SIMULATING THE EMERGENT AUTONOMOUS BEHAVIOUR OF UNMANNED AERIAL VEHICLE SWARMS

by

ANTONIO BRITO

SECOND DRAFT

Supervised by Nathan Griffiths

Department of Computer Science

University of Warwick

2024

Contents

List of Figures

List of Tables

SIMULATING THE EMERGENT AUTONOMOUS BEHAVIOUR OF UNMANNED AERIAL VEHICLE SWARMS

ANTONIO BRITO

Abstract

Path planning and control algorithms for mobile robots, specifically unmanned aerial vehicles (UAVs), which typically fall under the A* family of algorithms, have been well explored[3, 4]. Likewise, multi-agent pathfinding within the context of UAV systems has been explored, most notably by Burwell [5].

Notably, early developments in the field of swarm robotics on ground vehicles had been made[6], with the focus shifting from ground vehicles to aerial vehicles in recent years. With this have come descriptions of the technical frameworks[7] and operational challenges of UAV swarms[8].

This project will explore the feasibility of using a simpler, rule-based system to simulate emergent behaviour for control of UAVs, based on an extension of the principles of Reynolds' Boids[9].

We also explore the limitations of this approach, specifically in 3D, and explore an extension to our developed algorithm which focuses on local search and a potential field approach to path planning (Local Potential Field Emergence), to reduce complexity in the parameter space of the Boids algorithm.

Through this project, we aim to provide a proof of concept for the feasibility of emergent behaviour in UAV swarms and to provide a basis for further research in the field.

Keywords: Unmanned Aerial Vehicles, Swarm Robotics, Artificial Intelligence, Emergent Behaviour, Simulation, Function Optimisation, Artificial Potential Fields

1 Context

1.1 Introduction

The use of unmanned aerial vehicles (UAVs) has been growing rapidly, with both civil and military applications. Namely, their ease of deployment, low maintenance cost, high mobility and ability to hover mean they are well suited to a variety of tasks, such as surveillance, reconnaissance, search and rescue, and logistics[10].

Utilising multiple agents for these applications has several advantages. Firstly, it introduces redundancy into the system, allowing for effective fault tolerance, which is extremely vital in safety-critical applications[11]. A recent example of this is the use of UAV swarms in forest fire monitoring[12].

Additionally, the ability to scale the number of agents in the swarm allows for greater efficiency in surveillance and reconnaissance tasks, as well as the ability to cover a larger area in search and rescue operations over a given period.

In Section 4, we will explore tailoring the behaviour of our agents to simulate some scenarios, based on these expected applications of UAVs. To this end, we will explore the flexibility of UAV swarms in the context of adapting to different scenarios.

Implementations for the control and path planning of mobile robotics are well studied. Namely, A* and its variants are well studied and widely used for path planning in mobile robotics[4]. Additionally, Particle Swarm Optimisation (PSO) has been used for path planning in UAV swarms[13].

This project's focus will be on a simulated implementation of Reynolds' Boids[9] algorithm, which is a rule-based system for simulating emergent behaviour in a flock of birds. It has been demonstrated that this algorithm can be used for effective autonomous control of swarms[14], but lacks in the ability for pathfinding and obstacle avoidance[13]. Hence, we will look to extend this autonomous control in a world-like environment, by implementing goal-seeking, terrain avoidance and obstacle avoidance as added behaviours.

The implementation of this algorithm will be in the context of a 3D environment, to provide a proof of concept for the feasibility of emergent behaviour in UAV swarms, based on optimising the influence of the behaviours in our algorithm to align to our expected applications of UAV swarms. We will explore issues with this approach, and propose an extension to our developed algorithm which modifies the behaviours of the agents to simulate an artificial potential field, which has been shown to be effective as

a mobility model for UAV swarms[15].

1.2 Background

The foundation of this proof of concept is anchored in the Unity game engine, known for its robust physics engine and comprehensive 3D environment capabilities. These features are instrumental for our simulation as they proficiently manage collisions, rigid body dynamics, and raycasting. The versatility of Unity is evident from its proven track record in handling complex agent-based simulations, which extends to sophisticated artificial intelligence tasks, including reinforcement learning [16]. This versatility validates Unity as an apt platform for conducting extensive simulations required by this project.

Our project aims to advance the traditional two-dimensional Boids simulations used in UAV swarm control, as discussed by Madey and Madey [17]. Their work provides a comprehensive exploration of UAV swarm command and control strategies within a simulated environment. This paper offers valuable insights into the complexities of designing swarm behaviours that are not only autonomous but also capable of executing sophisticated tasks as a cohesive unit. The authors address both the potential and the challenges of UAV swarms in complex mission scenarios, making their research highly relevant to the development of 3D UAV swarm simulations. Their work extends the basic principles of Reynolds' Boids model, which uses simple rules to simulate flocking behaviour, by adapting these concepts to the operational needs of military UAV swarms. This adaptation is particularly relevant to our project, as we aim to extend two-dimensional Boids simulations into three-dimensional space, incorporating more complex dynamics such as quadcopter physics.

Additionally, our approach involves an innovative integration of individual behavioural rules, following the methodologies outlined by Watson et al. [18]. This enhancement is expected to provide a more realistic and practical simulation framework that better mirrors real-world UAV operational dynamics. The incorporation of realistic dynamics, such as simplified realistic aircraft dynamics, into the flocking simulation represents a significant advancement over traditional point-mass models. This approach ensures that the simulated UAVs behave more like actual aircraft, accounting for factors like inertia and aerodynamic forces, which are crucial for creating more accurate and reliable simulations. The detailed exploration of rule weighting and its impact on flock dynamics provides a valuable framework for our project, particularly in how we might

balance different behavioural influences to achieve desired swarm behaviours. Watson et al. [18] also emphasises the importance of developing meaningful statistical metrics to quantify flocking behaviour, an approach that could greatly enhance the analytical capabilities of our simulation. By understanding the relationships between rule weightings and flocking behaviour, we can more effectively tune our simulations to achieve specific operational goals or to mimic particular real-world scenarios.

To refine the effectiveness of the combined behavioural set, we employ simulated annealing as our optimisation technique. This probabilistic method is adept at approximating the global optimum of complex cost functions, making it particularly suitable for fine-tuning the behavioural weights within our algorithm to suit specific UAV swarm applications.

In their study, Alaliyat et al. [19] present a comparative analysis of the optimisation of the Boids swarm model using Genetic Algorithms (GA) and Particle Swarm Optimisation (PSO). Their research, grounded in the application of these algorithms to enhance simulations of flocking behaviour, is both timely and relevant, particularly for real-time applications like video games and interactive simulations where computational efficiency is paramount.

The study implements the Boids model in Unity3D and optimises it using both GA and PSO. The results demonstrate that PSO not only achieves faster convergence but also maintains better computational efficiency compared to GA. This is attributed to PSO's mechanism of adjusting solutions based on both individual and social memory, which seems particularly suited to the dynamic interactions modelled in flocking behaviour.

Notably, both of these techniques present certain limitations when applied to behaviour space optimisation. Genetic algorithms, for instance, often require extensive computational resources and can be prone to premature convergence, potentially leading to suboptimal solutions. PSO, while efficient for continuous search spaces, can struggle with the discrete parameters typically found in UAV control settings. These issues underscore the challenges in behaviour optimisation and highlight the necessity for a nuanced approach that can adapt to the complexities of UAV swarm behaviour dynamics.

Our selection of simulated annealing over these methods is driven by its capability to escape local minima more effectively, thereby offering a greater probability of reaching a more comprehensive global solution. However, it's imperative to acknowledge the inherent challenges in employing this technique, such as the sensitivity to cooling

schedules and the computational demands of evaluating multiple candidate solutions.

By building on established methodologies and integrating advanced optimisation techniques, this project seeks not only to extend the theoretical framework of UAV swarm simulations but also to provide actionable insights that could enhance practical implementations in real-world scenarios.

2 Physics Model

Here, we will discuss the physics model of our simulation, which is comprised of the agents that make up the flock and each agent's interactions with the surrounding environment. We will also discuss the control system for the agents; how they handle inputs and how they move in response to these inputs.

We first note the environment in which the agents will be operating, which is comprised of a three-dimensional coordinate space, representing the global reference frame. The coordinate axes of this space are x, y and z, representing north, up and east respectively.

2.1 Agent

We define an agent as a single unit within the simulation, which is capable of movement and interaction with the environment. The agent is capable of moving in three dimensions, with the ability to move north, south, east, west, up and down. The principal axes of an agent's movement are *pitch*, along the transverse (z) axis, *roll*, along the longitudinal (x) axis, and *yaw*, along the vertical (y) axis.

We define the agent as an unmanned aerial vehicle (UAV), which can be modelled as a rigid body with mass m, on Earth. As such, gravity acts on the agent at $9.81\frac{m}{s^2}$. The drag coefficient is estimated at 0.975 considering the mass of the body [20]. The dimensions and characteristics of the agent have been determined using both realistic averages [21] and estimations. These can be seen in Fig. 1.

Parameter	Value
Mass, m	10 kg
Length, l	0.7 m
Width, w	0.7 m
Height, h	0.25 m
Propeller Area, A	0.16 m^2
Distance To Propeller Centre, d	0.5 m
Drag Coefficient, C_d	0.975

Figure 1: Dimensions of the Agent

The agent is modelled as a quadcopter, with four propellers that can be independently controlled to affect the movement of the agent. Thamm et al. [22] notes that quadcopters are the most common type of UAV, due to their simplicity in design and

control, as well as their ability to hover and manoeuvre in tight spaces, owing to their ability to rotate about their principal axes. For this reason, we have chosen to model our agent as a quadcopter.

The body diagram of the quadcopter is shown in Fig. 2 alongside the local axes of movement. The agent as modelled in Unity is shown in Fig. 3 with the same axes.

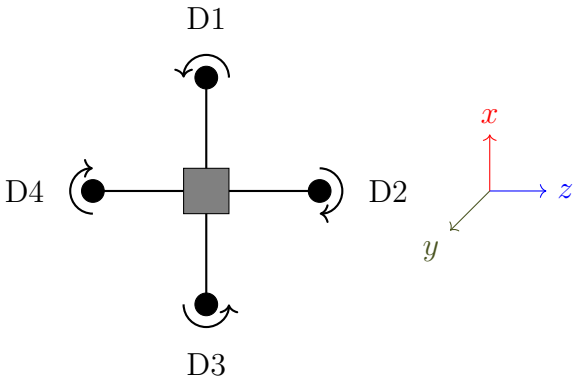


Figure 2: Quadcopter Body Diagram in the x-z plane

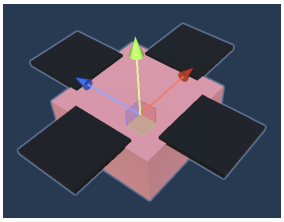


Figure 3: Agent in Unity

We can control the agent's movement by adjusting the pitch, roll and yaw angles, as well as the thrust level of the agent's propellers. We define these as:

- Roll angle (in radians), ψ
- Pitch angle (in radians), θ
- Yaw rate (in $\frac{rad}{s}$), ϕ
- Vertical thrust (in Newtons), T

For simplicity, these can be solely controlled by combinations of propeller thrust levels. A base thrust level, T_b, can be defined, which is the minimum thrust level required to maintain a stable hover. This is the thrust level required to counteract

gravity, such that $T_b = mg$. For each control operation, T_b can be augmented by a thrust level, T_{add}, such that $|T_{add}| < T_b$, which is the additional thrust required to perform the operation. As such, the control parameters can be operated by the following eight combinations of thrust levels for each propeller, with a high thrust level $T_{add} > 0$ and a low thrust level $T_{add} = 0$:

Operation	High Thrust	Low Thrust
Positive Roll, ψ_+	D_4	D_2
Negative Roll, ψ_-	D_2	D_4
Positive Pitch, θ_+	D_3	D_1
Negative Pitch, θ_-	D_1	D_3
Positive Thrust, T_+	D_1, D_2, D_3, D_4	none
Negative Thrust, T_-	none	D_1, D_2, D_3, D_4

Table 1: Control Operations and Thrust Levels

It is noted that yaw is omitted from Table 1. Within our simulation, the propellers do not rotate. This lends itself to needing a workaround for simulating yaw. In *Unity*'s physics engine, the rotation of opposite propellers can be simulated by applying forces along the x and z axes. As such, yaw is simulated according to the values in Table 2. The resultant forces are shown in Fig. 4.

Operation	Propeller	Direction of Force (respectively)
Positive Yaw, ϕ_+	D_1, D_2	Z_+, X_-
Positive Yaw, ϕ_+	D_3, D_4	Z_-, X_+
Negative Yaw, ϕ_-	D_1, D_2	Z_-, X_+
Negative Yaw, ϕ_-	D_3, D_4	Z_+, X_+

Table 2: Yaw Directions and Respective Forces

In real-world physics, a couple of propellers could not provide thrust in both extremes of the same axes, as they could not switch rotation direction. Likewise, forces in adjacent propellers would not cause rotation in the same direction. However, for simplicity within the simulation, this is ignored. It is then possible to set some thrust constants for each control operation. An example assignment for the pitch and roll operations is seen below in Fig. 3.

At this stage, we have a model for the control and movement of the agent. Input is given to the propeller controller, which outputs the thrust level as required in Fig. 3

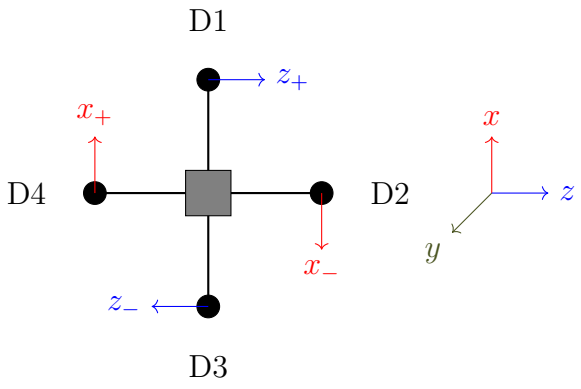


Figure 4: Propeller Forces for Positive (Clockwise) Yaw Control

Thrust Level	Thrust (N)
High	$\frac{2m \cdot g}{4}$
Normal	$\frac{m \cdot g}{4}$
Low	$\frac{0.5m \cdot g}{4}$

Table 3: Thrust Constants for a drone of mass m kg

to the individual propellers, causing a rotation of the agent in the desired direction.

2.2 Stabilisation

An issue arises with this implementation; the simulation becomes unstable as thrust cannot be provided in an accurate enough manner to counteract excessive rotation, notably in the roll and pitch axes. This can lead to the agent's rotation becoming overly large, causing it to roll over. If we consider an example input, represented by a single keypress to generate thrust to cause a pitch rotation, we can model the deviation from the desired pitch angle $\theta_d = 0.26$ over time, as shown in Fig. 5.

Fig. 5 shows that when given an input, the agent exceeds the desired pitch angle, crossing $\theta_d = 0$, then continues to rotate until it collides with the ground at $t \approx 2.2$ seconds. As such, the need for a control system becomes apparent, to mitigate the agent's desire to overshoot the desired angle.

2.2.1 Proportional Controller

A first intuition here is to control the rotation by reducing the thrust level as the agent approaches the desired angle, such that the thrust level at some time, t, is inversely

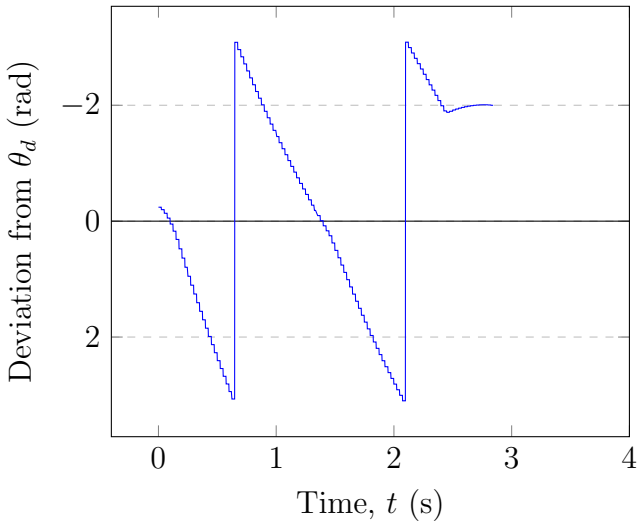


Figure 5: Pitch angle due to input over time

proportional to the deviation from the desired angle as shown in Eq. 1.

$$T_t \propto \frac{1}{\theta_d} \tag{1}$$

We can then define a proportionality constant, K_p according to the inverse relationship in Eq. 1. Using different values for K_p, we can observe some behaviours in the deviation from the desired angle over time, as shown in Fig. 6, which is split into two subfigures for clarity. The left-hand graph shows $K_p = 1$ and $K_p = 5$, and the right-hand graph shows $K_p = 5$ and $K_p = 10$.

As shown in Fig. 6, we can see that adding a proportional controller to the system creates a steady state, as we can see that the limit of θ_d as $t \to \infty$ is ≈ 0 in both graphs.

In the left-hand graph, we can see that increasing K_p decreases the amount of deviation from the desired angle. However, in the right-hand graph, we can see that this increase becomes negligible at a point, and does not reach zero. Instead, we can observe some oscillations in this state, as the agent continually overshoots the desired angle, before correcting itself and overshooting again, to a lesser extent each time. In some sense, we require the agent's controller to predict overshooting the desired angle.

2.2.2 Derivative Controller

To predict overshooting the desired angle, we need to modify the controller to consider the rate of change of the deviation. A deviation that is changing rapidly (oscillating)

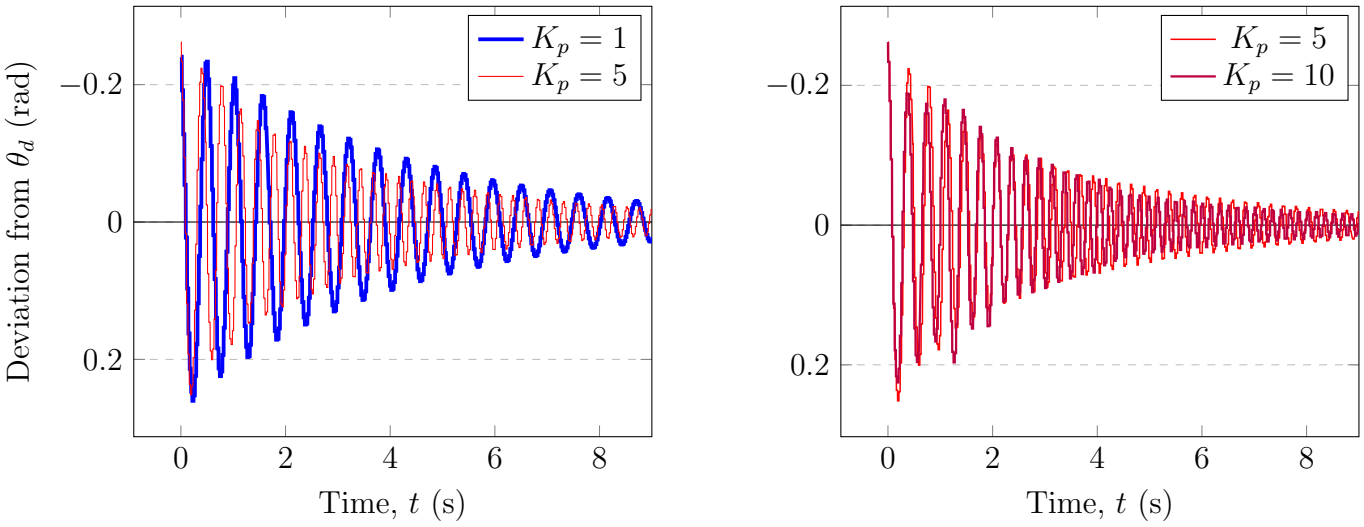


Figure 6: Pitch angle due to input over time with proportional error correction

will also have a large (but decreasing) rate of change of deviation. The aim is to reach a steady state; a deviation that is not changing over time, with a deviation of zero. Hence, we can aim to reduce this rate of change of deviation by adding a term to the controller that is proportional to the rate of change of deviation, namely, a derivative term. This is shown in Eq. 2.

$$T_t \propto \frac{1}{\theta_d} - K_d \frac{d\theta}{dt} \tag{2}$$

As before, we can introduce a proportionality constant, K_d, to the derivative term. We can then observe the effect of this term on the deviation from the desired angle over time, as shown in Fig. 7. We note that we keep $K_p = 5$ constant; it is the value that has shown the best performance in the proportional controller in the previous section.

