 1 0
    name "0"
    controller "crazyflie_controller_py_test1"
    controllerArgs
    window "<generic>"
    customData ""
    supervisor TRUE
    synchronization TRUE
    extensionSlot
        Emitter "emitter"
        Receiver "receiver"
```

Figure 3.24: Drone node in the Scene Tree

Figure 3.25 shows all the variables and methods implemented for the aggregation algorithms. All the robot type variables are from the webots simulator and are necessary to do the low level control it is the only way to keep track of the drone's positions. Furthermore it is necessary to calculate the velocities of the individuals relative to the inertial referential using the euler angles, but the reason they are needed as public variables is to update the low level PID controller developed by Bitcraze. The aggregation mode is chosen by the user when running the simulation, one can choose the values (1, 2 or 3) depending on the type of aggregation method desired (Linear Attraction and Bounded Repulsion, Linearly Bounded from Below Attraction and Unbounded Repulsion, Almost Constant Attraction and Unbounded Repulsion), respectively. Besides that, an array containing a boolean for all the drones was created, so that the each drone know when the others are already aggregated, i.e. the distance to the centroid is less than the threshold distance. The other array (all_op) is an initialization array, and guarantees that the drones are all operational and ready to start the motion control algorithms, this is because webots takes sometime to initialize and it is not advisable to start the aggregation procedure before they stabilize. Finally, the variables (a, b, c) are the attraction/repulsion function and (forward_desired, sideways_desired and height_diff_desired) are the control variables sent to the low level PID controller.

The methods implemented were used for:

- init_devices - used to initialize all the devices from the Crazyflie robot

- send_message - used to send data between drones, the message incorporates their id, position, operational flag and aggregated flag.

- receive_message - used to receive messages from other drones, no particular communication protocol was made, only a timeout was implemented on the receiver to guarantee it does not stop the loop.

- get_init_sensor_values - this method waits for the webots simulator to give logical values from its sensors, since as mentioned above, the simulator takes some seconds to initialize, especially when using a large number of drones.

- get_sensor_values - used to get the sensor values during the normal procedures of the program.
- stabilize - waits for all the drones to stabilize around their initial positions and the changes the operational flag to true.
- (aggregation, social foraging) loop
- moving_average - used as an auxiliary method in the stabilizing procedure.
- write_in_file - used to write in a file data needed to plot some important variables like the average distance to the centroid and the drones' positions overtime. Also helpful for debugging the code.
- control_loop - as the name states, this is the main loop in charge of deciding which procedure needs to be made next.

The normal routine for this implementation will be:

Aggregation

1. Initialize all devices and variables
2. Get concrete gps and imu values from the simulator
3. Guarantee the drones are stable and flying at their initial positions
4. Ask the user which method of aggregation is to be executed
5. Start aggregation method
6. End the aggregation method when all drones are at a distance to the centroid inferior to the threshold distance.

Social Foraging For social foraging, only the linear attraction and unbounded repulsion function was used to implement plane resource profiles, quadratic and quadratic sum resource profiles and gaussian recource profiles.

1. Initialize all devices and variables
2. Get concrete gps and imu values from the simulator
3. Guarantee the drones are stable and flying at their initial positions
4. Ask the user which resource profile he wants
5. Assign a_σ or a_{sigma} and c_σ if mode (plane, quadratic or quadratic sum)

6. Start social foraging

7. End the loop after N iterations, decided by the user.

8. plot parameters

In the plane resource profile, the parameters plotted were the average distance to the cetroid, the minimum distance between agents and the centroid's velocity. The centroid's velocity should be equal to $-a_\sigma$.

For the quadratic resource profile, the parameters plotted were the centroid's distance to the goal (c_σ), the minimum distance between agents and the centroid's velocity. Depending on the value of a_σ, the centroid's distance to the goal should converge to zero or diverge to infinity.

For the quadratic sum resource profile, the parameters plotted were the same as in the previous case, the goal was assigned to the most attractive point in the environment, ie: $c_{\sigma_i} : i = argmax_i(a_{\sigma_i})$.

After changing the existing code to this format, it was possible to test the algorithms in webots, as it is shown in the next section.

This chapter focuses on the development of motion control methods, more precisely, aggregation and social foraging methods.

For the aggregation methods, several attraction/repulsion functions were implemented and tested for the cases of linear attraction and bounded repulsion, linearly bounded from below attraction, almost constant attraction and unbounded repulsion and Agents with finite body size.

In the Linear Attraction and Bounded Repulsion Case, we saw how swarm agents can be made to converge within a specific hyper-ball. This can be useful in scenarios where a specific spatial distribution of the swarm is needed. In contrast, the Linearly Bounded from Below Attraction and Unbounded Repulsion case can ensure that the swarm agents maintain a minimum distance from each other while still allowing them to disperse over a larger area.

The Almost Constant Attraction and Unbounded Repulsion algorithm proved to be effective in bringing the swarm members closer to one another, thereby ensuring a compact formation. This can be beneficial in tasks where the swarm needs to navigate through narrow passageways or crowded spaces.

Finally, the case of Agents with Finite Body Size was examined. This algorithm takes into account the physical size of the individual robots, thereby preventing collisions. Any of the previous cases can be modified to consider agents with finite body size. However, for the social foraging methods, the attraction/repulsion function must have linear attraction and bounded repulsion. For this reason, that will be the function used in the Rigid Body Case and in the following sections.

For the social foraging methods, plane resource profiles, quadratic resource profiles and gaussian resource profiles were modelled. As mentioned before, for this methods the attraction/repulsion function with linear attraction and bounded repulsion for agents with finite body size was chosen. Several simulations were made and multiple parameters (average distance to centroid, minimum distance between agents, velocity of the centroid, trajectory,etc) were plotted and analyzed. These methods proved to be extremely relevant when trying to define attraction/repulsion "forces" in the surroundings of the swarm.

After having all the aggregation and social foraging algorithms implemented it was necessary to make them compatible with the simulator chosen, for that a new class for the drone was created which contains all the methods and attributes necessary to read the sensor values from the simulator, communicate between agents of the system and execute the motion control procedures developed.

In conclusion, the choice of the motion control algorithm can greatly influence the behavior of a robotic swarm. By adjusting the parameters of these algorithms, a wide range of behaviors can be achieved. However, further research and experiments are needed to optimize these parameters for different use-cases. Moreover, real-world constraints such as communication delays, sensor noise, and actuator limitations also need to be taken into account when implementing these algorithms on physical robots.

Chapter 4

Simulation and Results

This chapter presents the results in webots for the algorithms developed in the previous chapter.

4.0.1 Aggregation

For the Aggregation methods, the number of drones used was 21, the objective was to use a large number of drones without deteriorating the quality of the simulation by overloading the simulator. The threshold distance ε chosen was 0.5, since the drone's radius is 0.05 and the minimum swarm radius would be approximately 0.14. The initial positions were chosen randomly, with values so that the initial distance to the centroid would be between 10 to 20 times the threshold distance.