Fig. 7 shows that adding a derivative controller with a small K_d reduces the length of time of oscillation before reaching the desired angle. This behaviour can be seen at $K_d = 0.1$. At $K_d = 0.5$, this behaviour is improved and the oscillations stop entirely. However, at $K_d = 1$, whilst the controller achieves a steady state with minimal oscillation, it has overshot the desired angle. This is defined as steady-state error.

Furthermore, a second issue with the system has arisen; the agent's rotation is not quick enough to respond to an input. For example, we can see the responsiveness of the agent to a new input in Fig. 8, where a forward pitch command is given, as before, at $t = 0$ seconds, then a backward pitch command is given at $t = 3.7$ seconds, both of which are shown as vertical lines on the graph.

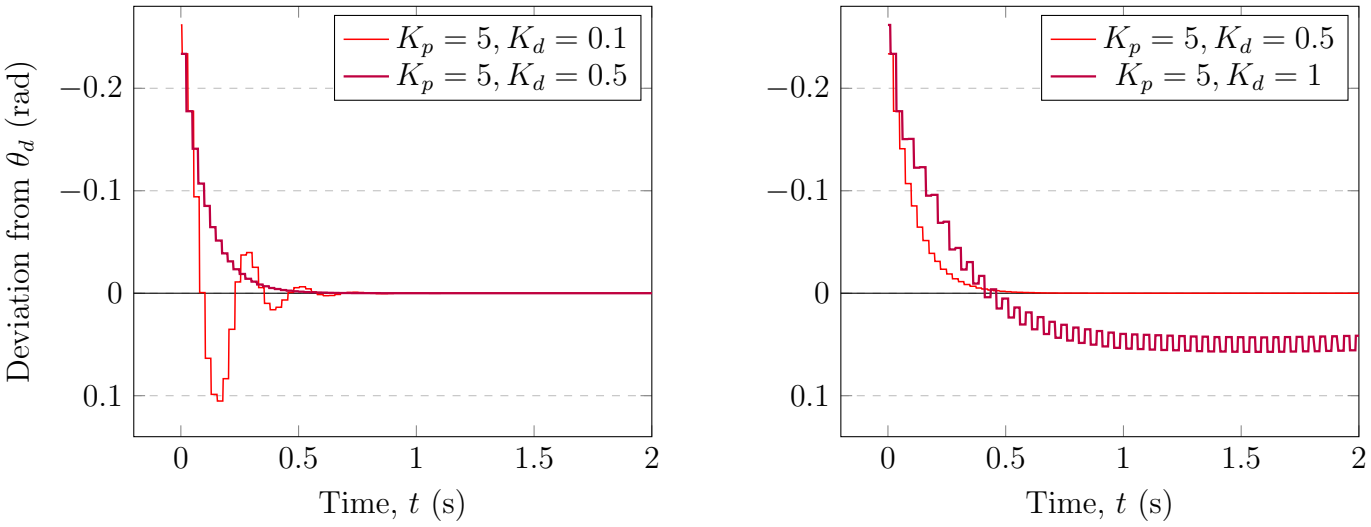


Figure 7: Pitch angle due to input over time with additional derivative error correction

We can see that the time to settle is ≈ 0.5 seconds, which may cause issues in our simulation. When introducing multiple agents into the system, responsiveness is key, to ensure last-minute direction changes can be made, for example, to avoid collisions.

2.2.3 Integral Controller

We can reduce this reaction time and the possibility of steady-state error by attempting to reduce the area between the θ_d and the actual angle (the curve in the graphs).

Namely, we can introduce an integral term, which is proportional to the area between the θ_d and the actual angle. This is shown in Eq. 3.

$$T_t \propto \int_0^t \theta_d - \theta(\tau) d\tau \tag{3}$$

We can then observe the effect of this term on the deviation from the desired angle over time, as shown in Fig. 9. We note that we keep $K_p = 5$ and $K_d = 0.5$ constant, as a sufficient set of results from the previous controllers, and test the values $K_i = 2$ and $K_i = 5$.

We note that the time to achieve a steady state is reduced, meaning the responsiveness of the system has improved. However, we note now that with the constant $K_p = 5, K_i = 5, K_d = 0.5$, an overshoot is present. To mitigate this, we can increase K_d. This is shown in Fig. 10.

Note that in Fig. 10, the overshoot is reduced, but we now have steady-state and

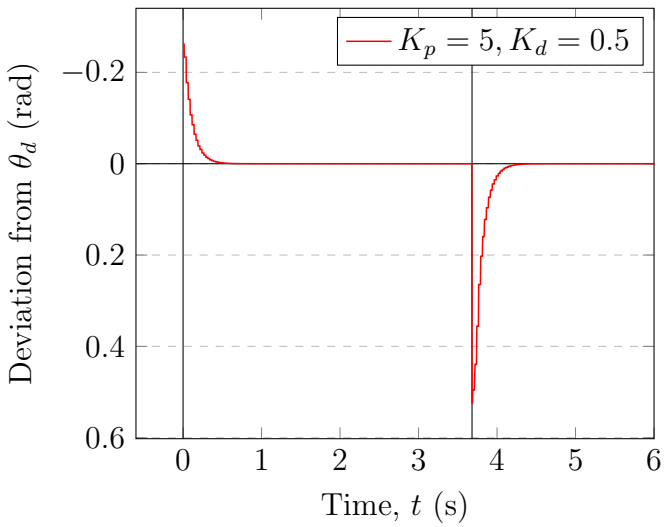


Figure 8: Pitch angle due to inputs at $t = 0$ and $t = 3.7$ seconds

small-angle oscillation errors. We can see that tuning the PID controller is a complex task akin to a chicken-and-egg problem, which we will explore further in Section 2.2.5.

2.2.4 PID Controller

We have constructed a PID (Proportional-Integral-Derivative) system; a feedback control system which uses the error between the current state and the desired state to calculate the control parameters. We now generalise this for all possible movement operations and call the desired state r and the current state y.

Fig. 11 shows the feedback loop of the PID controller. The error, $e(t)$, is calculated as the difference between the desired state, r, and the current state, y. The error is then fed into the three controllers, with the output of each controller then summed to produce the control variable, $u(t)$, which is then fed into the system. The system then produces the output, y, which is fed back into the error calculation.

The PID controller is defined by three constant parameters derived earlier, K_p, K_i and K_d, which are the proportional, integral and derivative gains respectively. The control variable, $u(t)$, is then defined as:

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)d\tau + K_d \frac{de(t)}{dt} \tag{4}$$

where $e(t)$ is the error at time t. Considering our simulation operates in discrete time intervals (frames), this can be approximated, using the *Euler method*, using the

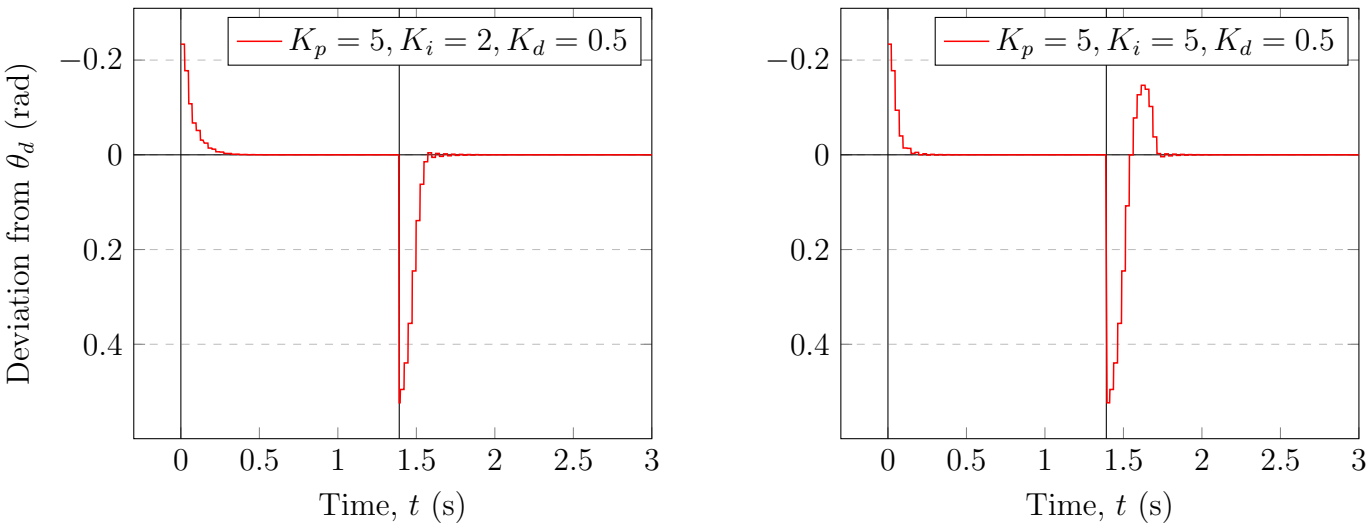


Figure 9: Pitch angle due to inputs at $t = 0$ and $t = 1.4$ seconds

slope of the tangent to the curve of the solution at a known point to estimate the value of the solution at a nearby point. Thus, we have:

$$u(t) = K_p e(t) + K_i \frac{(e_t + e_{t-1})t}{2} + K_d \frac{e_t - e_{t-1}}{t} \tag{5}$$

Hence, a feedback loop of the entire system can be produced, taking into account different PID controllers for each control operation. This is shown in Fig. 12.

2.2.5 PID Parameter Tuning

As noted previously in Section 2.2.3, tuning the PID controller is a complex task and often suboptimal[23]. We require the PID controller to be stable, responsive and have minimal steady-state error to ensure the agents are responsive and accurate in their movements. As a result, we require a method to tune the parameters of the PID controller to achieve these goals. We will explore two such methods.

Empirical Method Using a manual tuning method, we can observe the behaviour of the system and adjust the constants accordingly. Whilst this method is time-consuming due to the nature of the number of simulations required, it is simplistic and can be used to determine a set of sufficient values, which can then be fine-tuned using a more complex method if required.

This method consisted of performing a series of simulations with some initial values

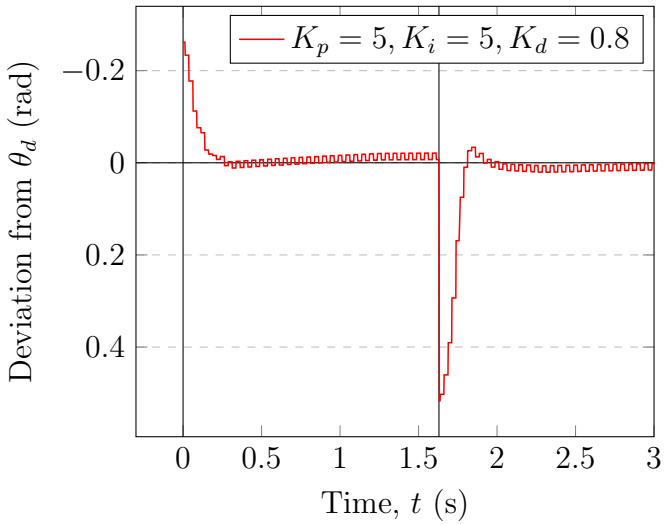


Figure 10: Pitch angle due to inputs at $t = 0$ and $t = 1.6$ seconds

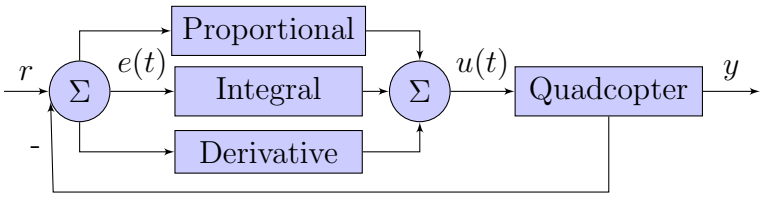


Figure 11: PID Controller

for the PID constants, adjusting these constants based on the behaviour of the system. The adjustments made were dependent on the behaviour of the system; specifically the type of error present, as shown in Table 4. A set of sufficient values was determined as shown in Table 5. It is noted that the agent is slightly unstable at very small angles. This is expected due to the imperfect nature of the empirical tuning method.

Parameter	Rise Time	Overshoot	Settling Time	Steady-State Error	Stability
K_p	Decrease	Increase	Small Increase	Decrease	Degrade
K_i	Decrease	Increase	Increase	Eliminate	Degrade
K_d	Decrease	Decrease	Decrease	No Effect	Improve

Table 4: Effects of increasing a parameter independently[2]

Ziegler-Nichols Method The Ziegler-Nichols method is a heuristic method for tuning PID controllers. It is based on the response of the system to a step input. McCormack and Godfrey [24] propose a set of tuning rules for a classic control system,

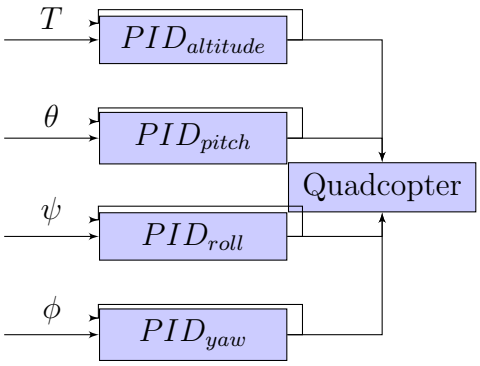


Figure 12: System Model

Control Operation	K_p	K_i	K_d
Thrust	6	5	2
Pitch	10	10	2
Roll	10	10	2
Yaw	10	10	2

Table 5: Tuning Constants

which result in the proportional gain K_p and two values representing the integral and derivative time constants, T_i, T_d, respectively. An extract from *Table 1* of the paper is shown in Table 6.

Tuning Rule	Required	Controller Parameters
ZN	K_u, T_u	$K_p = 0.6K_u, T_i = 0.5T_u, T_d = 0.125T_u$

Table 6: Ziegler-Nichols Method

The term K_u refers to the optimal gain; the gain at which the system oscillates at a constant amplitude and frequency. The term T_u refers to the period of this oscillation. Fig. 13 shows the oscillation of the system following a positive pitch command with K_p increasing from 1 at a rate of 1 every second. We can then determine the optimal gain to be the point at which the oscillations become stable. This is shown to be $K_p \approx 7$. A zoomed-in version of the plot shows the period of this oscillation to be $\approx T_u = 0.6$ seconds.

To fine-tune the controller parameters, we reference Table 6 which provides us with the initial values: $K_p = 4.2$, $T_i = 0.3$, and $T_d = 0.075$. These values serve as a basis for calculating the integral and derivative gains, according to the relationships established in Equation 6:

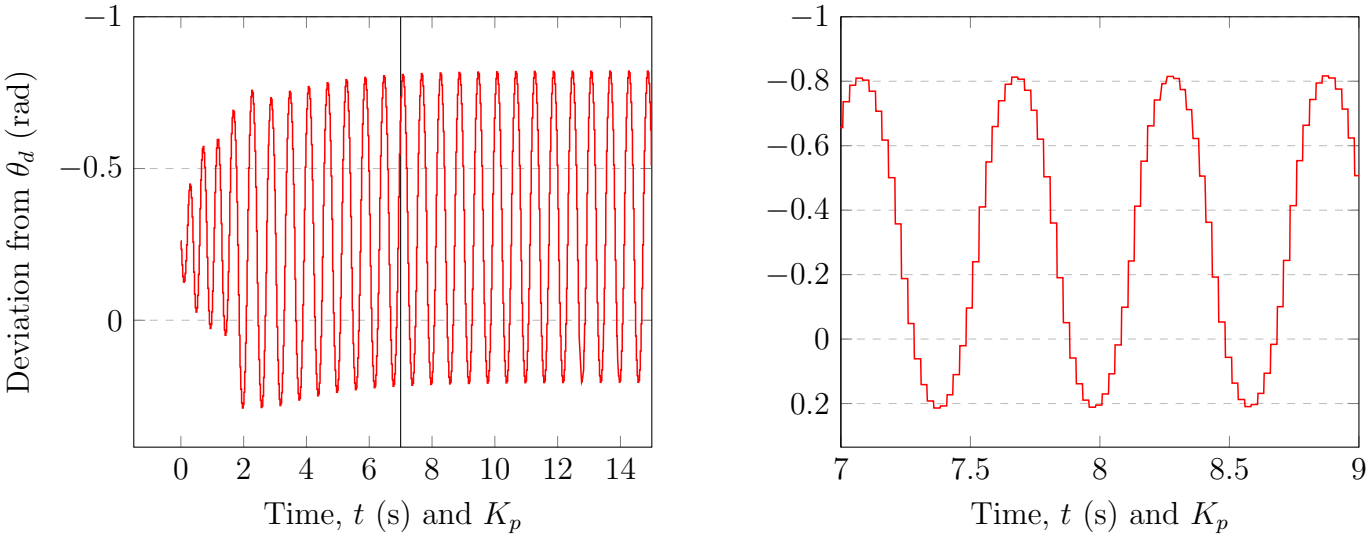


Figure 13: Oscillation of the system with varying K_p

$$K_i = \frac{K_p}{T_i} \quad \text{and} \quad K_d = K_p \cdot T_d \tag{6}$$

Applying these formulas, we compute the integral gain, K_i, to be 14, and the derivative gain, K_d, to be 0.315. To evaluate the effectiveness of these settings, we conduct a comparison with results derived from an empirical method previously discussed. Specifically, we simulate a scenario where a positive pitch command is applied to achieve a desired angle $\theta_d = 0.26$ radians, followed by a command release to reset the angle to $\theta_d = 0$. The response of the system under both sets of tuning constants is depicted in Figure 14, allowing us to assess and compare their behaviours in real-time conditions.