4.0.1.1 Linear Attraction and Bounded Repulsion Case

For the linear attraction and Bounded Repulsion Case the parameter a was set to 0.015, and parameter b and c were calculated using the formulas:

$$b = a\varepsilon \tag{4.1}$$

$$c = (\lambda^2)\ln\frac{b}{a}, c > 0 \tag{4.2}$$

$$c = \frac{b}{100}, \frac{b}{a} < 0 \tag{4.3}$$

Because the error e_i is bounded by $\frac{b}{a}$, then we guarantee that the distance of all drones to the centroid will eventually be less than 0.5.

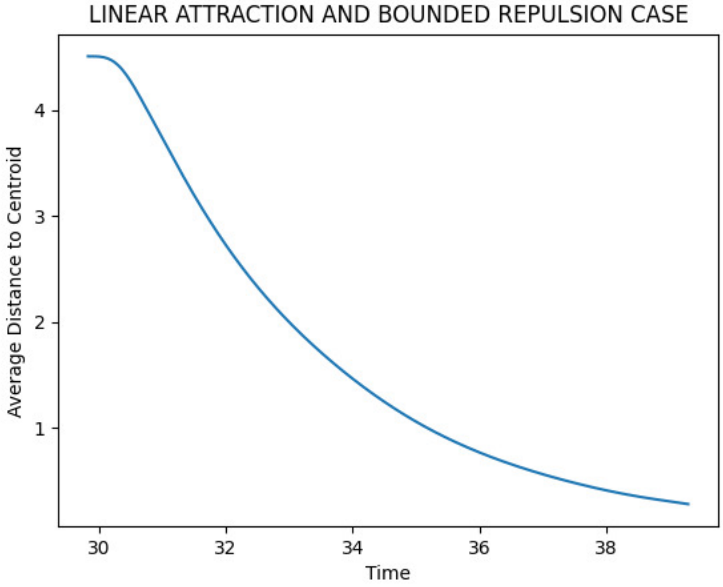


Figure 4.1: Average Distance to Centroid

As is is shown in figure 4.1, the average distance to the centroid is smoothly decreasing until all drones are at a distance smaller than the threshold distance.

Furthermore, from the file mentioned in the previous section used for plotting, a 3D representation of the positions of the drones was made for a better understanding of their behaviour.

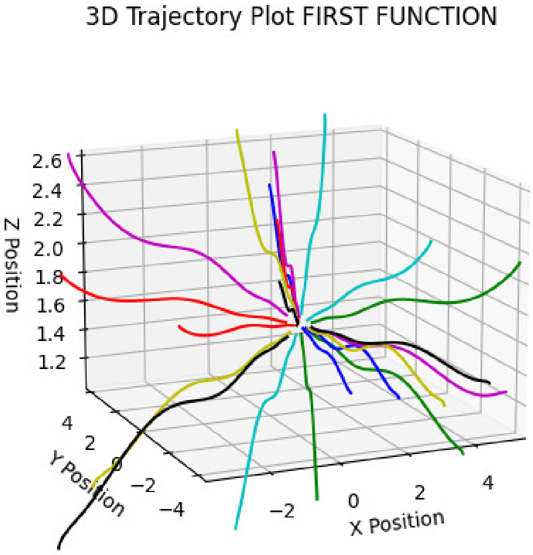


Figure 4.2: Drones' Positions overtime

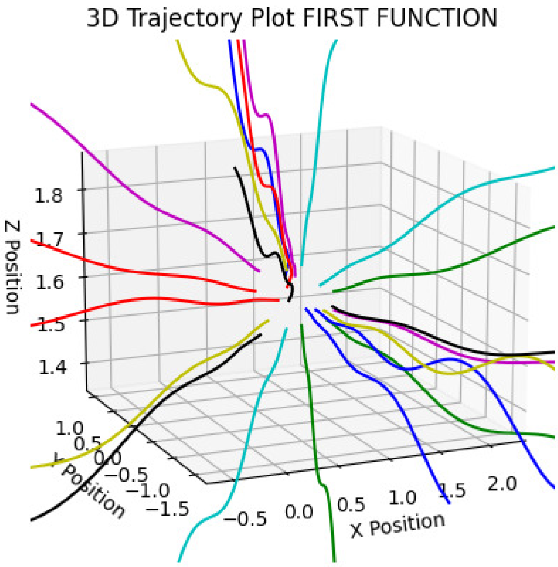


Figure 4.3: Drones' Positions overtime zoomed

In figure 4.2, it is possible to observe all the drones converging to the centroid position. Although it may look like it from the plot, there were no collisions and all the drones aggregated and stopped successfully. Figure 4.3 shows their positions zoomed, so it is easier to notice when they stop and that there are no collisions.

It is important to remark that for attraction constants (a) larger than the one used, there were some collisions, probably because since the simulator has a simulation timestep of 32ms, when the attraction term is to large there is not enough time to correct their control inputs before they crash.

This method was the one with the worst results, in several occasions the drones collapsed and started behaving in an erratic way. For Linear Attraction and Bounded repulsion it is advisable to set the attraction term to an extremely low value so the control inputs are not too aggressive.

4.0.1.2 Linearly Bounded from Below Attraction and Unbounded Repulsion

For the Linearly Bounded from Below Attraction and Unbounded Repulsion the parameter a was set equal to 0.02, and the repulsion term was calculated using this formula:

$$b = \frac{2a\varepsilon^2}{N} \tag{4.4}$$

So the error e_i will now be bounded by $\sqrt{\frac{bN}{2a}} \approx 0.5$.

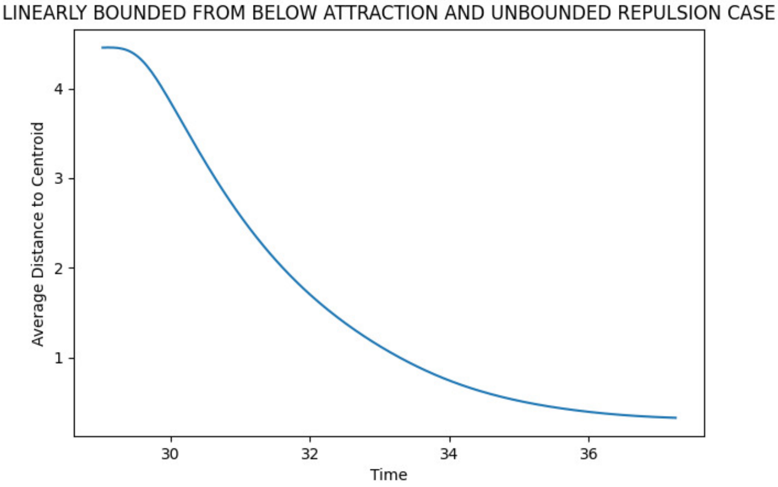


Figure 4.4: Average Distance to Centroid

Figure 4.4 shows the average distance to the centroid with linearly bounded from below attraction and unbounded repulsion. As in the previous example, the distance decreases overtime until reaching the aggregated state.