We can see that the empirical method results in a faster settling time, but has a small steady-state error, which is not present in the results of the Ziegler-Nichols method. However, the Ziegler-Nichols method results in a small oscillation centred at the desired angle.

As the Ziegler-Nichols method is heuristic, it is not guaranteed to be optimal. However, it is a good starting point for tuning the PID controller. We can see from our comparison that an ideal set of constants lies somewhere between the results from both methods. We can improve on our current results by observing the behaviour of the system and adjusting the constants accordingly.

Namely, we can see that the small oscillation present in the Ziegler-Nichols method can be reduced by increasing the derivative gain, such that $K_d = 0.9$. This should also

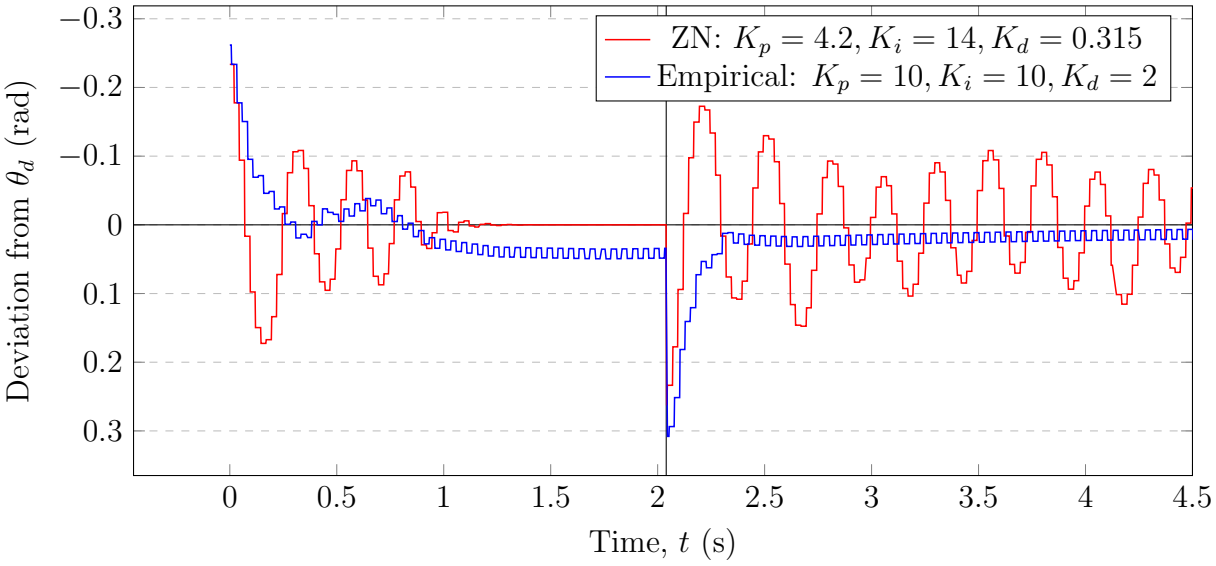


Figure 14: Pitch angle due to inputs at $t = 0$ and $t = 2.03$ seconds

mitigate the small steady-state error present in the empirical method. To reduce this further, we will adjust the proportional gain such that $K_p = 8$ and the integral gain such that $K_i = 12$. This is shown in Fig. 15.

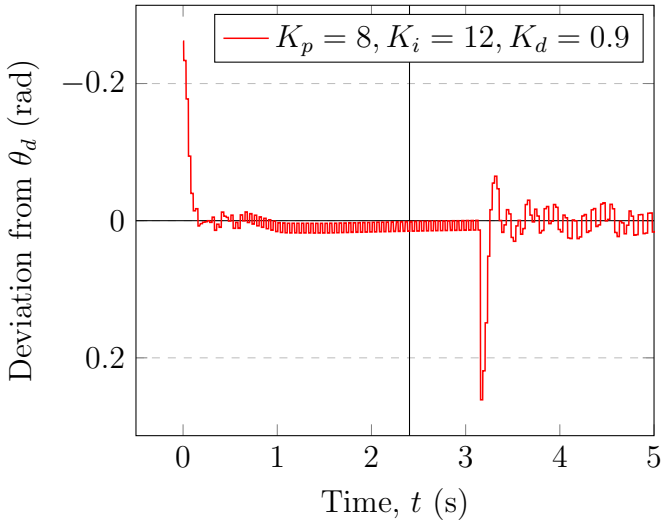


Figure 15: Pitch angle due to inputs at $t = 0$ and $t = 1.6$ seconds

Conclusion Figure 15 illustrates the outcomes of tuning a PID pitch controller using a hybrid approach that combines empirical and heuristic methods. The values obtained from this refined process will now be adapted to enhance the control mechanisms for

both roll and yaw. Meanwhile, for simplicity, we will retain the thrust constants that were previously determined through the empirical approach. The consolidated PID constants for all controls are summarized in Table 7.

Despite these improvements, a steady-state error persists within the system. While advanced tuning methods are available that could potentially minimize this error further, for the scope of this project, the current level of tuning is deemed adequate. This decision is based on the practical considerations and the specific objectives outlined for this project, balancing the need for precision with the feasibility and resource constraints.

Control Operation	K_p	K_i	K_d
Thrust	6	5	2
Pitch	8	12	0.9
Roll	8	12	0.9
Yaw	8	12	0.9

Table 7: Final PID Constants

2.3 Environment Model

We will now discuss how the agents interact with each other and objects in the environment according to the Unity physics engine.

Agents may collide with any object in the environment defined to have a 'collider'. This is a built-in Unity component that is defined by the shape of the object and is used to detect collisions with other objects. The agent's collider is comprised of the colliders of its parts, namely the agent's body and its rotors. Likewise, all objects we instantiate from this point onwards, such as the terrain, obstacles and platforms, will have colliders. We will also specify that the agents are resistant to collisions with other agents to reduce the complexity of the simulation.

3 Introducing Autonomy

The initial aim of the simulation is to achieve position control for an agent, such that when given a current position $p_0 = [x_0, y_0, z_0]$ and a goal position $p_{goal} = [x_1, y_1, z_1]$, the agent will move to the goal position. This sets the foundation for the behaviour of the agents; we will model these as a 'desire' to move to a specific position.

There are generally two methods to achieve this. A simple but inefficient method would be to rotate the initial heading of the agent towards the goal position, namely the yaw component, which would solely require the agent to pitch forward until it reaches the goal position.

This method is time inefficient, as the agent will need to rotate to face the goal position before moving towards it, which makes it unsuitable for environments where the goal position is not static or where the agent must be reactive to its surroundings. As such, we can introduce a more efficient method, which involves the agent determining an optimal combination of pitch and roll commands to reach the goal position.

3.1 Movement to Position

Once a goal position, p_{goal}, is determined, the agent must determine the optimal combination of pitch and roll (input) commands to reach the goal position from its current position, p_0.

In order to determine these input commands, we can model these two points on a coordinate grid. For representational clarity, we will assume the model to be in two dimensions for now. We can then calculate the vector between the two points, $v_g = p_{goal} - p_0$, which represents the vector the agent must move along to reach the goal position. We call this the transformation vector, which is shown in Fig. 16.

Once the agent has determined the transformation vector, v_g, it must then determine the optimal combination of pitch and roll commands to reach the goal position. Whilst v_g represents the direction the agent must move in, the components of v_g represent the movement along the global axes, not the agent's local axes. Specifically, the agent's pitch and roll commands will not cause a movement along the global axes, except in the trivial case where the agent is lined up with the global axes.

As such, a change of basis is required to convert the global vector v_g into the local frame of the agent. This change of basis between two vectors represents a linear map between the two vector spaces. More specifically, it represents a linear transformation

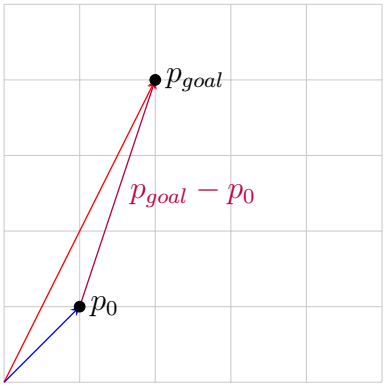


Figure 16: 2D Coordinate Grid with p_{goal} and p_0

which maps the global vector v_g to the local vector v_l. This transformation is represented by a transformation matrix, M_2, such that:

$$\begin{bmatrix} x_l \\ y_l \end{bmatrix} = M_2 \cdot \begin{bmatrix} x_g \\ y_g \end{bmatrix} \tag{7}$$

We note that the transformation matrix M_2 need only represent the rotation of the agent, as the representations are position and scale invariant; the agent's position and scale do not change between reference frames. In two dimensions, we can extend this simplification as the agent only has the freedom of rotation about a singular axis. Hence, M_2 need only be a rotation matrix representing the yaw of the agent.

We can derive M_2 by first considering a vector in the global frame, v_g and the local frame v_l. We note the vectors have equal magnitudes due to the scale invariance property, thus, it holds that $|v_g| = |v_l|$. We can then determine the angle between the two vectors, β, and the angle between the global x-axis and the vector v_g, α, as shown in Fig. 17.

Thus, we can determine that the x and y components of v_g are:

$$\begin{aligned} x_g &= |v_g| \cos(\alpha) \\ y_g &= |v_g| \sin(\alpha) \end{aligned} \tag{8}$$

It must then hold that:

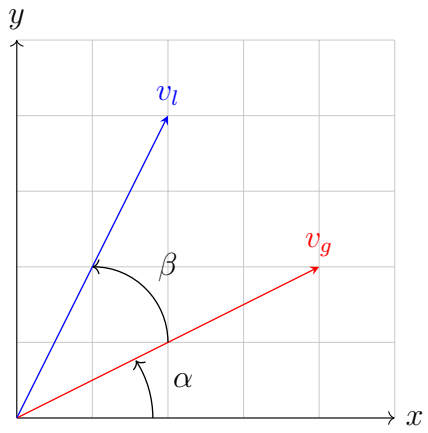


Figure 17: 2D Coordinate Grid with v_l and v_g

$$
\begin{aligned}
x_l &= |v_l| \cos(\alpha + \beta) \\
y_l &= |v_l| \sin(\alpha + \beta) \\
&\implies \\
x_l &= |v_g| \cos(\alpha)\cos(\beta) - |v_g| \sin(\alpha)\sin(\beta) \\
y_l &= |v_g| \sin(\alpha)\cos(\beta) + |v_g| \cos(\alpha)\sin(\beta) \\
&\implies \\
x_l &= x_g \cos(\beta) - y_g \sin(\beta) \\
y_l &= x_g \sin(\beta) + y_g \cos(\beta)
\end{aligned}
\tag{9}
$$

We can represent this as a matrix equation:

$$
\begin{bmatrix} x_l \\ y_l \end{bmatrix} = \begin{bmatrix} \cos(\beta) & -\sin(\beta) \\ \sin(\beta) & \cos(\beta) \end{bmatrix} \begin{bmatrix} x_g \\ y_g \end{bmatrix}, \text{ where } M_2 = \begin{bmatrix} \cos(\beta) & -\sin(\beta) \\ \sin(\beta) & \cos(\beta) \end{bmatrix} \tag{10}
$$

Returning to three dimensions to consider the full movement of the agent, we can extend this method to determine the optimal combination of pitch and roll commands to reach the goal position. We can then determine the transformation matrix M_3 to convert the global vector v_g to the local vector v_l, such that:

$$
\begin{bmatrix} x_l \\ y_l \\ z_l \end{bmatrix} = M_3 \cdot \begin{bmatrix} x_g \\ y_g \\ z_g \end{bmatrix} \tag{11}
$$

In the three-dimensional case, M_3 will be a rotation matrix representing the yaw, pitch and roll of the agent; we note it is still scale invariant and position invariant. As such, it should be comprised of three individual rotation matrices for each axis of rotation. Hence:

$$M_3 = R_y \cdot R_x \cdot R_z \tag{12}$$

We can then derive each rotation matrix in a similar method to M_2. We will consider the rotation about the x-axis first.

We note that R_x will represent a rotation about the x-axis, namely, the roll of the agent. This means that the y and z components of the vector v_g will be rotated through an angle of ψ, whilst the x component will remain constant. We can then determine the rotation matrix from earlier intuition:

$$R_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\psi) & -\sin(\psi) \\ 0 & \sin(\psi) & \cos(\psi) \end{bmatrix} \tag{13}$$

Considering the other two rotation matrices, R_y and R_z, where the xz and xy planes are rotated respectively, we can see:

$$\begin{aligned} R_y &= \begin{bmatrix} \cos(\theta) & 0 & -\sin(\theta) \\ 0 & 1 & 0 \\ \sin(\theta) & 0 & \cos(\theta) \end{bmatrix} \\ R_z &= \begin{bmatrix} \cos(\phi) & \sin(\phi) & 0 \\ -\sin(\phi) & \cos(\phi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \end{aligned} \tag{14}$$

At this stage, we have obtained the transformation matrix M_3 to convert the global vector v_g to the local vector v_l. We can then determine the optimal combination of pitch and roll commands to reach the goal position, which will be the x and z components of v_l respectively. This is shown in Fig. 18.

Usefully, this conversion step can be performed in the Unity physics engine using the built-in method `transform.InverseTransformVector()`, which will convert the vector from the global frame to the local frame of the agent. This method is the inverse of the `transform.TransformVector()` method, which converts the vector from the local frame to the global frame.

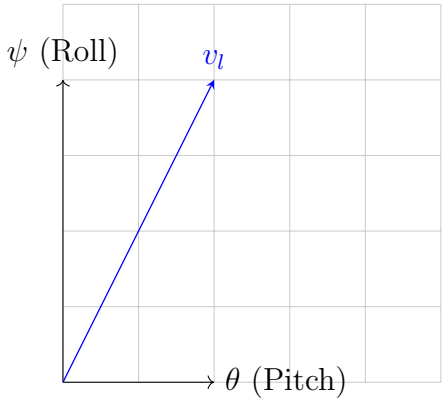


Figure 18: Constituent Pitch and Roll Components of v_l

3.2 Boids (1987)

We may now introduce multiple agents into the environment. To reduce the complexity of the position control of the agents, we will assume the agents hover by default. Specifically, when given no control input, the agents resort to a default hover state.

We will make some assumptions about the awareness of the agents at this stage and provide some important definitions. These are:

- Agents are aware of the positions of all other agents within a certain radius, r. This is defined as the *neighbourhood radius*.
- Agents are aware of their position with respect to the global reference frame.
- Agents in a flock of size n are defined as $a_1, ..., a_n \in A$.

The motivation behind the use of the neighbourhood radius is to reduce computational complexity. Generally, an increase in the number of adjacent agents will increase the number of calculations required. We can reduce this by discounting the agents that have no effect on the behaviour of the agent in question.

We will then introduce the concept of *Boids* [9]. This is a method of simulating flock behaviour, where agents are programmed to exhibit a set of behaviours that result in emergent flock behaviour. We will explore the implementation of these behaviours in the next section. For all of the following behaviours, we calculate a force vector that is applied to the agent, concerning the agent's local frame.

3.2.1 Avoidance

The avoidance behaviour is used as a method for agents to avoid collisions with each other. To this end, the behaviour causes agents to move away from each other when they are within a certain distance of each other.

We will call this distance d, and define it as the *square avoidance radius*. This is the distance at which agents will begin to move away from each other. We will define this as the square of the distance between two agents, $a_1, a_2 \in A$, such that $d = (|a_1| - |a_2|)^2$, for the purposes of algorithmic efficiency. This is a common method in the *Unity* physics engine [25].

We can then determine the avoidance vector for an agent, a, by calculating the mean distance vector between a and all other agents within the neighbourhood, a_i, such that $|a_i| - |a| < d$. Specifically, we populate the set R_a with all agents $a \in A$ that are within the neighbourhood of a.

$$\vec{A} = \begin{cases} 0 & \text{if } R_a = \emptyset \\ \frac{1}{|R_a|} \sum_{a_i \in R_a} \vec{a_i} - \vec{a} & \text{otherwise} \end{cases} \tag{15}$$

We can model an example with two agents in two-dimensional space. This can be seen in Fig. 19. We see the two agents a_1 and a_2 have position vectors $[2, 2]$ and $[3, 3]$ respectively. Hence, the avoidance vectors for the two agents are:

$$\begin{aligned} \vec{a^1_{avoid}} &= \frac{1}{1}\left(\begin{bmatrix}3\\3\end{bmatrix} - \begin{bmatrix}2\\2\end{bmatrix}\right) = \begin{bmatrix}1\\1\end{bmatrix} \\ \vec{a^2_{avoid}} &= \frac{1}{1}\left(\begin{bmatrix}2\\2\end{bmatrix} - \begin{bmatrix}3\\3\end{bmatrix}\right) = \begin{bmatrix}-1\\-1\end{bmatrix} \end{aligned} \tag{16}$$

3.2.2 Alignment

The alignment behaviour causes agents to align their headings with those of their neighbours. This may, in turn, reduce collisions. We can determine the alignment vector for an agent, a, by calculating the mean heading vector between a and all other agents within the neighbourhood, a_i, such that $|a_i| - |a| < r$. Specifically, we have:

$$\vec{B} = \begin{cases} 0 & \text{if } R_b = \emptyset \\ \frac{1}{|R_b|} \sum_{a_i \in R_b} h(\vec{a_i}) & \text{otherwise} \end{cases} \tag{17}$$

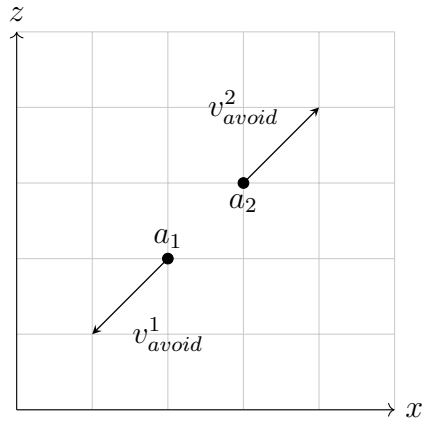


Figure 19: Avoidance vectors for two nearby agents

We can model three agents in two-dimensional space. This can be seen in Fig. 20. We see the three agents a_1, a_2 and a_3 have heading vectors $[1, 1]$, $[1, 0]$ and $[0, 1]$ respectively. We introduce three agents as the simplest stable state of the system. Using only two agents would result in a system where the heading vectors would oscillate between the two agents without any external forces. We can then determine the alignment vectors for the three agents as follows:

$$\begin{aligned} \vec{a^1_{align}} &= \frac{1}{2}\left(\begin{bmatrix}1\\0\end{bmatrix} + \begin{bmatrix}0\\1\end{bmatrix}\right) = \begin{bmatrix}0.5\\0.5\end{bmatrix} \\ \vec{a^2_{align}} &= \frac{1}{2}\left(\begin{bmatrix}1\\1\end{bmatrix} + \begin{bmatrix}0\\1\end{bmatrix}\right) = \begin{bmatrix}0.5\\1\end{bmatrix} \\ \vec{a^3_{align}} &= \frac{1}{2}\left(\begin{bmatrix}1\\1\end{bmatrix} + \begin{bmatrix}1\\0\end{bmatrix}\right) = \begin{bmatrix}1\\0.5\end{bmatrix} \end{aligned} \tag{18}$$

We can visualise the current heading vectors at time t and the determined heading vectors for time $t + 1$ (indicated in red) in Fig. 20[1], where we see the new heading vectors have less variance in their angles.