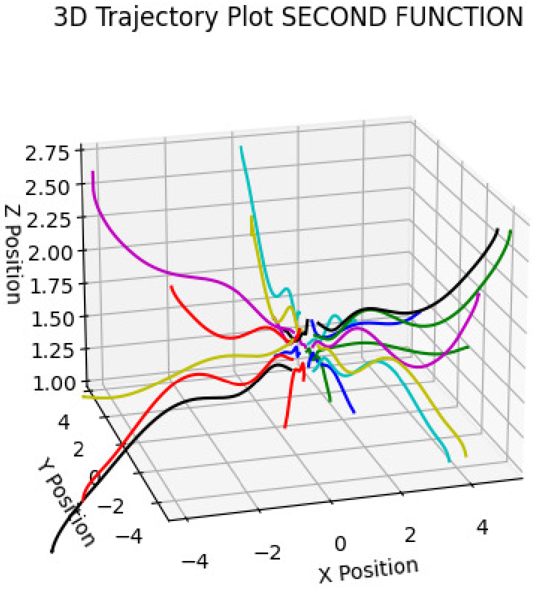


Figure 4.5: Drones’ Positions overtime

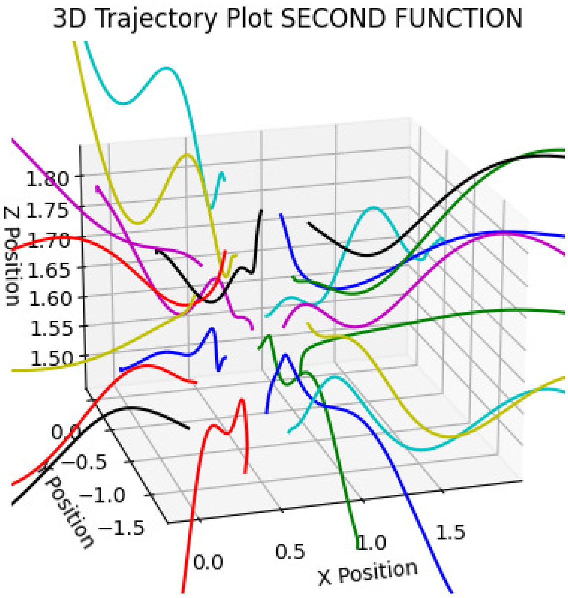


Figure 4.6: Drones' Positions overtime zoomed

Figures 4.5 and 4.6 show the drone's trajectories through time and when they met their convergence criteria. The results for this implementation were as expected, the drones' control was smooth without much erratic behaviour. However, for larger attraction term values, the swarm starts to exhibit some failures.

4.0.1.3 Almost Constant Attraction and Unbounded Repulsion

For the last case, the attraction term was set to 0.025, and the repulsion term was calculated using the formula:

$$b = \frac{\varepsilon a}{N} \tag{4.5}$$

So the error e_i will now be bounded by $\frac{bN}{a} \approx 0.5$.

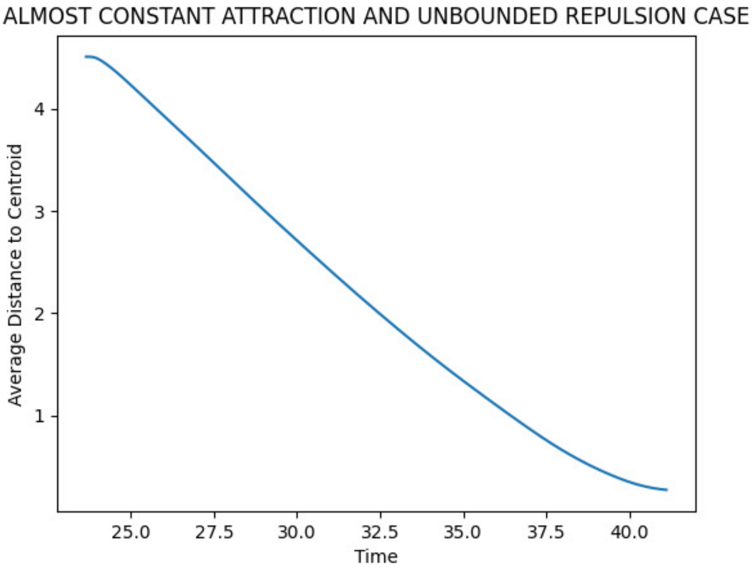


Figure 4.7: Average Distance to Centroid

By looking at figure 4.7, it is readily noticed that this method had the most linear response, which can be useful for applications with a large noise component.

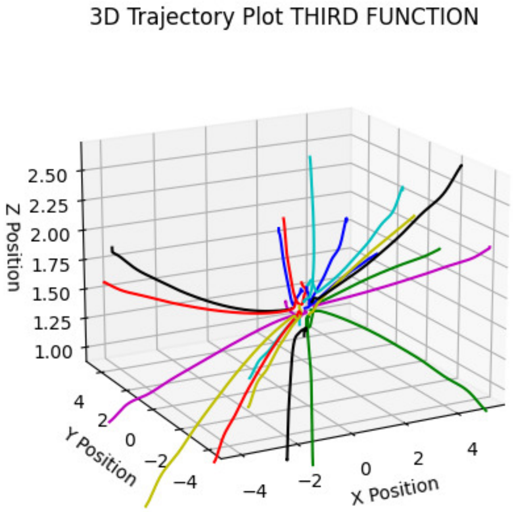


Figure 4.8: Drones' Positions overtime

From figures 4.8 and 4.9 one concludes the aggregation method converged and the drones stopped without any collisions.

To conclude, figure 4.10 depicts the drones in their final state after aggregating. The method used in this example was with almost constant attraction and unbounded repulsion. Since their

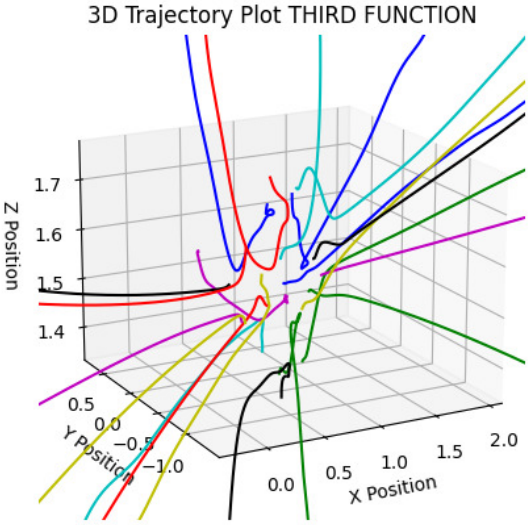


Figure 4.9: Drones' Positions overtime zoomed

configuration is extremely similar in all three cases, it was decided to only show one image of the drones aggregated. Moreover, an image with their initial positions was not taken since it was not illustrative enough and it is hard to identify all the drones due to their size compared to the distances between their initial positions.

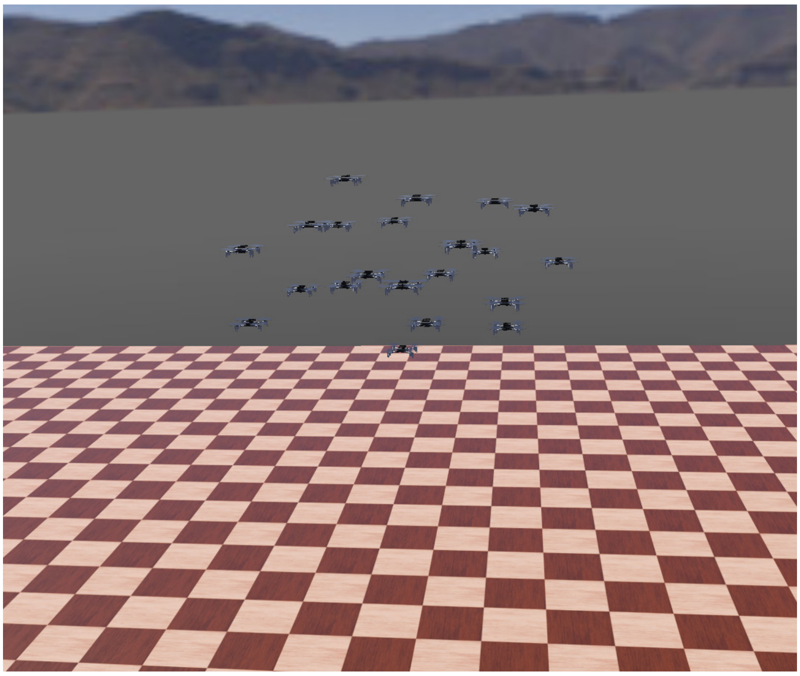


Figure 4.10: Final Position in Webots

4.0.2 Social Foraging

In social foraging, a resource profile function is added to the attraction/repulsion function previously developed. In this section, the results of the simulations for plane resource profiles, quadratic resource profiles and quadratic sum resource profiles are presented. Table 4.1 shows the parameters and resource profile type selected for each simulation. Two tests were made for plane resource profiles, two for quadratic resource profiles and three for quadratic sum resource profiles. In the first test, a scalar value, denoted as A_σ, was selected, while in the second test, A_σ was assigned a 3D array. In the quadratic resource profile tests, all parameters remained consistent between both trials except for the sign of A_σ, which was positive in the first test and negative in the second. In the quadratic sum resource profile tests, it is possible to define multiple attraction and repulsion points, since now $c_\sigma \in \Re^{Mn}$, M represents the number of points to be considered, and $a_\sigma \in \Re^M$ represents each point's attraction or repulsion (depending on the signal) magnitude. For this reason, both M, a_σ and c_σ are changed in between simulations to test the collective response of the drones.

	Test 1	Test 2	Test 3	Test 4	Test 5	Test 6	Test 7
ε	1.5	1.5	1.5	1.5	1.5	1.5	1.5
δ	1.2	1.2	1.2	1.2	1.2	1.2	1.2
a	0.015	0.05	0.05	0.05	0.02	0.02	0.02
b	0.0225	0.075	0.075	0.075	0.05	0.05	0.05
N	21	21	21	21	21	21	21
η	0.03	0.03	0.03	0.03	0.03	0.03	0.03
Type	Plane	Plane	Quad	Quad	Quad sum	Quad sum	Quad sum
M	–	–	–	–	3	3	10
c_σ	-	-	(8,8,8)	(8,8,8)	[-20;20]	[-20;20]	[-20;20]
A_σ	[-0.1 0 0]	-0.1	0.05	0.1	[-0.01;0.1]	[-0.002;0.06]	[-0.4;0.07]
x_{jk}	[-20, 20]	[-20, 20]	[-20, 20]	[-20, 20]	[-20, 20]	[-20, 20]	[-20, 20]

Table 4.1: Simulation parameters for each test

4.0.2.1 Plane Resource Profile

For the plane Resource Profile, two tests were made using both scalar and array a_σ. Figure 4.11 shows the centroid position over time. It is possible to notice the swarm's centroid following the direction of the x-axis. Figure 4.12 demonstrates that, as $t \rightarrow \infty, \dot{\bar{x}} \rightarrow -A_\sigma$

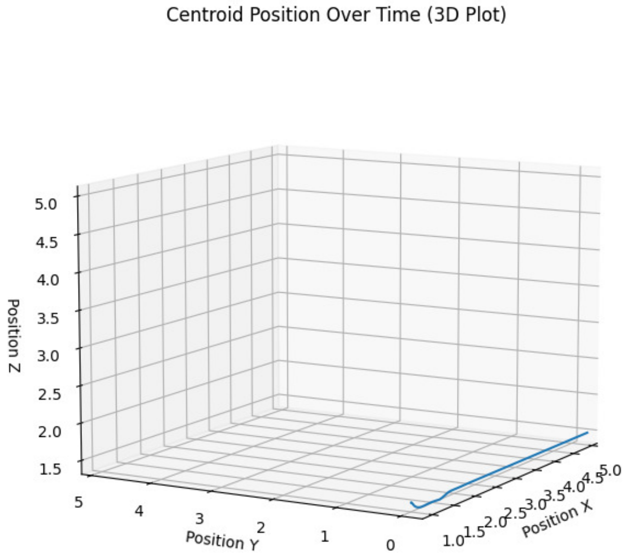


Figure 4.11: Centroid Trajectory

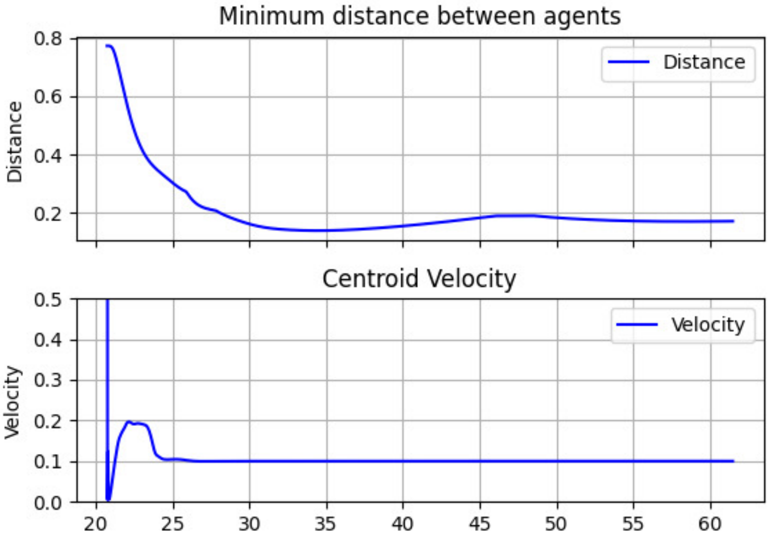


Figure 4.12: Minimum distance between agents and centroid velocity

In the second simulation, a_σ was chosen as a scalar. The results are shown in figure 4.13 and figure 4.14, it is similar to the first simulation, the only difference is now the centroid's trajectory has a component in each axis.