3.2.3 Cohesion

Finally, we implement the third principle, *cohesion*. This is in principle, the inverse to the avoidance principle. It ensures members of the flock stay together. We can

[1]N.B. The agents will likely not be in the same position after one timestep. We consider this as negligible in the visualisation.

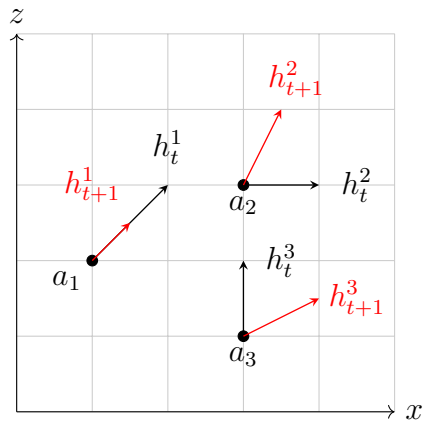


Figure 20: Alignment vectors for three nearby agents

determine the cohesion vector for an agent, a, by calculating the mean position vector between a and all other agents within the neighbourhood, a_i, such that $|a_i| - |a| < r$. Specifically, we have:

$$\vec{C} = \begin{cases} 0 & \text{if } R_c = \emptyset \\ \left(\frac{1}{|R_c|} \sum_{a_i \in R_c} \vec{a_i}\right) - \vec{a} & \text{otherwise} \end{cases} \tag{19}$$

We consider three agents a_1, a_2 and a_3 in two-dimensional space with position vectors $[1, 2]$, $[3, 3]$ and $[3, 1]$ respectively. We can then determine the cohesion vectors for the three agents as follows:

$$\begin{aligned} \vec{a^1_{cohere}} &= \frac{1}{2}\left(\begin{bmatrix}3\\3\end{bmatrix} + \begin{bmatrix}3\\1\end{bmatrix}\right) - \begin{bmatrix}1\\2\end{bmatrix} = \begin{bmatrix}2\\0\end{bmatrix} \\ \vec{a^2_{cohere}} &= \frac{1}{2}\left(\begin{bmatrix}1\\2\end{bmatrix} + \begin{bmatrix}3\\1\end{bmatrix}\right) - \begin{bmatrix}3\\3\end{bmatrix} = \begin{bmatrix}-1\\-1.5\end{bmatrix} \\ \vec{a^3_{cohere}} &= \frac{1}{2}\left(\begin{bmatrix}1\\2\end{bmatrix} + \begin{bmatrix}3\\3\end{bmatrix}\right) - \begin{bmatrix}3\\1\end{bmatrix} = \begin{bmatrix}-1\\1.5\end{bmatrix} \end{aligned} \tag{20}$$

Modelling the two agents, we can see the effect of cohesion on the determined cohesion vectors in Fig. 21.

3.2.4 Combining Behaviours

Fig. 21 demonstrates that the balance between cohesion and separation is important. If the cohesion force is too strong, the agents will collide, as shown. If the separation

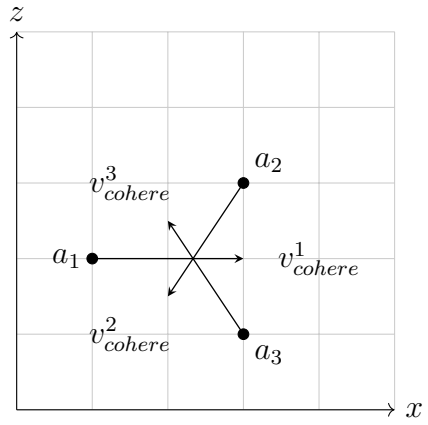


Figure 21: Alignment vectors for three nearby agents

force is too strong, the agents will become disconnected and fly apart. This is similar in principle to the *strong nuclear force* acting within the nucleus of an atom. The force is repulsive at close ranges (the avoidance radius), but attractive at larger ranges (the neighbourhood). We will explore this further in Section 6.3.

In general, tuning the influence of all three components is challenging. Even considering the constants in the simulation, Reynolds [9] notes that the relative weights that influence the strength of each component '*is a precarious interrelationship that is difficult to adjust*'. Despite this, we can combine these three independent behaviours and attempt to tune them.

We can define a *composite behaviour*, which is defined as a linear combination of weights and behaviour vectors. In three dimensions, for the three behaviours, we have:

$$\vec{V} = \alpha \begin{bmatrix} x_{avoid} \\ y_{avoid} \\ z_{avoid} \end{bmatrix} + \beta \begin{bmatrix} x_{align} \\ y_{align} \\ z_{align} \end{bmatrix} + \gamma \begin{bmatrix} x_{cohere} \\ y_{cohere} \\ z_{cohere} \end{bmatrix} \tag{21}$$

We will utilise an empirical method for tuning the constants α, β and γ. We will simulate the agents in a controlled environment and adjust the constants until the agents exhibit appropriate emergent flock behaviour. We will then analyse the emergent behaviour and adjust the constants accordingly.

Initially, we will set the constants as shown in Table 8. Additionally, we will set the weights to be equal for each component as shown in Table 9.

Figure 22a shows the x-z positions of the agents within the flock over a simulated run with these constants. We see that the agents move together in a straight line,

Constant	Value
Number of Agents	8
Spawn Radius	4
Neighbour Radius	1
Square Avoidance Radius	16

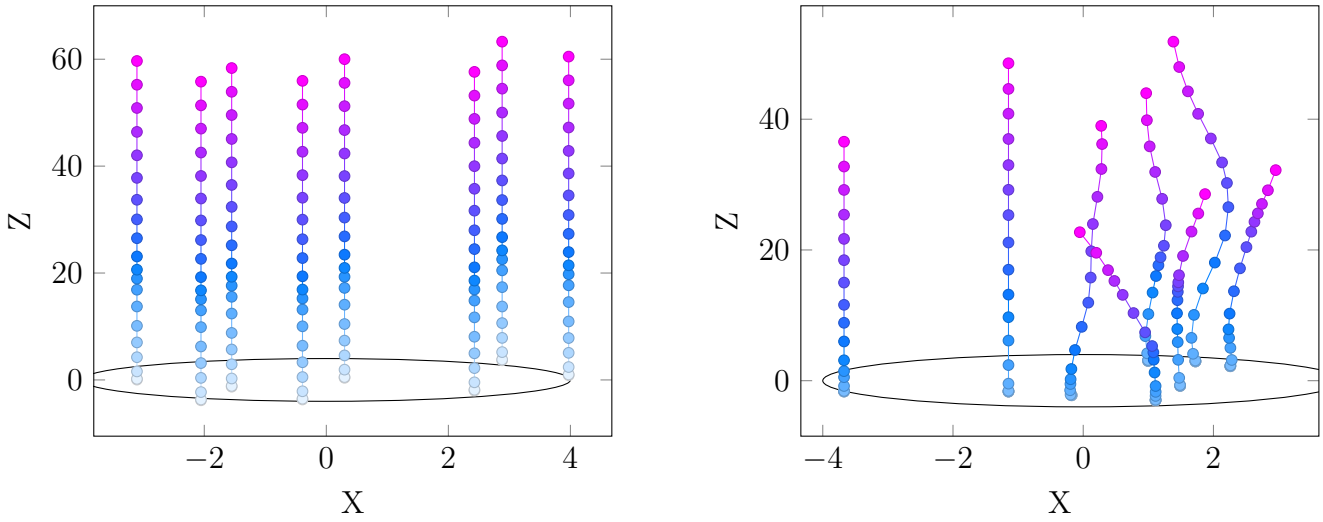
Table 8: Spawn Constants

Behaviour	Weight
Separation	1
Alignment	1
Cohesion	1

Table 9: Boids Weights

signifying that the alignment weight is initially well-tuned.

We note that the behaviour of some agents is not as is desired, as in Fig. 22a, we see there exists a large gap between two agents. This is likely due to the avoidance weight being too low, causing the agents to not move closer to each other. As a result, we will increase the avoidance weight to **1.5** and re-run the simulation. We will also increase the neighbourhood radius to **3** to mitigate issues with agents not being aware of each other.



(a) Initial

(b) Avoidance Tuning

Figure 22: Simulation Run (Start is indicated by black circle)

The results of the simulation with the updated values are shown in Fig. 22b. We note that the agents initially move away from each other due to the increased effects of avoidance, however, the leftmost two agents appear to not be pulled closer together, despite being within each other's neighbourhood.

To mitigate this, we can increase the weight of cohesion to **1.4** and reduce the square avoidance radius to **12**. Finally, we can increase the weight of alignment slightly, to

1.2, to reduce the effects of agents veering at the sides of the flock. This simulation results in the positions in Fig. 23.

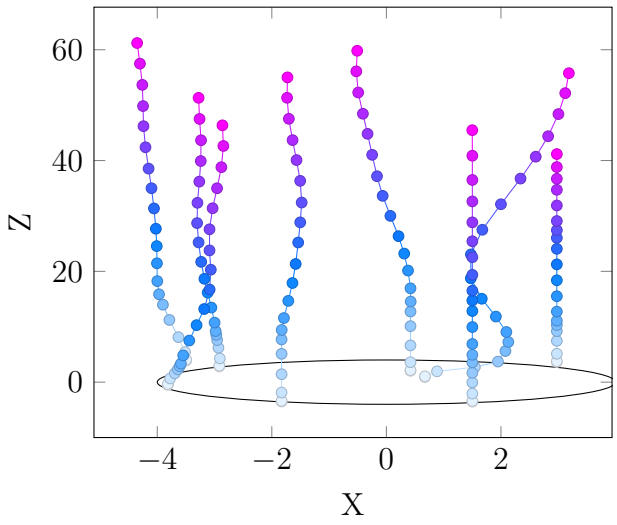


Figure 23: Final Tuning

We note that the agents now move together in a cohesive manner and are responsive to each other's movements. However, the flock's movement is not perfect; the agents have a tendency to spread apart and occasionally overcorrect, causing collisions and erratic behaviour. At this stage, we have obtained a reasonable balance between the three behaviours, with the final constants shown in Table 10.

Behaviour	Weight
Separation	1.5
Alignment	1.2
Cohesion	1.4

Table 10: Final Boids Weights

We have demonstrated the difficulty in tuning the parameters for the Boids algorithm using an empirical method. In a similar fashion to the exploration of tuning the PID controller in Section 2.2.5, we will explore heuristic methods to tune the parameters of the Boids algorithm at a later stage.

3.3 Extended Boids

We have implemented the three fundamental behaviours of the Boids algorithm. At this stage, we can extend the algorithm to include additional behaviours, allowing us

to simulate more complex emergent behaviours and provide a framework for simulating a more life-like environment for the agents.

We will consider two such extensions: *goal-seeking* and *terrain avoidance*. We will implement these behaviours in the simulation and analyse the emergent behaviours of the agents.

3.3.1 Goal Seeking

One key extension to the Boids algorithm is the inclusion of a *goal-seeking* behaviour, which allows the agents to seek a goal position within the world. Specifically, we can implement a variation of a simple pathfinding ability for the agents. The ability for autonomous agents to perform effective path planning plays a key role in autonomous control systems[26].

Whilst in the real world, path planning is a complex problem involving balancing computational complexity and optimality, we can implement a simple pathfinding algorithm for the agents in the simulation by making a key assumption about the environment; that the agents are aware of the goal position.

In reality, the agents would need to be able to sense the environment and determine the goal position. This could be achieved through the use of sensors, such as cameras or LIDAR, or through the use of a global positioning system (GPS)[27, 5].

We can implement the agents' ability to seek a goal position in a similar method to the three behaviours Reynolds mentions. We can define a *goal sphere*, with a given radius, centred around a random position in the world. We can then implement the *seeking* behaviour to calculate the distance vector between the agent and the goal in order to apply a force in the direction of the goal.

Specifically, we have:

$$\vec{D} = \vec{g} - \vec{a} \tag{22}$$

Where $\vec{D}$ is the distance vector between the agent, a, and the goal, g.

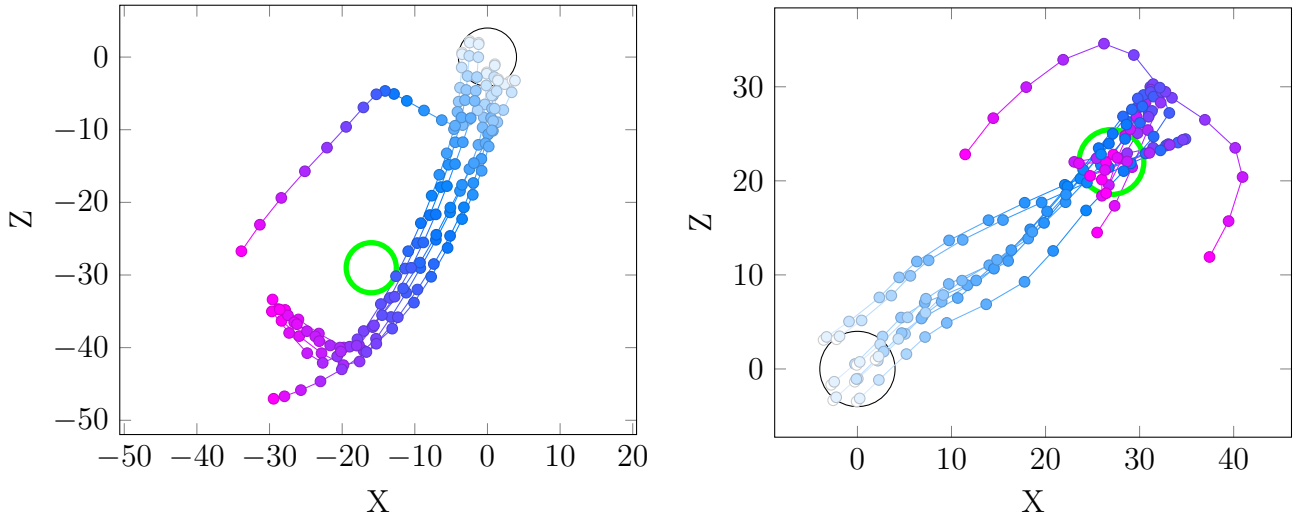
As with the previous behaviours, we will simulate the agents in a controlled environment and adjust the constants until the agents exhibit appropriate emergent flock behaviour. A run of the simulation with the goal-seeking behaviour weight set at **2** is shown in Fig. 24a[2].

[2]The location of the goal is shown by the green circle in the figure.

We note that the agents do not actually enter the goal radius, instead orbiting around the goal. This behaviour was likely down to the method being used before to implement the agent's moves; the agent would receive the given force vector, and then normalise it - acting as a maximum clamping value for the force.

This turned out to not be the best practice, as whilst in close spaces, the direction of the desired goal *(e.g. avoidance of nearby neighbours)* would be more important than the magnitude of the force, at higher distances between the current and goal position of the agent, the magnitudes, specifically of the pitch and roll components, become more significant in path planning.

As an alternative, we simply use Unity's built-in `Math.Clamp` function, which clamps a value between a minimum and maximum value. This allows us to set the minimum and maximum force values to ones we deem safe.



(a) Normalised Force Vector

(b) Clamped Force Vector

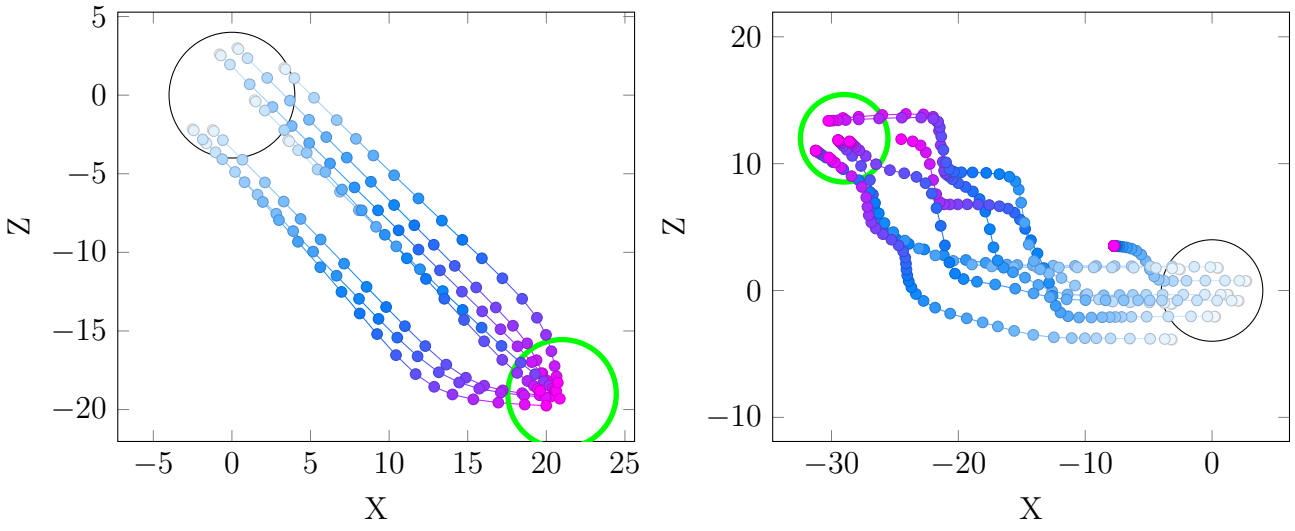
Figure 24: Seeking Behaviour

The behaviour now seen in Fig. 24b is accurate, but not ideal, as the agents overshoot the goal and find themselves backtracking during pathfinding. This can be mitigated by implementing an error controller, using the fundamentals of control theory discussed in Section 2.2.

To this extent, we may implement a proportional controller to adjust the force applied to the agents based on the distance between the agent and the goal. Specifically, the determined force is inversely proportional to the distance between the agent and the goal.

We can then instruct the agents to stop once they have detected that they are within the goal space. An example simulation of this is shown in Fig. 25a. We note that the agents gravitate towards the centre of the goal space. This is unwanted, as it does not make enough considerations for adjacent agents and their respective avoidance radii.

At this stage, we can then reintroduce our three initial behaviours and their final values from the empirical tuning in Table 10 and tune the influence of the seeking behaviour accordingly.



(a) Final Seeking Behaviour

(b) Three Initial Behaviours + Seeking

Figure 25: Finalising Seeking Behaviour

We note that the agents now exhibit the flocking behaviour present previously, with the additional ability to seek a goal; this causes the agents to move in a cohesive manner towards the goal. It follows that the weight of the seeking behaviour will also need tuning alongside the three initial behaviours.

3.3.2 Terrain Generation

In previous sections, we have focused on the pathfinding behaviour of the agents in empty space. However, in the real world, there are many obstacles that the agents must avoid. Hence, in this section, we introduce a terrain model to the simulation to parallel real-world deviations in terrain heights.

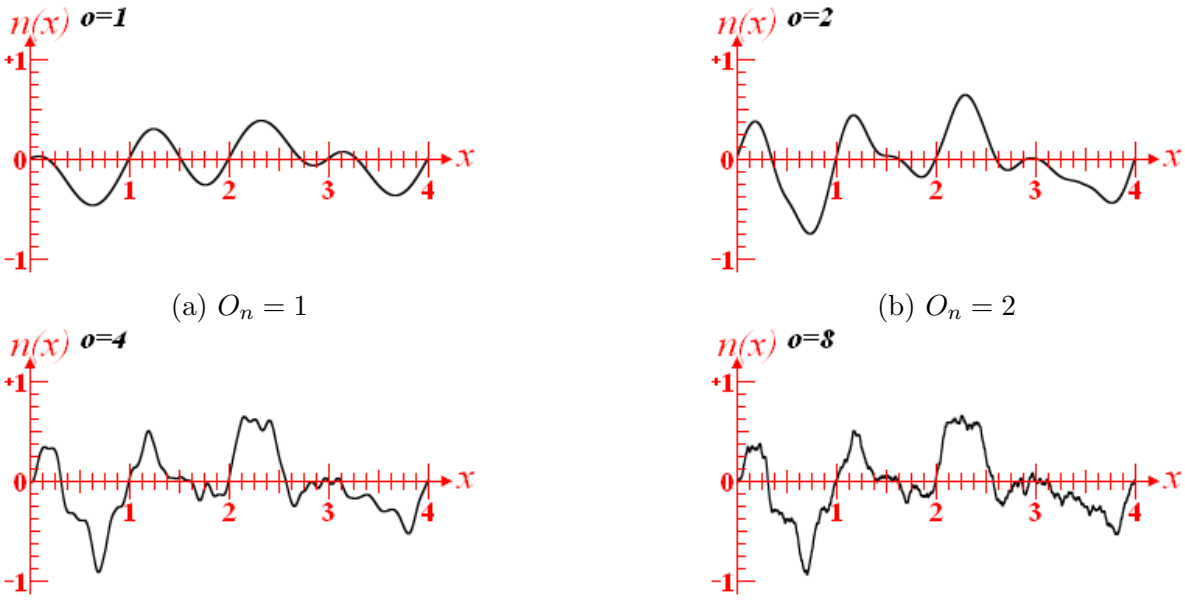
In this project, we employ Perlin noise[28, 289-293] to create textures that effectively model terrain. This method generates a texture that, when translated into a heightmap,

assigns elevations based on color intensity at specific points. The smooth terrain this method produces, thanks to the interpolated noise function, aligns perfectly with our project's goals.

For our simulation, we focus on crafting a waveform characterised by a specific frequency, f, and amplitude, A, to replicate the terrain's undulations. This waveform is crucial in generating our heightmap. We delve into four critical parameters that define our terrain generation process: the number of octaves, O_n; lacunarity, l; persistence, p; and scale, s.

An octave in this context refers to the frequency ratio within a waveform, echoing its musical counterpart by signifying a frequency doubling. The octave count, O_n, indicates how often the waveform's frequency is doubled, offering us a tool to enhance the terrain's detail level. A higher octave count means more nuanced terrain details, catering to the precision our simulation demands.

The effect of the number of octaves on one-dimensional Perlin noise is shown in Fig. 26. We can see that as the number of octaves increases, the detail in the waveform increases. Generalising over multiple dimensions, we can see that the number of octaves controls the level of detail in the terrain.



(a) $O_n = 1$ (b) $O_n = 2$

(c) $O_n = 4$ (d) $O_n = 8$

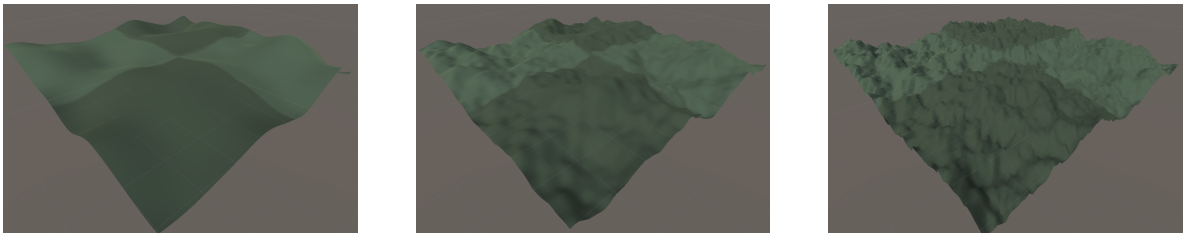
Figure 26: O_n effect on 1-D Perlin noise[1]

Accordingly, the lacunarity, l, controls the increase in frequency between octaves; an increase in lacunarity results in an increase in frequency between octaves. This is

characterised by the relationship:

$$\forall x \in \{O_1, O_2, ..., O_n\}, f_x = l^{O_x}$$

Lacunarity, then, allows us to tune the increase in detail between octaves. We can see in Fig. 27, with four octaves, the effect of lacunarity on the generated terrain. We can see that as the lacunarity increases, the terrain becomes more 'spiky' due to the increased detail level.



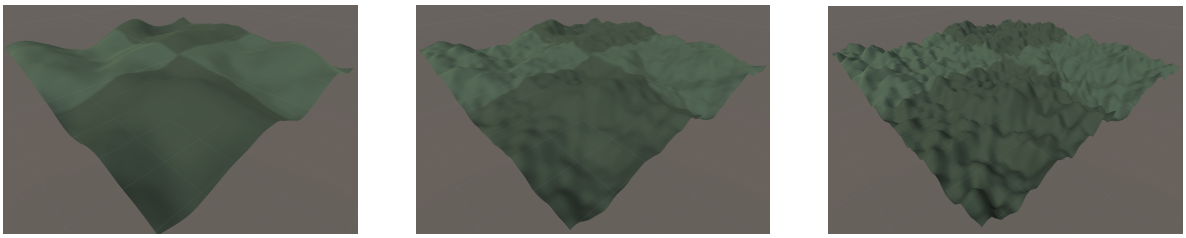
(a) $l = 1$ (b) $l = 2$ (c) $l = 4$

Figure 27: The effect of lacunarity on the generated terrain

Thus, lacunarity, denoted as l, governs the frequency increment across octaves. Specifically, a higher lacunarity value escalates the frequency from one octave to the next, embodying the formula:

$$\forall x \in \{O_1, O_2, ..., O_n\}, f_x = l^{O_x}$$

In essence, lacunarity fine-tunes the progression of detail from one octave to another. As illustrated in Figure 27, the influence of lacunarity on the terrain is evident over four octaves. An increase in lacunarity amplifies the terrain's detail, rendering it more rugged and intricately textured.



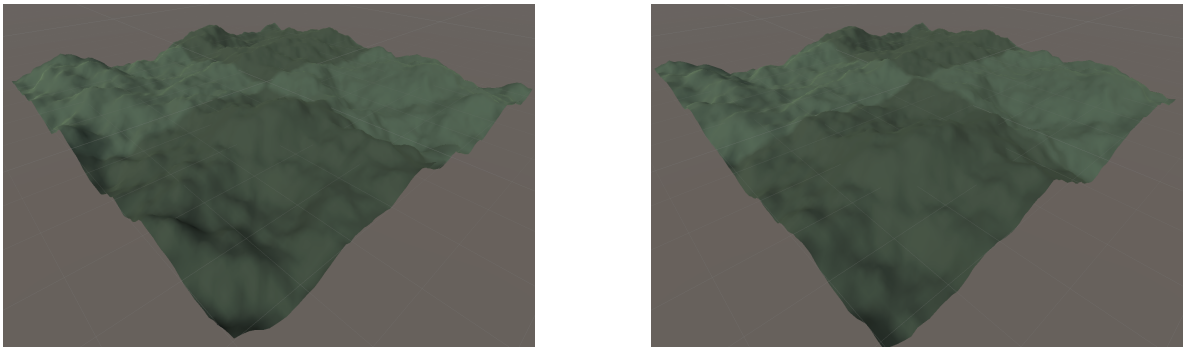
(a) $p = 0.2$ (b) $p = 0.5$ (c) $p = 0.8$

Figure 28: The effect of persistence on the generated terrain

Lastly, the scale parameter, s, dictates the granularity of detail within the waveform

when applied to the terrain model. The impact of scale on terrain formation is depicted in Figure 29, where an increase in scale leads to a 'zoomed out' appearance, rendering the details less notable.

An observable change occurs in the mountain ridge situated in the quadrant nearest to the camera, which becomes larger and more distant from the origin[3] as we transition from Figure 29a to Figure 29b. This effect highlights the scale's role in adjusting the detail level of the terrain's features.



(a) $s = 25$ (b) $s = 28$

Figure 29: The effect of scale on the generated terrain

When generating the terrain, we can use a seed value to ensure that the terrain generation is repeatable. This means that the same seed value will always produce the same terrain. This feature is useful for debugging, testing, and ensuring consistency across multiple runs of the simulation.

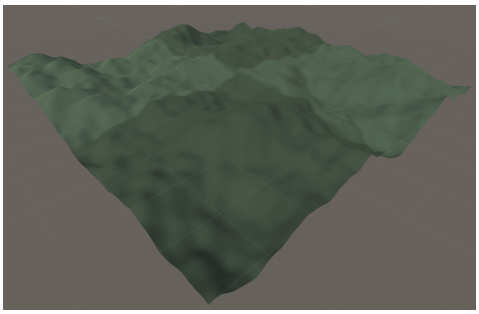
After refining our parameters through empirical testing and insights from [29, pg.3-4], we've finalised our terrain mesh, shown in Figure 30a. The terrain mesh is a crucial addition to the simulation, as it allows us to test the agents' ability to avoid obstacles. We will discuss the implementation of terrain avoidance in the next section.

3.3.3 Terrain Avoidance

Our final additional behaviour, then, will be to implement terrain avoidance. This will allow the agents to avoid the terrain mesh and will be a crucial addition to the agents' autonomy. We will discuss the implementation of this below.

At present, we do not have a method to detect terrain. Common implementations

[3]Here, the origin refers to the top-left quadrant, positioned furthest from the camera, on the terrain mesh.



(a) Final Terrain Mesh

Parameter	Value
Number of Octaves	4
Lacunarity	2
Persistence	0.5
Scale	25

(b) Perlin Noise Parameter Values

Figure 30

of terrain avoidance in UAVs include the use of LIDAR, radar, or cameras to detect the terrain in front of the UAV[27, 5].

Given the complexities and computational demands of simulating a LIDAR sensor within our framework, we opt for a more straightforward yet effective solution: a singular ranging sensor. This sensor will project a ray downward from the agent, measuring the distance to the terrain at each simulation frame.

However, this method does impose limitations, particularly a reduced field of view. Unlike many practical applications that offer a 360-degree view, our singular sensor restricts detection to directly beneath the agent. To circumvent terrain obstacles, we propose a method that triggers an upward thrust, u to cause an increase in the agent's local y position, a_y, if the distance between the agent and the terrain is less than a given threshold, t. We can define this behaviour as follows:

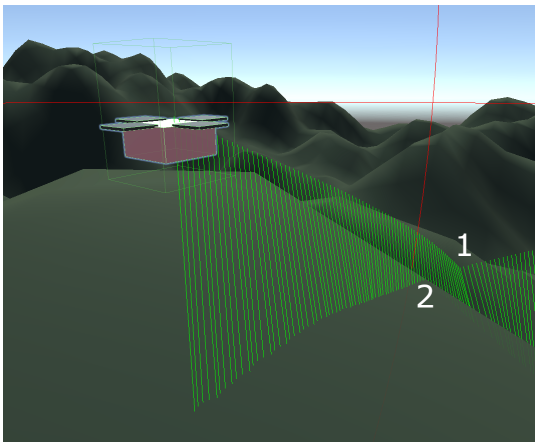
$$\vec{E} = \begin{bmatrix} 0 \\ U \\ 0 \end{bmatrix}, U = \begin{cases} \vec{0} & \text{if } a_y > t \\ u & \text{if } a_y \leq t \end{cases} \tag{23}$$

For initial testing, we set the threshold, t, to 5m and the upward thrust, u, to 10. We will adjust these values through empirical testing to ensure the agents can avoid the terrain effectively.

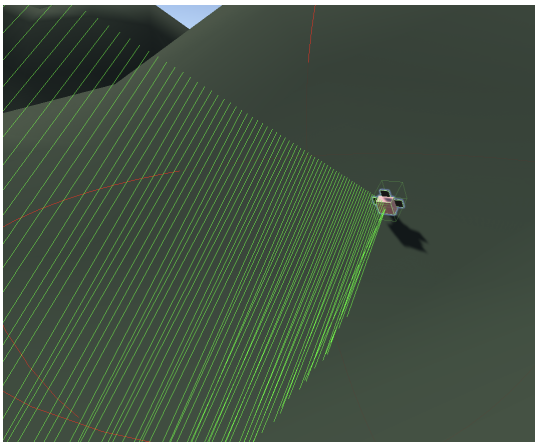
This behaviour is illustrated in Figure 31a, where we observe an agent nearing a steep hill. Initially, up to point 1, the agent maintains its altitude without any adjustments. However, as it approaches point 1 and casts a ray downward, it detects the terrain is closer than the set threshold, resulting in an increase in altitude to avoid the hill. Point 2 marks the hill's crest. Here, the agent realises the gap between itself and the terrain still falls short of safety margins, leading to a further increase in altitude to navigate

the obstacle safely.

Further development of this scenario is shown in Figure 31b. Despite recognising the impending terrain, the agent's substantial forward momentum and its resulting tilt impede an effective manoeuvre, leading to a situation where it struggles to clear the hill.



(a) Agent approaching a steep hill



(b) Agent failing to avoid terrain

Figure 31: Naive Terrain Avoidance

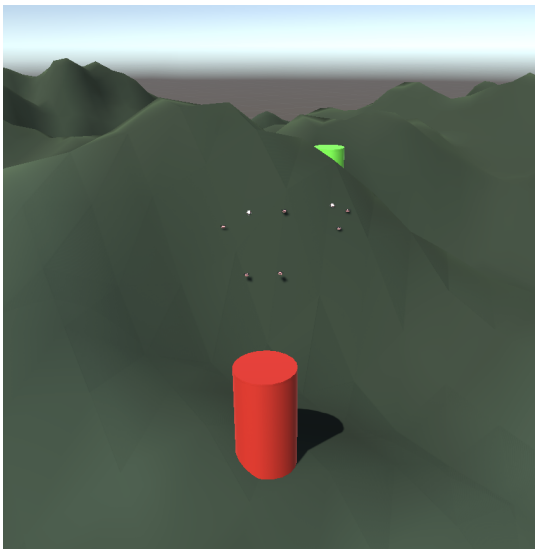
Our observations highlight deficiencies in the current approach to terrain avoidance, necessitating a refined strategy. To improve the agents' capability for navigating terrain, we propose adjusting the alignment of the ranging sensor to tilt slightly in the direction of the agent's velocity. This adjustment will enable the agent to detect terrain not only directly below but also ahead, significantly enhancing its ability to anticipate and circumvent obstacles.

However, the extensive scale of the terrain presented substantial challenges in optimising avoidance behaviours. A notable limitation in the current framework is the agents' difficulty with steep and large hills, primarily because their avoidance strategy is limited to increasing altitude. This approach is far from ideal; agents should ideally navigate around obstacles rather than merely over them. We intend to explore alternative strategies in Section 6.3 to overcome this issue.

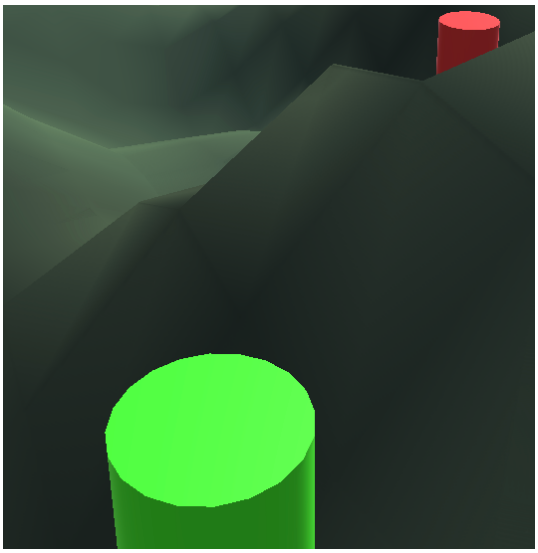
Alongside these changes in terrain interaction, modifications to how the simulation commences and concludes are also planned. We will introduce two platforms: a red start platform and a green end platform. The start platform is a flat area with a 4-metre radius, situated at the origin, while the end platform, also with a 4-metre radius, will be located at a random point within a 100-metre radius of the origin.

Agents will start their journey on the start platform and aim to reach the end platform, as illustrated in Figure 32. These adjustments are designed to provide a clear start and end point for the agents, enhancing the structure and coherence of the simulation.

With behaviours established for agents to navigate complex terrain, our next step is to optimise these behaviours for greater autonomy. This marks a crucial phase, focusing on improving agent independence and efficiency in navigating with minimal oversight, advancing our project towards a more sophisticated and autonomous simulation environment.



(a) Start Platform



(b) Goal Platform

Figure 32: Start and Goal Platforms in a single simulation

4 Methodology

Building on the foundational discussions in Section 1.1 regarding the expansive applications of UAV swarms, we move towards a targeted exploration of optimising the behaviour of the agents. Acknowledging the diverse operational demands on UAVs, we introduce three specific scenarios designed to mirror real-world challenges. These scenarios aim to evaluate the swarms' agility, efficiency, and adaptability under various conditions.

To ensure a comprehensive assessment, we've established a set of metrics. These will critically inform the development of an advanced cost function targeted for optimisation. This step marks a significant stride towards refining UAV behaviours, ensuring their effectiveness across a range of applications and edging us closer to realising the extensive capabilities of autonomous UAV swarms.

4.1 Scenario 1: Shortest First Arrival Time

Firstly, we will consider the scenario where reaching the goal in the shortest time is the most important factor. This is a present requirement for time-sensitive situations, where the redundancy of multiple agents is used to ensure that the goal is reached as quickly as possible. In essence, it means agents can take more risks in their behaviour to achieve the objective.

Some example situations where this scenario would be of use include military operations in hostile environments, gaining situational awareness in disaster zones, and surveying hazardous environments[30]. For the simulations of this scenario, we will consider the following metrics:

First Arrival Time We must consider the time taken for the first agent to reach the goal; it is the primary objective of the agents in this scenario. Specifically, this is the time taken for the first agent to enter the goal radius as defined in Section 3.3.1. We define a simulation run time, which will be the length a simulation is allowed to run for.

For the purposes of normalisation, we will use the time taken for the first agent to reach the goal, divided by the simulation run time. This will give us a value between 0 and 1, where a lower number indicates a better performance.This is the primary objective of the agents in this scenario, so we will give it a larger weight in our cost

function.

Proportion of Agents Arriving A simulation run where more agents arrive at the goal is more successful than one where fewer agents arrive. Furthermore, a simulation run where no agents arrive would be considered a failure.