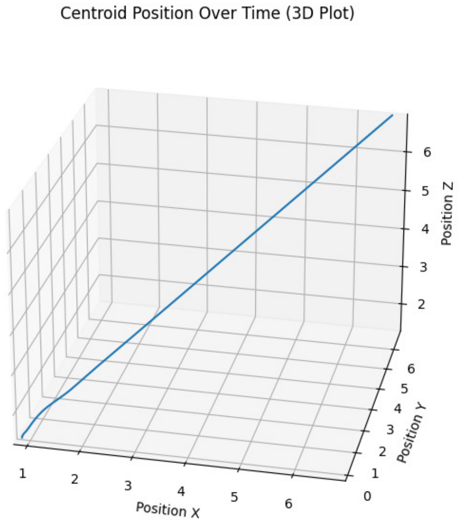


Figure 4.13: Centroid Trajectory

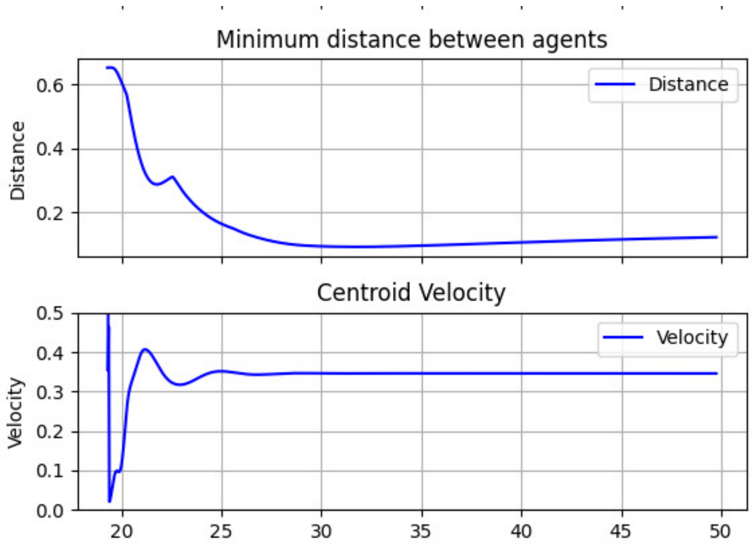


Figure 4.14: Minimum distance between agents and centroid velocity

Plane resource profiles are simple to implement, but offer a good perspective on how the potential function designs works, and how it influences each swarm agent, as well as the swarm's centroid.

4.0.2.2 Quadratic resource profile

For the quadratic resource profile, two simulations were also made. In the first simulation, A_σ is assigned a positive value and the swarm is supposed to aggregate around c_σ. In the second simulation, a_σ is also assigned a positive value, until the distance of the centroid to c_σ reaches a certain threshold. After that, a_σ is set to negative to test the "divergence" behaviour of the swarm, in the presence of a "threatening" resource in their surroundings.

The results of the first simulation are shown in figures 4.15 and 4.16. As stated before, the swarm centroid will eventually reach the goal (c_σ). The centroid's velocity eventually reaches zero as intended.

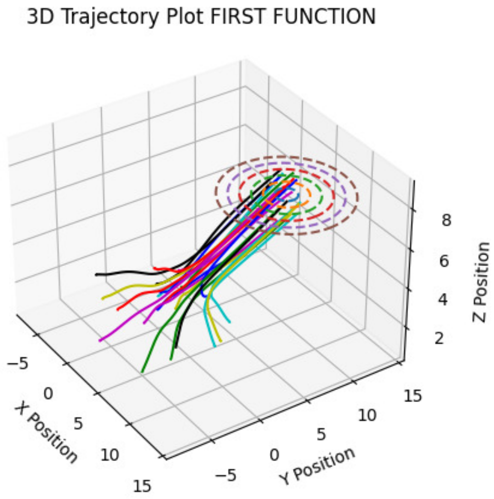


Figure 4.15: Centroid Trajectory

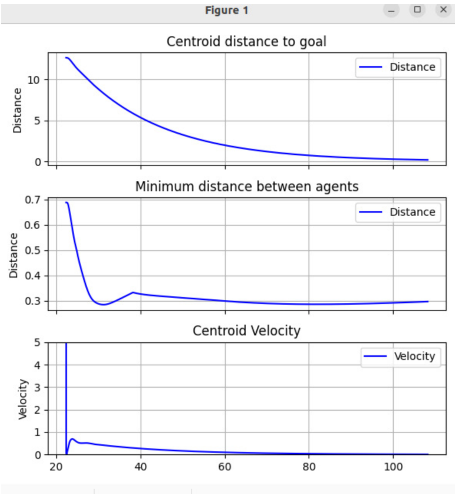


Figure 4.16: Minimum distance between agents and centroid velocity

The results for the second test of quadratic resource profiles are shown in figures 4.17, 4.18 and 4.19.

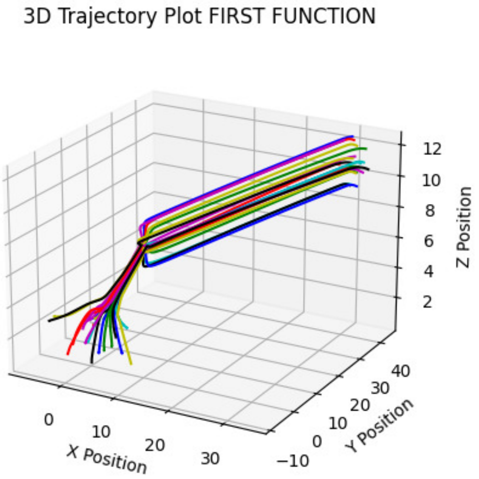


Figure 4.17: Centroid Trajectory

Figure 4.18 shows how the drones, despite dispersing from the repulsive point, they keep the desire to stay together and continue diverging from the goal point in formation.

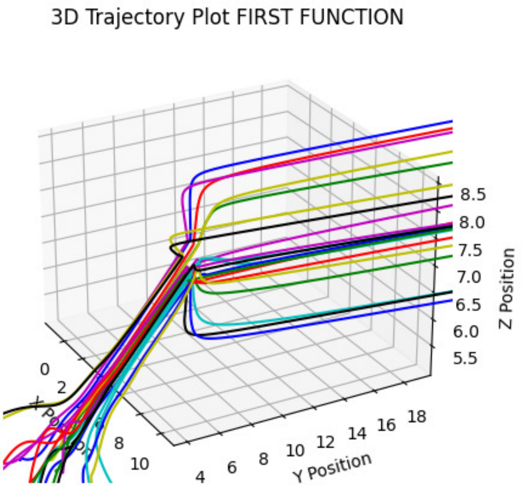


Figure 4.18: Centroid Trajectory

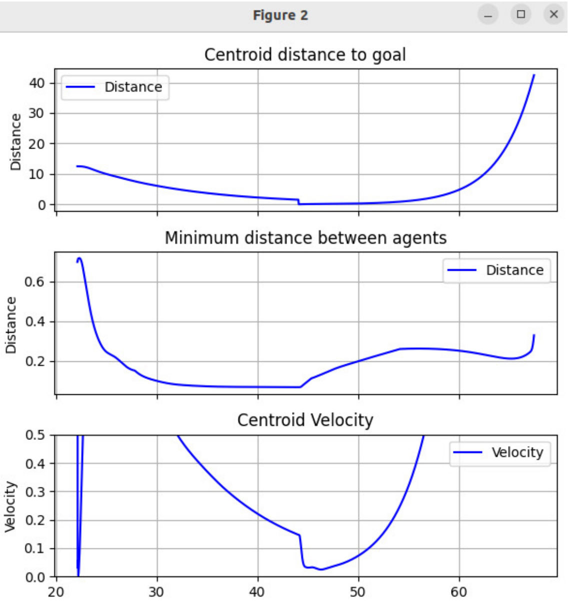


Figure 4.19: Minimum distance between agents and centroid velocity

Quadratic resource profiles can be extremely useful when the goal is to define a region of interest for the swarm. The simulation results satisfied and corroborate the fundamental laws discussed in the previous sections.

4.0.2.3 Quadratic sum resource profile

For the quadratic sum resource profiles, three tests were made, in the first two tests M was set to 3, and in the final test, M was set to 10. The attraction and repulsion magnitudes of the regions were also changed in between simulations. The goal was to see if the swarm would be able to keep a formation and aggregate around the goal point, avoiding the repulsive regions without crashing or producing erratic behaviour.

It is important to notice that in this situation there were many erratic behaviours in some simulations. One of the main reasons is the simulation having a floor, which is as object. When the drones try to avoid repulsive regions, they may end up colliding with the floor. Once they collide, the simulation is no longer helpful to classify the effectiveness of the algorithms developed. For this reason, most repulsive points were chosen with low z-axis values, to make the drones diverge to $z \in \Re^+$ and subsequently, avoid collisions with the floor.

Figures 4.20 and 4.21 show the results obtained for the first simulation.

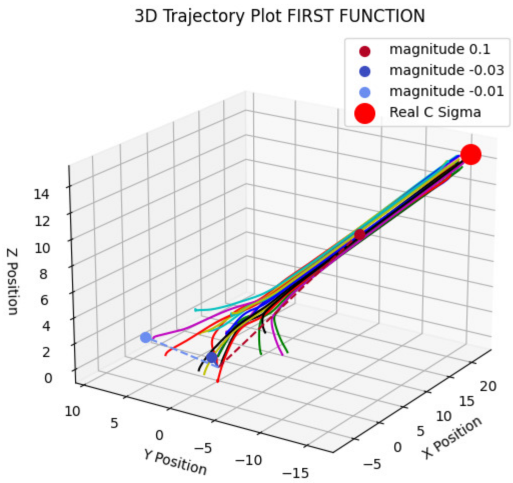


Figure 4.20: Centroid Trajectory

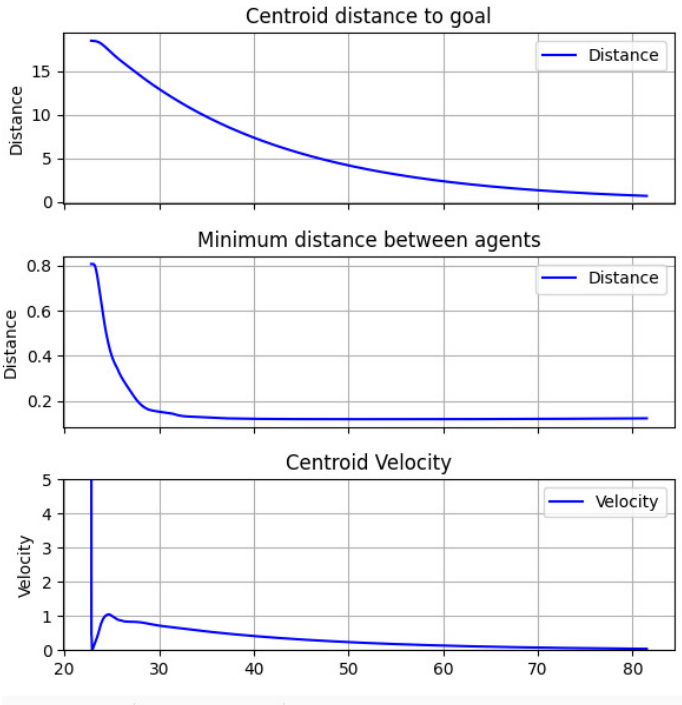


Figure 4.21: Minimum distance between agents and centroid velocity

The swarm is able to converge to the goal point without colliding, all the simulation metrics in figure 4.21 smoothly converge to the expected values.

In the second test, a_σ was assigned so that there would be more repulsive regions than attractive ones. The objective is that the swarm's centroid moves further away from the defined region.

Figure 4.22 shows this behaviour exactly, the drones are moving in the opposite direction of the region.

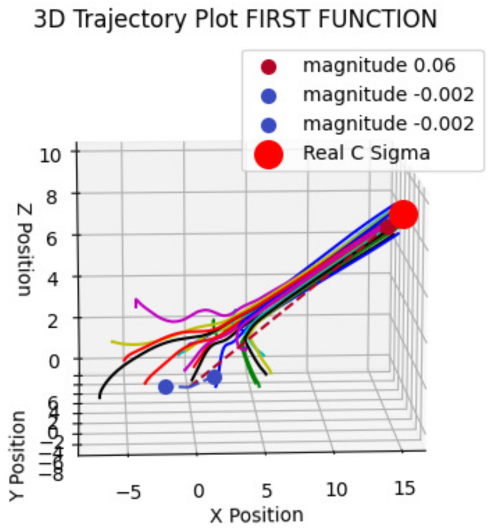


Figure 4.22: Centroid Trajectory

Figure 4.23 shows both the drone's centroid velocity and the centroid's distance to the region increasing indefinitely.