We will then, also consider the proportion of agents that do not arrive at the goal by the simulation run time. This will give us a value between 0 and 1, where a lower number indicates a better performance. This is a secondary objective of the agents in this scenario.

4.2 Scenario 2: Reduced Spread (Cohesivity)

This scenario aims to reduce the spread of the agents in the flock. This is important in situations where the agents must maintain a close formation, such as in search and rescue operations, or in the delivery of goods. A simulation run's success in this scenario will be dependent on two metrics:

Standard Deviation of Arrival Times We will consider the times at which the agents arrive at the goal, for all agents that arrive before the simulation run time. We will then calculate the standard deviation of these times. A lower standard deviation indicates that the agents are arriving at the goal more uniformly. We will give this metric a larger weighting in our cost function.

A key issue with this metric is that it does not consider the method by which the agents arrive at the goal. For example, if all agents arrive at the goal at the same time, the standard deviation will be 0. This does not necessarily imply that the agents are in a close formation.

First Arrival Time As with the first scenario, we still consider the time taken for the first agent to reach the goal. A run where all agents reach the goal with a low standard deviation between them, but take a long time to do so is suboptimal. We will give this metric a smaller weighting in our cost function.

Proportion of Agents Arriving As with the first scenario, we will consider the proportion of agents that do not arrive at the goal by the simulation run time. We

do this to ensure that the cost function accounts for the number of agents within an arriving group.

4.3 Scenario 3: Reducing Collision Numbers

This scenario aims to reduce the number of collisions between agents and between agents and other objects in the environment. This is important in situations where the agents must operate in close proximity to each other, such as in the delivery of goods, or the area monitoring. We will consider both the arrival time of the first agent and the proportion of agents that arrive at the goal, as in the previous scenarios, to a small extent, for similar reasons as justified previously.

We will also consider the following metrics:

Number of Collisions We will consider the number of collisions that occur between agents and between agents and objects in the environment. A simulation run where fewer collisions occur is more successful than one where more collisions occur. We will give this metric a larger weighting in our cost function.

5 Optimisation

Earlier, we utilised an empirical method to optimise our initial three behaviours. However, we note that for each extra parameter we introduce, the complexity of the parameter space increases exponentially. Consider an example in which each parameter is bounded by integers within the inclusive range $(0, 10)$. With 3 parameters, this is a three-dimensional parameter space with $11^3 = 1331$ possible combinations. With 5 parameters, this is a five-dimensional parameter space with $11^5 = 161051$ possible combinations. This is a combinatorial explosion, and it is not feasible to optimise the parameters in this way.

In the quest for more scalable optimisation techniques, we explore two prominent probabilistic methods. The first is genetic algorithms, inspired by natural selection, which have shown promise in the field of swarm robotics behaviour optimisation. A significant study by Alaliyat et al. [19] demonstrated the application of a genetic algorithm to optimise a Boids model, though it is notably computation-intensive.

An alternative approach is the hill climbing algorithm—more accurately termed gradient descent in our context, aimed at minimising the cost function. This method iteratively refines a solution by adjusting a single parameter incrementally. While less computationally demanding than genetic algorithms, hill climbing risks getting trapped in local minima, potentially missing the global minimum.

Given these considerations, our focus shifts to exploring an alternative optimisation strategy that can efficiently navigate the expansive parameter space to identify optimal settings.

5.1 Simulated Annealing

The method we will use to optimise the parameters is simulated annealing. This is a probabilistic method that is used to find the global minimum of a function within a given search space. It is based on the physical process of annealing, where a material is heated and then cooled slowly to remove defects and reduce the material's energy. It is, in principle, a more advanced version of the hill climbing algorithm which can overcome local minima.

The simulated annealing algorithm begins its search from a randomly selected point within the search space, proceeding to explore the neighbouring points. For each neighbour, the algorithm evaluates the cost function, denoted as t. Movement to this new

point from the current position is determined by comparing their cost functions. When the new point offers a lower cost, indicating an improvement, the algorithm naturally transitions to it. Conversely, if the new point's cost is higher, suggesting a step back, the algorithm may still move to this point with a probability influenced by the cost function increase, $|c(t) - c(t+1)|$, and a temperature parameter, T. This probabilistic acceptance is key for the algorithm's ability to navigate out of local minima by occasionally accepting worse solutions temporarily.

The method of temperature reduction, or the cooling schedule, plays a pivotal role in the success of the algorithm. The temperature parameter T dictates the algorithm's readiness to accept suboptimal moves; as T decreases, the algorithm becomes increasingly picky, reducing the probability of accepting higher-cost solutions.

The cooling schedule must strike a careful balance: a rapid reduction in temperature can lead the algorithm to settle prematurely on local minima, while too slow a decrease can result in an inefficient search that takes too long to converge to an optimal solution. Adopting a linear cooling schedule allows the temperature to decrease steadily with each iteration, aiming to guide the algorithm towards global optimality with an effective balance between exploration and exploitation.

We will define our acceptance function, $P \in (0, 1)$ as follows:

$$P = \begin{cases} 1 & \text{if } c(t+1) < c(t) \\ e^{-\frac{c(t+1)-c(t)}{T}} & \text{if } c(t+1) \geq c(t) \end{cases} \tag{24}$$

The exponential term in the simulated annealing algorithm's acceptance function draws inspiration from the Boltzmann distribution - a concept from statistical mechanics. This distribution helps predict how likely a system is to be found in a certain state, based on its energy level. The formula for the Boltzmann distribution is:

$$P(i) = e^{-\frac{E_i}{kT}} \tag{25}$$

Here, $P(i)$ represents the likelihood of the system occupying a state with energy E_i, given a temperature T and the Boltzmann constant k. This equation essentially tells us that at higher temperatures, a system is more likely to be found in states of higher energy due to particles moving more vigorously and exploring a broader range of states.

Translating this concept into the realm of simulated annealing and optimisation, we can think of the 'energy' of a system as analogous to the cost of a solution; the lower the energy, the better the solution. The 'temperature' in our algorithm simulates

the physical temperature, controlling how willing the algorithm is to explore solutions of varying quality. At high temperatures, similar to particles in a heated system, the algorithm is more open to exploring less optimal solutions, increasing the chance of escaping local minima. As the temperature cools, the algorithm becomes more discerning, progressively favouring solutions that lower the "energy" or improve the optimization, guiding it towards the global minimum.

This clever use of mechanics principles in a computational algorithm allows for a more dynamic and effective search for optimal solutions, embodying the idea that sometimes, taking a step back (or exploring higher-cost solutions) can ultimately lead to finding the best path forward.

We can then define our parameters and our cost function. As noted previously in Section 3.2.4, the parameter space can be defined as a linear combination of the behaviours and their weights. For clarity, we now redefine this in terms of our extended parameter space, which includes goal-seeking and terrain avoidance behaviours.

$$\vec{V} = \alpha \begin{bmatrix} x_{avoid} \\ y_{avoid} \\ z_{avoid} \end{bmatrix} + \beta \begin{bmatrix} x_{align} \\ y_{align} \\ z_{align} \end{bmatrix} + \gamma \begin{bmatrix} x_{cohere} \\ y_{cohere} \\ z_{cohere} \end{bmatrix} + \delta \begin{bmatrix} x_{seek} \\ y_{seek} \\ z_{seek} \end{bmatrix} + \epsilon \begin{bmatrix} x_{terrain} \\ y_{terrain} \\ z_{terrain} \end{bmatrix} \tag{26}$$

We can then define our cost function, which is a linear combination of the metrics defined in Section 4. We will define a singular, modular cost function to allow for the easy addition of new metrics and the ability to tweak the weighting of the existing metrics.

The cost function will normalise the given weights, and the given metrics will also be normalised to ensure the output of the cost function lies in the range $(0, 1)$. Specifically, we define the normalisation of the four weights as:

$$\forall w_i \in W, w_i = \frac{w_i}{\sum_{i=1}^{|W|} w_i} \tag{27}$$

We also define the normalised metrics as follows:

First Arrival Time We can normalise this value by dividing it by the simulation time (the length the simulation is allowed to run for). Hence,

$$m_1 = \frac{\text{firstArrivalTime}}{\text{simulationRuntime}} \tag{28}$$

Proportion Reached In a similar nature to the normalisation method for the *first arrival time* metric, we can divide the number of agents that have not reached the goal by the simulation time by the total number of agents in the simulation, namely:

$$m_2 = \frac{\text{numberOfAgents} - \text{numberOfAgentsReached}}{\text{numberOfAgents}} \tag{29}$$

Number of Collisions Normalising a metric that has, in theory, no upper bound, is a challenge. In this case, we will look at fitting the data to a bounded function. The *logistic function* is a good candidate for this. The logistic function is defined as:

$$f(x) = \frac{1}{1 + e^{-k(x-x_0)}} \tag{30}$$

Specifically, we can set the midpoint value of the function, x_0, to be the mean of the number of collisions over an example set of simulations, such that $f(x_0) = 0.5$. We determined this to be ≈ 40 collisions. We can then set the steepness of the function, k, to be 0.1. This will give us a function that is bounded between 0 and 1 and is steep enough to give a good approximation of the number of collisions.

Standard Deviation of Arrival Times An upper bound on the standard deviation, σ, of a population is the population range. We can then normalise the standard deviation of the arrival times by dividing it by the range of the possible arrival times, which is equal to the simulation runtime.

A caveat here is that if no agents arrive at the goal, the standard deviation will be 0, which would set the normalised value to 0. This is not ideal, as the agents have not arrived at the goal. We will consider this as an edge case and set the normalised value to 1 in this case. Hence, we can define the normalised standard deviation of arrival times as:

$$m_4 = \begin{cases} 1 & \text{if } \sigma = 0 \\ \frac{\sigma}{\text{simulationRuntime}} & \text{otherwise} \end{cases} \tag{31}$$

We can then represent our cost function as a function that returns a weighted sum of the normalised metrics:

$$C(W, M) = w_1 m_1 + w_2 m_2 + w_3 m_3 + w_4 m_4 \tag{32}$$

At this stage, it is important to note the disjoint nature of the parameters (inputs) and metrics (observations). The simulation is non-deterministic, due to small fluctuations in starting positions and randomised goal positions. To help mitigate this, we will introduce our simulation process in the next section.

5.2 Simulations

To refine our parameter settings effectively, we embark on a series of simulations for each parameter set, aiming to capture the average performance metrics across multiple runs. This approach helps mitigate the inherent randomness in simulation outcomes, providing a more robust basis for evaluating and refining our UAV swarm optimisation strategies.

Achieving the right balance in the number of simulations per parameter set is crucial. Too few simulations may lead to significant variance in the metrics, compromising the accuracy of our cost function. Conversely, conducting too many simulations can incur excessive computational overhead. To strike this balance, we initially opted for 10 simulations per parameter set, with the flexibility to adjust this number based on the insights gained from our initial simulation results.

Our parameter space, bounded between 0 and 20 for each parameter, is systematically explored with a step size of 0.5. This granularity allows us to thoroughly sample the parameter space, ensuring comprehensive coverage without overwhelming computational resources. In defining the neighbourhood of a parameter set, we consider adjustments of ± 0.5 for each parameter, facilitating an efficient exploration of nearby solutions.

As for the cooling schedule employed in simulated annealing, we tailor it to the remaining simulations to be executed. For instance, in a scenario involving 500 iterations comprising 10 simulations each, the initial temperature is set to 500, decreasing by 1 after each simulation. This linear reduction ensures a gradual cooling process, effectively guiding the algorithm towards optimal solutions while efficiently utilising computational resources.

5.3 Results

A varying number of simulations were run for each scenario. In this section, we explore the results and discuss the potential reasons for the outcomes.

5.3.1 Scenario 1: Shortest First Arrival Time

Within this scenario, we meticulously conducted 821 valid iterations, complemented by several strategic restarts to ensure the robustness of our findings. Such restarts are paramount in simulations to prevent potential biases or anomalies within the data set, thus enhancing the reliability of the results. The core objective of this scenario is to optimise the swarm's efficiency in achieving the designated target in the shortest possible time. This goal is crucial for the practical deployment of UAV swarms in scenarios such as emergency response or rapid reconnaissance, where time is of the essence.

The analysis hinges on the detailed observation of the cost function's behaviour throughout the iterations. As illustrated in Fig. 33, a notable observation is the appearance of a global minimum around iteration 300. A closer inspection within this range reveals a gradual reduction in the cost function over time, indicative of the algorithm's effective optimization process.

Interestingly, around iteration ≈ 250, there is a discernible increase in the cost function. This phenomenon is attributed to the algorithm's strategic exploration of the parameter space, allowing it to escape a potential local minimum. Such exploration is instrumental in the discovery of a more optimal solution, underscoring the algorithm's adaptive capability in navigating the complex parameter landscape.

Analysing the weight assignments indicated in Table 11 reveals that the seeking behaviour is accorded the highest weight. This prioritisation aligns with the scenario's primary objective, where the swift attainment of the goal location is prioritised.

Additionally, the terrain avoidance behaviour is also assigned a high weight, reflecting the necessity to navigate complex terrain meshes efficiently. In contrast, the alignment and cohesion behaviours are assigned lower weights. This weighting strategy underscores the relatively lesser importance of these behaviours in the context of this specific scenario, where the primary focus is on expedient target location rather than the maintenance of formation or collective movement.

5.3.2 Scenario 2: Reduced Spread (Cohesivity)

A total of 1214 runs were completed for this scenario, again, with several restarts throughout. The cost function over the iterations is shown in Fig. 34. The zoomed-in section shows the global minimum at the 226th iteration, of which the parameter values are detailed in Table 12.

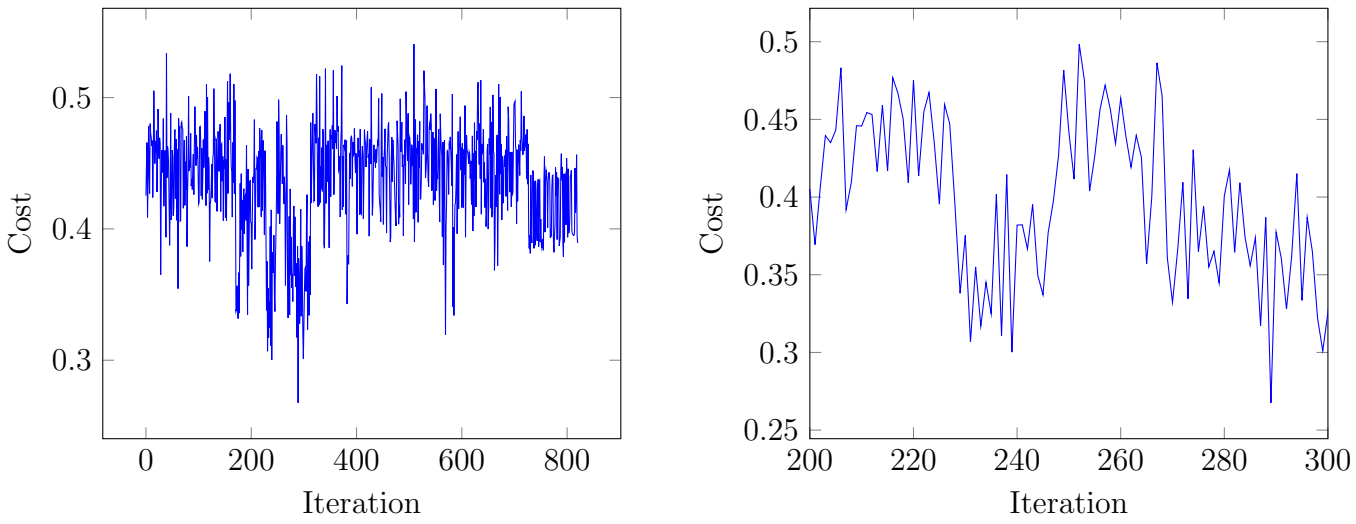


Figure 33: Cost Function over Iterations for Scenario 1

Parameter	Weight
Avoidance	8.5
Alignment	2.5
Cohesion	0.5
Seeking	17
Terrain Avoidance	16.5
Cost	0.268

Table 11: Optimal Parameters for Scenario 1

Fig. 34 shows some peculiar behaviour with the cost function, namely, that it appears to consistently fluctuate between two collections of values, centred at ≈ 0.37 and ≈ 0.31. We also notice that the general trend of the cost function before approaching the determined global minimum is not strongly negative.

To attempt to mitigate this, the simulation environment was tweaked, increasing the number of simulations per iteration from 10 to 100. The results for this set of simulations are shown in Fig. 35. These results indicate the problem persists, but the outliers present from the previous runs (where the cost function evaluated to a value not centred around the poles at $\approx 0.31, 0.37$) have been reduced.

To delve deeper into what was causing this, the metrics themselves were analysed for a set of iterations. Fig. 36 shows the metrics over one set of annealing. We would expect the values of both metrics to decrease over time, as the cost function is minimised. However, while we can see the variance of the *first arrival time* metric

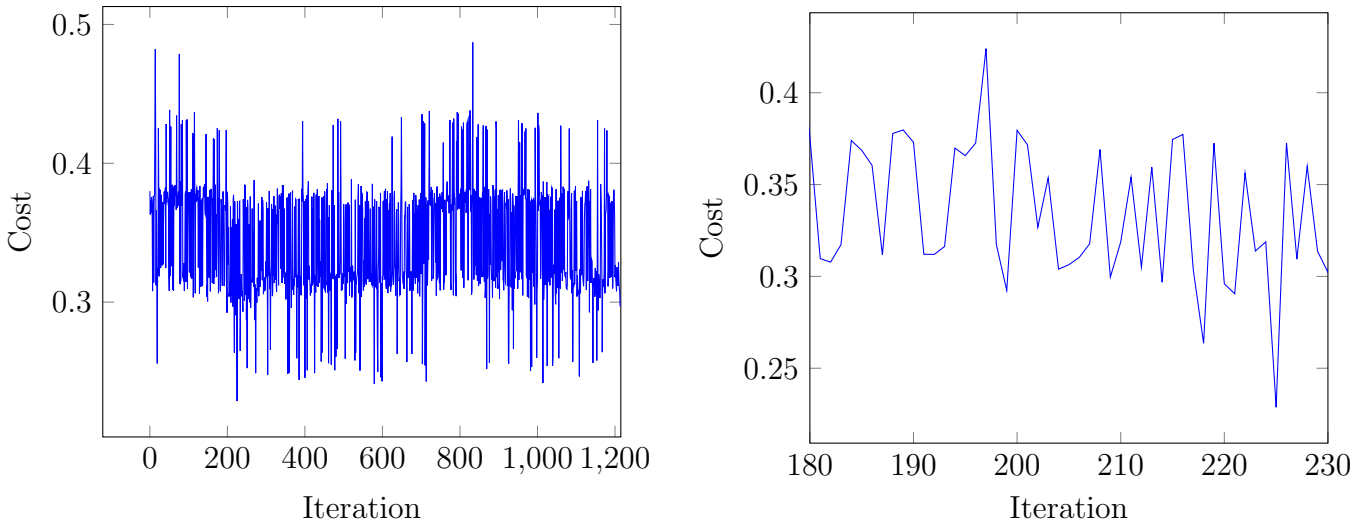


Figure 34: Cost Function over Iterations for Scenario 2

Parameter	Weight
Avoidance	12.5
Alignment	4
Cohesion	15
Seeking	18.5
Terrain Avoidance	0.5
Cost	0.229

Table 12: Optimal Parameters for Scenario 2

decrease over time, as expected, due to the temperature decrease in the algorithm, the *standard deviation of arrival times* metric does not show a clear trend.