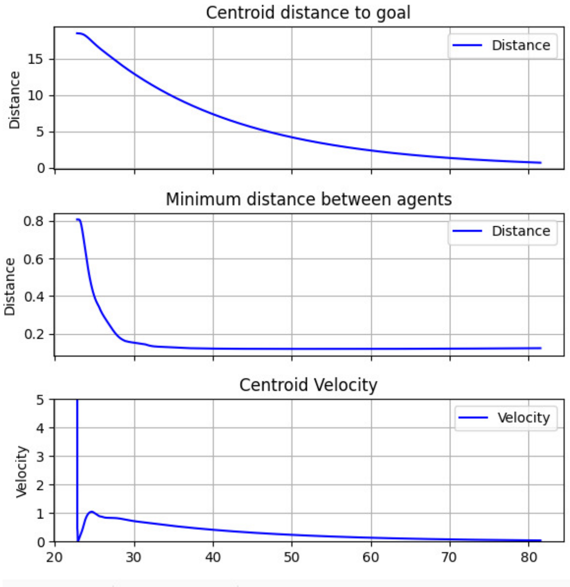


Figure 4.23: Minimum distance between agents and centroid velocity

Finally, for the third test, the number of attractive and repulsive points in the environment was set to ten, to test if the drones were still able to find their way to the region of interest.

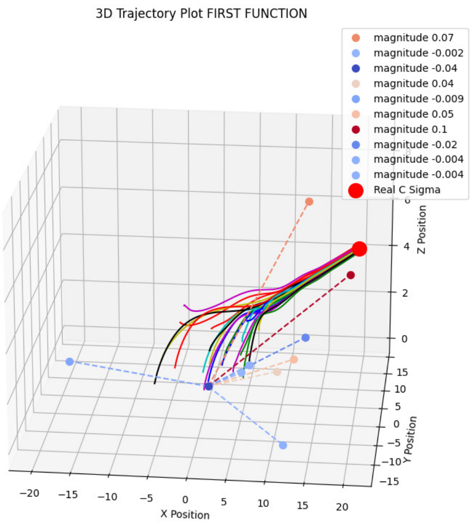


Figure 4.24: Centroid Trajectory

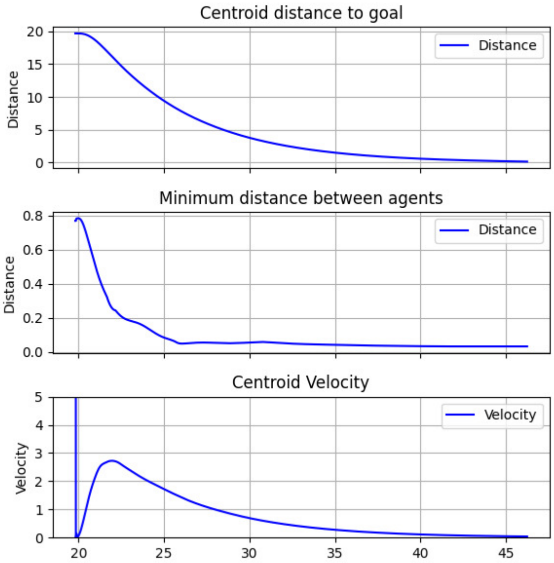


Figure 4.25: Minimum distance between agents and centroid velocity

Although in the simulation example, shown in figures 4.24 and 4.25, the drones were able to reach the region in question, for some cases that was not possible, and further simulations and improvements one the code should be made to achieve a better response in this types of overcrowded simulations.

Webots simulator proved to be extremely useful for testing the developed agorithms. Also, the webots world developed by Bitcraze enabled a faster implementation of the methods in the simulator. For both aggregation and social foraging, the results were satisfatory, and in agreement with the book in the previous sections.[28]

However, this implementation is not "bullet-proof", and does not guarantee the expected behaviour for all the situations possible, as there are still a lot of improvements to be made to the code, the simulator itself and to the low level PID controller developed by Bitcraze and used in this dissertation.

Chapter 5

Conclusion

The goal of this dissertation was implementing and testing different formation algorithms such as aggregation, leader-follower, goal location, and exploration using the Crazyflie *MAVs* developed by bitcraze.

Formation control in micro aerial vehicles is a subject widely studied in the robotics community. Its applications can be found in many different areas ranging from surveillance to farming. Furthermore, the cost of robots is on a downward trend as technology advances, making it easier to conduct real-time physical tests in practical scenarios.

This dissertation presented different architectures found on the literature that study different motion control problems on micro-aerial vehicles such as aggregation, formation, social foraging and obstacle avoidance.

Special emphasis was placed on Gazi and Passino's model, which was thoroughly examined and ultimately selected as the primary method for implementation due to its straightforwardness compared to other examples in the existing literature.

Although the initial goal was to implement aggregation, goal location, and other motion control algorithms, it was only possible to implement some aggregation and social foraging procedures from the model previously referenced.

To test those algorithms, webots simulator was chosen, since it already has a world implementation developed by bitcraze fully operational, unlike Gazebo, which was the second option, that still has some problems related to the low level dynamics of the drone.

The final part of this dissertation demonstrated the results obtained using the webots simulator for the aggregation methods and for the several resource profiles implemented in the social foraging simulations.

The developed algorithms have produced interesting results for both aggregation and social foraging methods, and have demonstrated the suitability of Gazi and Passino's model for real-time swarm applications.

5.1 Future Work

There are several improvements that could be made in this implementation, the communication procedure could be changed to one more reliable, with more data exchange and probably with have less messages between the drones. There were other motion control problems that could have been adressed such as leader-follower and pattern formation. Another important feature would be implementing other motion control models, especially decentralized ones and observe how different their responses would be. Moreover, it would be possible to use the camera and the distance sensor in webots to identify obstacles and implement obstacle avoidance in flock formation without any previous knowledge of the environment. One could improve the world developed by Bitcraze to add complexity to the system and also making it more realistic. Ultimately, the key enhancement for this study would involve testing the existing algorithms on real Crazyflies, as Webots does not entirely replicate the real-world constraints of the environment. For that it would be necessary to develop a localization procedure using LOCO positioning system and a communication procedure using the Bitcraze's libraries. Once these two procedures were concluded, it would only be necessary to integrate them with the aggregation and social foraging methods developed in this work.

References

[1] Sheraz Sadiq. "it's a bird, it's a plane, it's a drone light show!"., 2023. URL: https://www.opb.org/article/2023/07/28/its-a-bird-its-a-plane-its-a-drone-light-show/.

[2] Manuele Brambilla, Eliseo Ferrante, Mauro Birattari, and Marco Dorigo. Swarm robotics: A review from the swarm engineering perspective. *Swarm Intelligence*, 7:1–41, 03 2013. doi:10.1007/s11721-012-0075-2.

[3] bitcraze. Crazyflie, 2019. URL: https://www.bitcraze.io/products/crazyflie-2-1/.

[4] bitcraze. Controllers in the crazyflie, 2019. URL: https://www.bitcraze.io/documentation/repository/crazyflie-firmware/master/functional-areas/sensor-to-control/controllers/.

[5] Gazebo. Gazebo introduction, 2014. URL: https://classic.gazebosim.org/tutorials?cat=guided_b&tut=guided_b1.

[6] Webots. http://www.cyberbotics.com. Open-source Mobile Robot Simulation Software. URL: http://www.cyberbotics.com.

[7] Reza Olfati-Saber. Flocking for multi-agent dynamic systems: Algorithms and theory. *IEEE Transactions on automatic control*, 51(3):401–420, 2006.

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

[9] A Pedro Aguiar. Single and multiple motion control of autonomous robotic vehicles. In *2017 11th International Workshop on Robot Motion and Control (RoMoCo)*, pages 172–184. IEEE, 2017.

[10] João Carlos Santana de Sá. Rviz based simulation for motion control of swarm robotics. 2022.

[11] Carlos Luis and Jérôme Le Ny. Design of a trajectory tracking controller for a nanoquadcopter. *arXiv preprint arXiv:1608.05786*, 2016.

[12] CAI Zhihao, WANG Longhong, ZHAO Jiang, WU Kun, and WANG Yingxun. Virtual target guidance-based distributed model predictive control for formation control of multiple uavs. *Chinese Journal of Aeronautics*, 33(3):1037–1056, 2020.

[13] International Olympic Committee. Spectacular intel drone light show helps bring tokyo 2020 to life, 2021. URL: https://olympics.com/ioc/news/spectacular-intel-drone-light-show-helps-bring-tokyo-2020-to-life-1.

[14] M. Hieslmair. Drone 100: A world record with 100 points, 2012. URL: https://ars.electronica.art/aeblog/en/2016/01/12/drone100/.

[15] droneforbeginners. Top 9 best drone shows that you sure like to watch, 2021. URL: https://droneforbeginners.com/best-drone-shows/.

[16] Ritu Sharma. Indian air force 'bets big' on swarm drone technology to overwhelm, outfox enemy defense systems, 2023. URL: https://www.eurasiantimes.com/the-era-of-swarm-drones-is-here-the-indian-air-force/.

[17] Gabriel Dominguez. U.s. allies and partners critical for pentagon's drone swarm strategy, 2023. URL: https://www.japantimes.co.jp/news/2023/09/13/world/politics/us-military-small-drones-allies/.

[18] V Suresh Kumar, M Sakthivel, Dimitrios A Karras, Shashi Kant Gupta, Syam Machinathu Parambil Gangadharan, and Bhadrappa Haralayya. Drone surveillance in flood affected areas using firefly algorithm. In *2022 International Conference on Knowledge Engineering and Communication Systems (ICKES)*, pages 1–5. IEEE, 2022.

[19] Mauro S Innocente and Paolo Grasso. Self-organising swarms of firefighting drones: Harnessing the power of collective intelligence in decentralised multi-robot systems. *Journal of Computational Science*, 34:80–101, 2019.

[20] Kimberly McGuire. Indoor swarm exploration with pocket drones, 2019.

[21] Marco Carpentiero, Luca Gugliermetti, Marco Sabatini, and Giovanni B Palmerini. A swarm of wheeled and aerial robots for environmental monitoring. In *2017 IEEE 14th international conference on networking, sensing and control (ICNSC)*, pages 90–95. IEEE, 2017.

[22] Xiaohui Li, Hailong Huang, Andrey V Savkin, and Jian Zhang. Robotic herding of farm animals using a network of barking aerial drones. *Drones*, 6(2):29, 2022.

[23] Paolo Tripicchio, Massimo Satler, Giacomo Dabisias, Emanuele Ruffaldi, and Carlo Alberto Avizzano. Towards smart farming and sustainable agriculture with drones. In *2015 international conference on intelligent environments*, pages 140–143. IEEE, 2015.

[24] F. A. Santos. "case study: Amazon já faz entregas por drone"., 2021. URL: https://galp.com/pt/pt/empresas/blog/blog-post/Case-study-Amazon-ja-faz-entregas-por-drone.

[25] Wingcopter. Drone delivery solutions for humanitarian commercial applications, 2023. URL: https://wingcopter.com/solutions.

[26] Liz Hughes. "walmart, wing team for dallas area drone deliveries"., 2023. URL: https://www.iotworldtoday.com/transportation-logistics/walmart-wing-team-for-dallas-area-drone-deliveries.

[27] Minh Anh Nguyen and Minh Hoàng Hà. The parallel drone scheduling traveling salesman problem with collective drones. *Transportation Science*, 2023.

[28] Veysel Gazi and Kevin M Passino. *Swarm stability and optimization*. Springer Science & Business Media, 2011.

[29] Tamás Vicsek and Anna Zafeiris. Collective motion. *Physics reports*, 517(3-4):71–140, 2012.

[30] Jun Yi, Shi Chen, Xiangnan Zhong, Wei Zhou, and Haibo He. Event-triggered globalized dual heuristic programming and its application to networked control systems. *IEEE Transactions on Industrial Informatics*, 15(3):1383–1392, 2018.

[31] Erkin Bahceci, Onur Soysal, and Erol Sahin. A review: Pattern formation and adaptation in multi-robot systems. *Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, Tech. Rep. CMU-RI-TR-03-43*, 2003.

[32] M. Egerstedt and Xiaoming Hu. Formation constrained multi-agent control. *IEEE Transactions on Robotics and Automation*, 17(6):947–951, 2001. doi:10.1109/70.976029.

[33] C. Belta and V. Kumar. Trajectory design for formations of robots by kinetic energy shaping. In *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No.02CH37292)*, volume 3, pages 2593–2598 vol.3, 2002. doi:10.1109/ROBOT.2002.1013622.

[34] bitcraze. Loco positioning system in the crazyflie, 2019. URL: https://www.bitcraze.io/documentation/repository/crazyflie-firmware/master/functional-areas/loco-positioning-system/.

[35] Bitcraze. Crazyflie simulation github, 2022. URL: https://github.com/bitcraze/crazyflie-simulation.

[36] Webots. Webots github samples, 2023. URL: https://github.com/cyberbotics/webots/tree/released/projects/samples.

[37] ARGoS. Large-scale robot simulations, 2023. URL: https://www.argos-sim.info/concepts.php.

[38] gymnasium. An api standard for reinforcement learning with a diverse collection of reference environments, 2023. URL: https://gymnasium.farama.org/.

[39] Pybullet. Pybullet documentation, 2023. URL: https://pybullet.org/wordpress/index.php/forum-2/.