This is likely the cause of the fluctuating cost function, and a key reason explaining why this behaviour was not present in the results of the first scenario. We will hypothesise why this occurs in Section 5.5.

5.3.3 Scenario 3: Reducing Collision Numbers

A total of 508 runs were completed for this scenario, with several restarts. The cost function over the iterations is shown in Fig. 37. The zoomed-in section shows the global minimum at the 186th iteration, of which the parameter values are detailed in Table 13.

The results for this scenario again show peculiar behaviour. It would be expected for the avoidance behaviour to have a higher weight than the alignment and cohesion

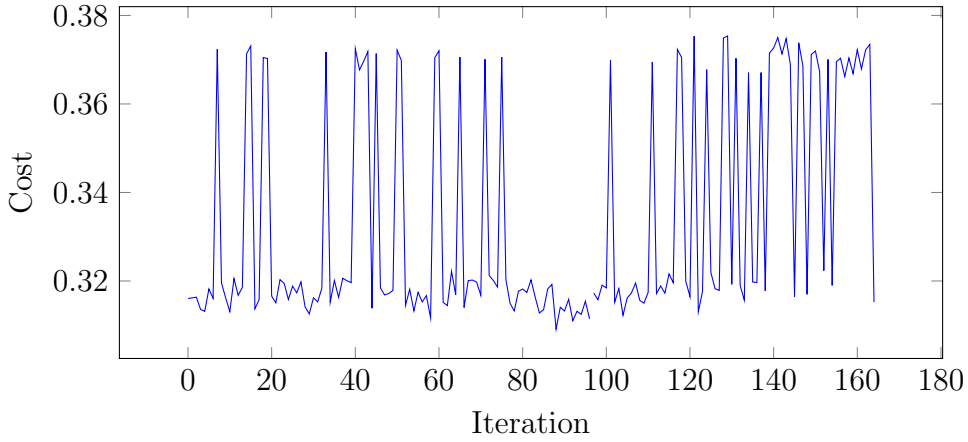


Figure 35: Cost Function over Iterations for Scenario 2 with 100 Simulations

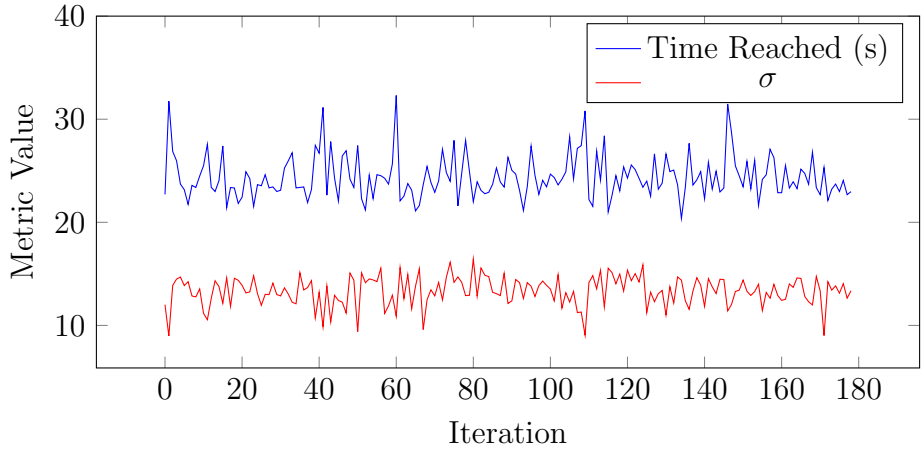


Figure 36: Metrics over Iterations for Scenario 2

behaviours, given the primary objective of the agents in this scenario. However, the alignment and cohesion behaviours have higher weights. We will discuss this in Section 5.5.

5.4 Sensitivity Analysis

We will now consider the sensitivity of the cost function to the weights of the metrics by exploring the nearby parameter space of the optimal parameters found in the previous section. We will do this by varying the weights of the metrics by ± 0.5 and observing the change in the cost function. We will do this solely for the first scenario, as the results for the other scenarios are less intuitive.

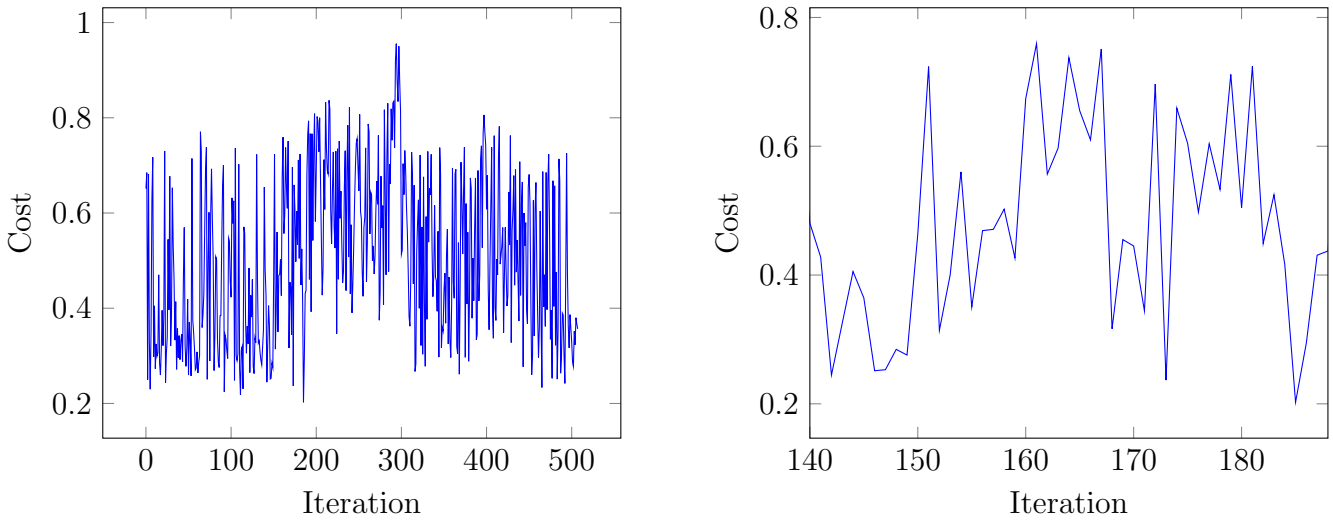


Figure 37: Cost Function over Iterations for Scenario 3

Parameter	Weight
Avoidance	3
Alignment	15
Cohesion	15.5
Seeking	4.5
Terrain Avoidance	2.5
Cost	0.202

Table 13: Optimal Parameters for Scenario 3

Fig. 38[4] shows the sensitivity of the cost function to the weights of the metrics. Visually, opposite axes in the spider diagram show the polarity of the change. The lengths of the axes represent the difference between the cost function at the optimal parameters and the cost function at the parameters with the weights of the metrics increased or decreased by ± 0.5.

This results in a 'topological' view of the sensitivity of the cost function to the weights of the metrics at the point of the determined global minimum. We can see that this scenario seems sensitive to changes in the alignment and avoidance behaviours most. On the contrary, it seems less sensitive to the seeking behaviour. To this end, we have likely found an optimal set of parameters in the more sensitive dimensions.

It is vital to note that the values of the sensitivity analysis may be dependent on

[4]The cohesion metric is not modified in the negative direction, as it is already set at the minimum value of 0.5.

external factors. Namely, the nondeterministic nature of the simulation may cause the sensitivity of the cost function to the weights of the metrics to vary; the results are likely to be different if the simulation was run a different number of times.

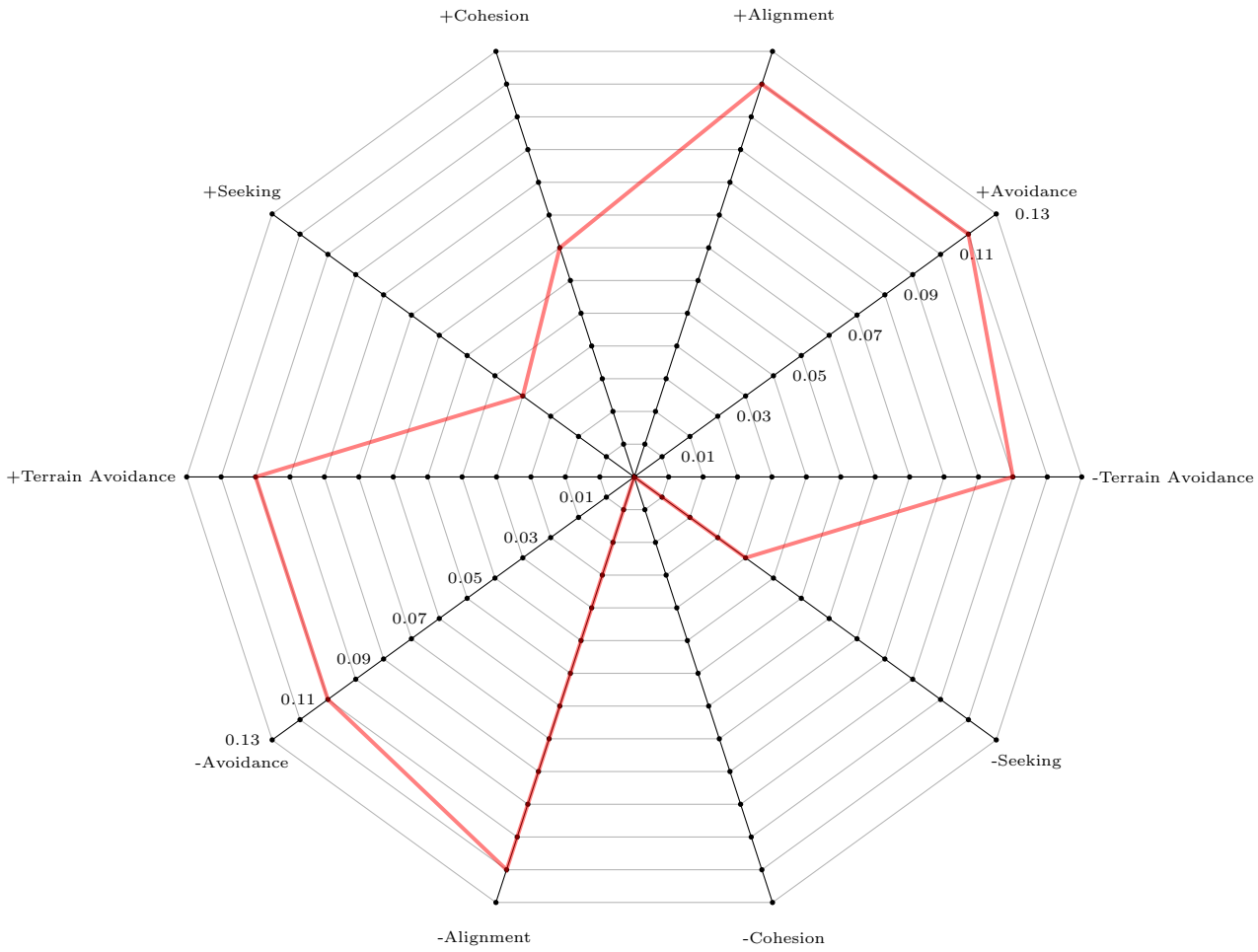


Figure 38: Sensitivity Analysis

5.5 Discussion

The results of the simulated annealing algorithm show that the algorithm is able to find optimal parameters for the given scenarios. However, the results also show some peculiar behaviour, particularly in the second and third scenarios. Whilst sufficient solutions were found, we should consider why these solutions are counterintuitive.

5.5.1 Parameter-Metric Relationship

The process of optimising a deterministic space, especially where the relationship between the input parameters and the metrics used within the cost function is clear, by simulated annealing is not challenging.

When evaluating the optimisation of a simulation environment, Deng [31] notes that the relationship between the cost function[5] as determined by the analysis of a simulation and the input parameters is often inaccurate, leading to complications in the optimisation process. In simulations, where the dynamics are complex and influenced by a myriad of factors, even a small deviation in input parameters can lead to disproportionately large changes in the metrics, and thereby, the cost function.

Adding to the complexity is the role of noise within the simulation environment. Noise, in this context, can arise from various sources, including the inherent randomness of the simulated processes, limitations in computational accuracy, or external factors not accounted for within the model. This noise can obscure the relationship between input parameters and metrics, making it difficult to discern genuine optimisation paths from artefacts introduced by the simulation's stochastic nature.

Namely, the relationship between the *standard deviation of arrival times* metric and the observation of the determined 'optimal' input parameters for scenario 2 seems unclear, as reinforced by the values of the metric seen in Fig. 36.

To tackle these challenges, two potential pathways emerge: adjusting the metrics or refining the parameter space. Adjusting the metrics to better align with the observable outcomes of the parameter space might offer a more straightforward path to optimization but could risk oversimplifying the model or losing sight of critical dynamics. On the other hand, refining the parameter space to better suit the metrics involves a deeper reconsideration of what parameters are most influential and how they can be more precisely controlled or varied within the simulation.

5.5.2 Collision Metrics

In the analysis of the third scenario's optimisation results, an intriguing pattern emerges: the alignment and cohesion behaviours are assigned higher weights compared to the avoidance behaviour. This finding appears counterintuitive given the scenario's primary goal of minimising collisions among agents. Intuitively, one might anticipate the

[5]Deng calls the cost function the 'objective function' in their paper.

avoidance behaviour, which directly contributes to collision reduction, to be prioritised over alignment and cohesion behaviours in the weighting scheme.

In Section 5.1, we introduced a method of normalising the number of collisions metric to fit within a bounded function. This raises two potential explanations for the observed anomaly: either the collision metric's weight within the cost function is undervalued, or the normalisation process fails to accurately represent the data's distribution.

The level of complexity in tweaking the simulation environment to achieve expected results is high, which may cause some of the issues present in the results for the latter two scenarios. We will explore this further in the next section.

6 Improving Autonomy

Here, we introduce a further extension to the Boids algorithm, discuss the complexity of the parameter space and discuss possible solutions to the issues arising from the optimisation of the parameter space in the previous section.

6.1 Boids: Introducing Obstacle Avoidance

In an update to his original description of the Boids algorithm, Reynolds [32] introduces predictive obstacle avoidance (and an implementation of goal seeking). We can utilise almost an identical copy of the *avoidance* behaviour, modifying it to avoid randomly generated 'obstacles' in the environment, meant to represent buildings. We also set the avoidance radius to be wider to represent the difference in scale between agents and obstacles.

We define the obstacle avoidance behaviour for an agent of position a with the set of nearby, detectable obstacles, $\vec{o} \in R_o$, as follows:

$$\vec{O} = \begin{cases} 0 & \text{if } R_o = \emptyset \\ \frac{1}{|R_o|} \sum_{o \in R_o} \vec{o} - \vec{a} & \text{otherwise} \end{cases} \tag{33}$$

The obstacles themselves are generated as identical meshes to the start and goal platforms described earlier but with random scales across all three axes within the ranges shown in Table 14. A scale of 1 on all axes represents a cylinder of radius 1 and height 2.

Axis	Range
x	(3, 8)
y	(5, 20)
z	(3, 8)

Table 14: Ranges for Obstacle Generation

125 obstacles were chosen to be spawned within an area bounded by 125 metres in each axis, which is slightly further than the distance of the goal platform at 100 metres. This replicated a semi-dense city-like environment, which can be seen in Fig. 39. The obstacles are indicated by the colour blue.

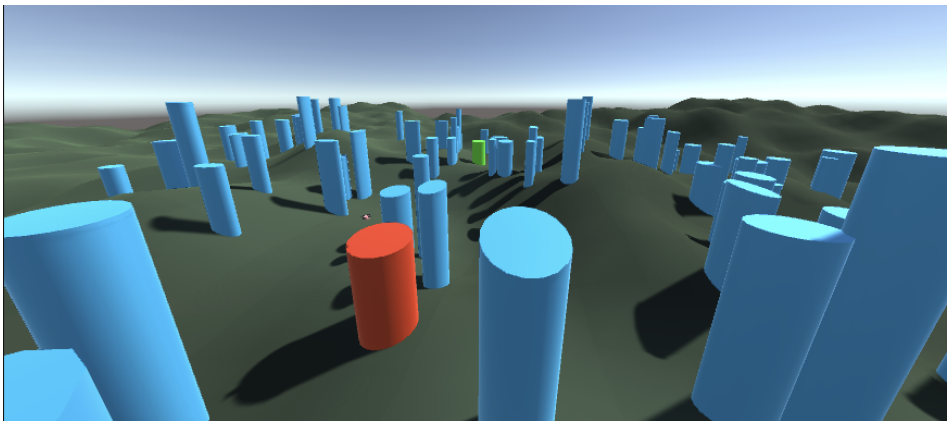


Figure 39: Obstacles in the Environment

6.2 A Simplified Model

We now note that we have six behaviours with the aim of traversing a complex three-dimensional space. As noted earlier, the complexity of the parameter space increases exponentially with each additional parameter. Additionally, the relationship between the parameters and the emergent behaviour cannot be easily characterised, leading to difficulties in optimisation.

We now aim to reduce the complexity of the parameter space in the hopes of strengthening the link between the parameters and the emergent behaviour. For clarity, we reiterate the six behaviours as:

- Avoidance
- Alignment
- Cohesion
- Seeking
- Terrain Avoidance
- Obstacle Avoidance

6.2.1 Repulsion

We notice that the nature of 'avoiding' other objects in the simulation is shared across multiple behaviours. In theory, other agents could be considered as obstacles with a

smaller range of avoidance. Likewise, the terrain could be considered as an obstacle with a larger range of avoidance. We can then look at combining these three behaviours into a single 'repulsion' behaviour, with the general idea being that agents are repelled by other objects in the environment.

6.2.2 Attraction

On the contrary, the agents have some natural 'attraction' to two main parts of the simulation. Primarily, the agents are attracted to the goal platform at longer distances. At shorter distances, the agents are additionally attracted to the centre of the flock. We can then look at combining the 'seeking' and 'cohesion' behaviours into a single 'attraction' behaviour.

For now, we will not consider the 'alignment' behaviour in this simplification, as it is not directly related to the 'repulsion' or 'attraction' behaviours.

6.2.3 Combining Behaviours: Potential Fields

Essentially, we have reduced the six behaviours into two behaviours which, in essence, encapsulate similar information. This refinement suggests that optimisation efforts can be more focused, essentially targeting these two key parameters. This approach draws a parallel with the concept of artificial potential fields, a foundational idea in robotics that illustrates the interplay between an agent and its surroundings. In this framework, agents are simultaneously drawn towards specific points of interest and repelled from obstacles, navigating the environment by moving in the direction of the net force exerted upon them.

We define the potential field, U, as the sum of the repulsion and attraction fields:

$$U(x) = U_{rep}(x) + U_{att}(x) \tag{34}$$

We can then define the repulsion field as:

$$U_{rep}(x) = \begin{cases} \frac{1}{2}k_r(\frac{1}{\rho} - \frac{1}{\rho_0}) & \text{if } \rho < \rho_0 \\ 0 & \text{otherwise} \end{cases} \tag{35}$$

Namely, the repulsion field is a function of the distance between the agent and the obstacle, ρ, and the range of avoidance, ρ_0. The repulsion field is only active if the agent is within the range of avoidance. Similarly, we can define the attraction field as:

$$U_{att}(x) = \frac{1}{2} k_a |x - x_d|^2 \tag{36}$$

We see that the attraction field is a function of the distance between the agent and the goal, $|x - x_d|$. The attraction field is always active. Notably, we can define the resultant force acting on the agent as the gradient of the potential field. More specifically, the negative derivative of the potential field:

$$F_{rep}(x) = \begin{cases} \frac{1}{2} k_r (\frac{1}{\rho} - \frac{1}{\rho_0}) \frac{1}{\rho^2} \frac{\partial \rho}{\partial x} & \text{if } \rho < \rho_0 \\ 0 & \text{otherwise} \end{cases} \tag{37}$$

$$F_{att}(x) = -k_a (x - x_d) \tag{38}$$

In the equations outlined, the coefficients k_r and k_a represent the weights for the repulsion and attraction forces, respectively. These weights are crucial parameters that require optimisation to fine-tune the interaction between the agent and its surroundings. Obstacles within the simulation environment, which include both static terrain features and dynamic entities like other agents, are assigned a specific radius, ρ_0. This radius determines the threshold distance at which repulsion forces become active, thus preventing collisions by maintaining a safe distance between the agent and obstacles.

While it would be ideal to pre-calculate the potential fields and the resulting forces guiding agent movements, the dynamic nature of the simulation environment complicates this approach. Agents, serving as moving obstacles, constantly change the landscape of potential fields, rendering any pre-calculated forces obsolete within moments. Given this dynamic environment, calculating the resultant forces in real time becomes necessary. However, this approach is inherently computationally demanding, posing significant challenges, especially as the scale of the simulation environment and the number of agents increase.

To address these challenges, we propose an innovative method for online calculation of resultant forces that aims to minimise computational overhead. This method, which we will delve into in the following section, offers a practical solution for managing the complex dynamics of UAV swarms with greater efficiency and reduced computational resource requirements.

6.3 Local Potential Field Emergence

We can reduce the computational cost of calculating the resultant forces acting on the agents by only considering the forces acting on an agent within a certain radius. This is known as a *local potential field*. We can then define the resultant force acting on an agent as the sum of the forces acting on the agent within a certain radius, r, such that x_r defines the position of the agent and the influence radius r.

$$F(x_r) = F_{att}(x_r) + F_{rep}(x_r) \tag{39}$$

6.3.1 Two Dimensional Abstraction

Looking at the simulation environment along the x and z axes, we can produce a vector field to visualise some examples of what the force field may look like[6].

Consider the attractive force, which considers two poles of attraction, one at the goal platform at $[-8, -8]$ and one at the centre of the flock at $[2, 2]$.

Here, we describe a position as $p = [x, z]$.

We can then define the force field as:

$$F_{att}(p) = -k_{a1}(p - p_{goal}) - k_{a2}(p - p_{centre}) \tag{40}$$

We note that, because we represent multiple behaviours in the same attraction function, we have multiple weights, k_{a1} and k_{a2}, for the two poles of attraction. These are equivalent to the weight of behaviours in the Boids implementation. We note that we can keep them equal to reduce the complexity of the environment. However, to ensure the agents are attracted to the goal platform at longer distances, we can set $k_{a1} > k_{a2}$. We will not explore tuning these values independently, as they should not relate to the repulsive behaviours.

Fig. 40 shows the force field for the attraction behaviour. We note that the pole of the field lies between both the goal platform and the centre of the flock. This is expected, as the agents should be attracted to both points. We could set a maximum distance for cohesion attraction, but this is not necessary for this simulation, as over time, we can see that the flock will naturally move towards the goal, due to having a higher weight in the attraction function. This is evident in the centre of the field, which is closer to the goal platform than the centre of the flock.

[6]As an aside, we note that the force field is a vector field, and the potential field can be represented as a scalar field. We use the vector field here, as it is easier to visualise.

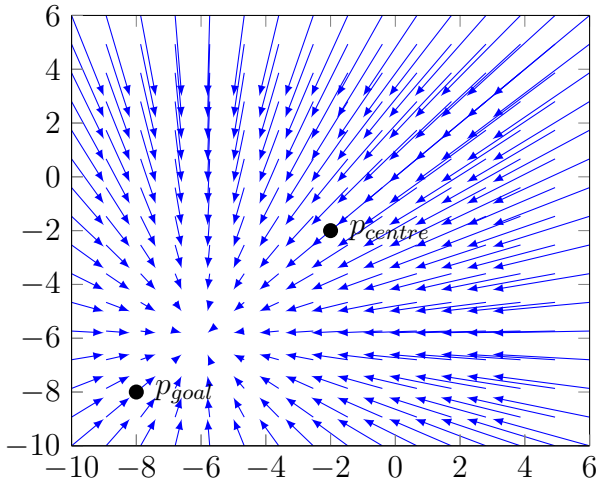


Figure 40: Attraction Force Field

We can do the same for our repulsive behaviours. We do not consider the terrain as an obstacle in two dimensions, so will solely consider other agents, τ, and the obstacles, ρ, in the environment. Hence, we can define the repulsive force field as:

$$F_{rep}(p) = \begin{cases} \frac{1}{2}k_r(\frac{1}{\rho} - \frac{1}{\rho_0})\frac{1}{\rho^2}\frac{\partial\rho}{\partial p} & \text{if } \rho < \rho_0 \cap \tau > \tau_0 \\ \frac{1}{2}k_r(\frac{1}{\tau} - \frac{1}{\tau 0})\frac{1}{\tau^2}\frac{\partial\tau}{\partial p} & \text{if } \rho > \rho_0 \cap \tau < \tau_0 \\ \frac{1}{2}k_r(\frac{1}{\rho} - \frac{1}{\rho_0})\frac{1}{\rho^2}\frac{\partial\rho}{\partial p} + \frac{1}{2}k_r(\frac{1}{\tau} - \frac{1}{\tau 0})\frac{1}{\tau^2}\frac{\partial\tau}{\partial p} & \text{if } \rho < \rho_0 \cap \tau < \tau_0 \\ 0 & \text{otherwise} \end{cases} \tag{41}$$

We can then see the resulting repulsive force field in Fig. 41, with obstacles located at $[2, 2]$ and $[5, 5]$, with $\rho_0 = 1$, and another agent located at $[2, 5]$, with $\tau_0 = 0.5$.

We can then combine these two fields to produce the resultant force field acting on the agents. We can then use this field to move the agents in the simulation environment.

The challenge with this model is that, whilst we have removed the complexity of the parameter space, we have introduced new parameters (such as the avoidance radius of the obstacles and other agents) which need to be optimised. However, unlike the arbitrary nature of the behaviour weightings, these parameters have a clear relationship with the behaviour of the agents in the environment.

We then have the parameters k_r, k_{a1}, k_{a2}, ρ_0, and τ_0 to optimise. With an extension into three dimensions, we would also have the parameter υ_0 to consider, which would define the terrain avoidance distance. We could then use the simulated annealing algorithm to optimise these parameters, with the cost function being the same as before,

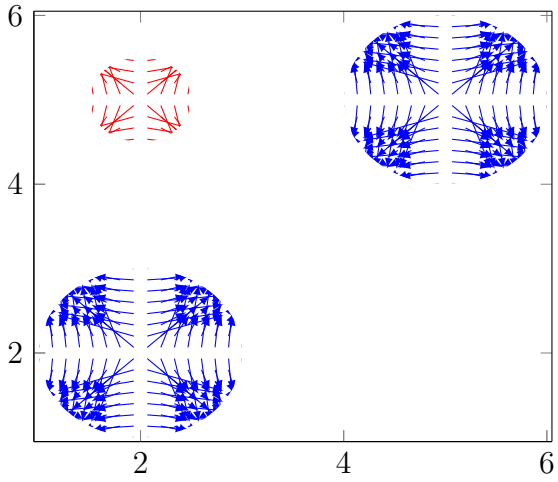


Figure 41: Repulsion Force Field

but with the new parameters. This was determined to be out of scope for this project, where this novel alternative model was introduced as a potential solution to the issues faced in the previous section.

6.4 Comparing Models

We can compare the Boids model with the LPFE model in terms of complexity and the relationship between the parameters and the emergent behaviour.

We note that the LPFE model is an extension of the Boids model, which, in theory, has more meaningful connections between the parameters and the observed behaviour. As such, we would expect the optimisation process to be more straightforward for the LPFE model than the Boids model. Specifically, we can expect the cost function to be more accurately characterised by the parameters in the LPFE model.

A comparison of the two models is shown in Table 15.

Feature	Boids	LPFE
Number of Behaviours	6	2
Number of Parameters	6	6
Complexity of Parameter Space	High	Low
Parameter-Metric Relationship	Innacurate	More Accurate
Ease of Implementation	Easy	Hard

Table 15: Comparison of Boids and LPFE Models

It is vital to note that as the LPFE model has not been implemented, the comparison

is theoretical.

7 Future Work

Whilst this project demonstrates the ability to use simulated annealing to optimise the parameters of a Boids simulation concerning given scenarios and the drawbacks of the method, there are several areas for future work. Namely, implementing the local potential field emergence model would be a good next step. We also explore increasing the complexity of the simulation to mirror real-life scenarios more accurately.

7.1 LPFE Implementation

Primarily, extending the model discussed in Section 6.3 to three dimensions and performing simulations to test the viability of the model is a natural next step.

We can define the model in three dimensions by considering the position of the agent as $p = [x, y, z]$. We can then define the force field as the sum of the attractive forces acting on the agent and the repulsive forces acting on the agent within a certain radius, r. We would also include the terrain as an obstacle in the environment, with a similar avoidance radius to the other obstacles, represented by υ_0, with the two force fields defined as:

$$F_{att}(p) = -k_{a1}(p - p_{goal}) - k_{a2}(p - p_{centre}) \tag{42}$$

$$F_{rep}(p) = \begin{cases} \frac{1}{2}k_r(\frac{1}{\rho} - \frac{1}{\rho_0})\frac{1}{\rho^2}\frac{\partial\rho}{\partial p} & \text{if } \rho < \rho_0 \cap \tau > \tau_0 \cap \upsilon < \upsilon_0 \\ \frac{1}{2}k_r(\frac{1}{\tau} - \frac{1}{\tau_0})\frac{1}{\tau^2}\frac{\partial\tau}{\partial p} & \text{if } \rho > \rho_0 \cap \tau < \tau_0 \cap \upsilon < \upsilon_0 \\ \frac{1}{2}k_r(\frac{1}{\rho} - \frac{1}{\rho_0})\frac{1}{\rho^2}\frac{\partial\rho}{\partial p} + \frac{1}{2}k_r(\frac{1}{\tau} - \frac{1}{\tau_0})\frac{1}{\tau^2}\frac{\partial\tau}{\partial p} & \text{if } \rho < \rho_0 \cap \tau < \tau_0 \cap \upsilon < \upsilon_0 \\ \frac{1}{2}k_r(\frac{1}{\rho} - \frac{1}{\rho_0})\frac{1}{\rho^2}\frac{\partial\rho}{\partial p} + \frac{1}{2}k_r(\frac{1}{\upsilon} - \frac{1}{\upsilon_0})\frac{1}{\upsilon^2}\frac{\partial\upsilon}{\partial p} & \text{if } \rho < \rho_0 \cap \tau > \tau_0 \cap \upsilon > \upsilon_0 \\ \frac{1}{2}k_r(\frac{1}{\tau} - \frac{1}{\tau_0})\frac{1}{\tau^2}\frac{\partial\tau}{\partial p} + \frac{1}{2}k_r(\frac{1}{\upsilon} - \frac{1}{\upsilon_0})\frac{1}{\upsilon^2}\frac{\partial\upsilon}{\partial p} & \text{if } \rho > \rho_0 \cap \tau < \tau_0 \cap \upsilon > \upsilon_0 \\ \frac{1}{2}k_r(\frac{1}{\rho} - \frac{1}{\rho_0})\frac{1}{\rho^2}\frac{\partial\rho}{\partial p} + \frac{1}{2}k_r(\frac{1}{\tau} - \frac{1}{\tau_0})\frac{1}{\tau^2}\frac{\partial\tau}{\partial p} + \frac{1}{2}k_r(\frac{1}{\upsilon} - \frac{1}{\upsilon_0})\frac{1}{\upsilon^2}\frac{\partial\upsilon}{\partial p} & \text{if } \rho < \rho_0 \cap \tau < \tau_0 \cap \upsilon > \upsilon_0 \\ 0 & \text{otherwise} \end{cases} \tag{43}$$

We would then perform simulations to test the viability of the model. This would involve testing the model in a variety of environments, with different obstacles and terrain, to ensure the model is robust. Once this has been completed, we can then look at optimising the parameters of the model, specifically k_r, k_{a1}, k_{a2}, using the simulated

annealing algorithm.

Following this implementation, we can then compare the LPFE model to the Boids model in terms of complexity and the relationship between the parameters and the emergent behaviour. This would allow exploration of the effectiveness of the LPFE model in comparison to the Boids model.

7.2 Other Models

The simulated annealing optimisation model is only one out of many methods of determining an optimal parameter set. Here, we discuss the set of possible alternative methods for optimisation in our specific problem.

7.2.1 Online Optimisation

The models we have explored have fixed parameters which are determined before a simulation run. One possible avenue of exploration is looking at online parameter optimisation. Some parameters are likely more important and as such, should have higher weightings, at different points in a simulation. For example, looking at the Boids model, we can expect the need for seeking to be higher the further away the agent is from the goal. Likewise, in life-like scenarios, we would expect the agents to have a stronger avoidance weight when they are expected to be closer together, such as when leaving the start area or near the goal area.

The lack of this behaviour meant collisions occurred more frequently near the goal platform in the simulated annealing process; this is seen in Fig. 42. As discussed earlier, we would expect the avoidance behaviour to have an inverse correlation with the number of collisions, thus, this may have thrown off the annealing process.

7.2.2 Neural Network Optimisaton

A more complex learning technique, the use of a neural network, would likely have the ability to converge to a global minimum more accurately. While the promise of neural networks is enticing, their integration into real-time simulation environments is hampered by substantial computational demands. Training neural networks, especially deep models, requires significant computational resources and time, which might not be feasible in scenarios requiring quick adaptations. Additionally, the implementation of

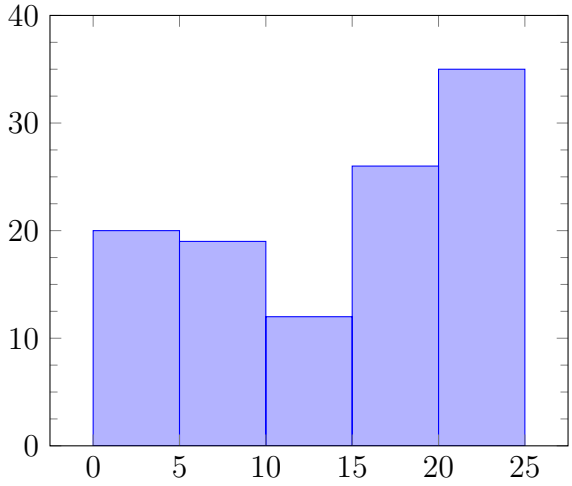


Figure 42: Number of Collisions (First agent reaching the goal at $t \approx 25$)

such models in a dynamically changing environment adds another layer of complexity due to the need for continual retraining or fine-tuning as new data becomes available.

The use of neural networks in the context of optimisation is not new; Caldwell et al. [33] introduce a method of utilising deep neural networks for combinatorial optimisation, which could be adapted to solving the parameter optimisation problem in this project. Notably, as mentioned earlier, optimisation problems require appropriate parameter selection - a difficult task which may be alleviated by the use of neural networks. Zhao et al. [34] use a *reward modulated spiking neural network* to allow a flock to learn to avoid obstacles, rather than using a pre-defined set of rules.

7.3 Hostility

An interesting extension to the Boids model is the introduction of hostile agents and environments. This would likely require a more complex model, as the agents would need to be able to detect the hostility and react accordingly. We discuss potential methods for this here.

7.3.1 Hostile Agents

We may consider the introduction of agents which are hostile to the flock. This could be implemented by having the hostile agents move towards the flock, with the flock needing to avoid them; a concept which has been explored with the Boids model[35].

Delgado-Mata et al. [36] modify the Boids model to include an 'escape' behaviour, where the flock moves away from a hostile agent. Additionally, the model uses the concept of 'fear contagion', where the flock is more likely to move away from hostility if neighbouring agents are moving away from said hostility.

An example implementation of this may include the introduction of a hostile agent with the seeking behaviour tailored to the centre of the flock. The flock would have a method of detecting hostile agents, similar to the obstacle avoidance behaviour. We may then choose to add the element of fear contagion, where an agent's fear is dependent on the fear of its neighbours. Namely,

$$F_{fear}(a) = -k_{fear}(a - a_{centre}) \tag{44}$$

An agent's fear may also be determined by the number of hostile agents within an agent's avoidance radius. Attempting to optimise this additional parameter would serve as a potential extension to the Boids model.

Additionally, the hostile agents could have the ability to attack the flock from a distance with projectiles, which would require the flock to avoid the projectiles as well as the agents. Incorporating the ability to detect and avoid projectiles would bring this model closer to real-life scenarios.

7.3.2 Hostile Environments

In addition to hostile agents, the possibility of varying environmental conditions could be explored. For example, environments where there are more complex obstacles, such as narrow corridors, or environments where the terrain is more difficult to traverse. This would require the flock to adapt to the environment and change their behaviour accordingly.

We may consider the introduction of temperature, precipitation, humidity, and wind speed to the model, which would affect the flock's behaviour. An implementation of this may focus on modifying the world generation with the introduction of biomes; Jiang et al. [37], Whittaker [38] note the common classification of biomes from precipitation and temperature, which could be extended to include other environmental factors.

To complement this, we may also simulate an agent's battery and signal strength when navigating the environment. As a result, the environmental conditions would affect the agent's signal strength and battery life, which should be considered when optimising the parameters of the model. Discussing more scenarios dependent on the

environment in this way may increase the robustness of the model. Additionally, simulating the communication between agents and the effect of the environment on this communication would bring the model closer to a realistic simulation, as described by Zhou et al. [7].

8 Evaluation

This section evaluates the project as a whole, discussing the successes and failures of the project and some key remarks.

8.1 Project Management

In the initial stages of planning, a Gantt chart was produced to outline the project timeline. This can be seen in Fig. 43.

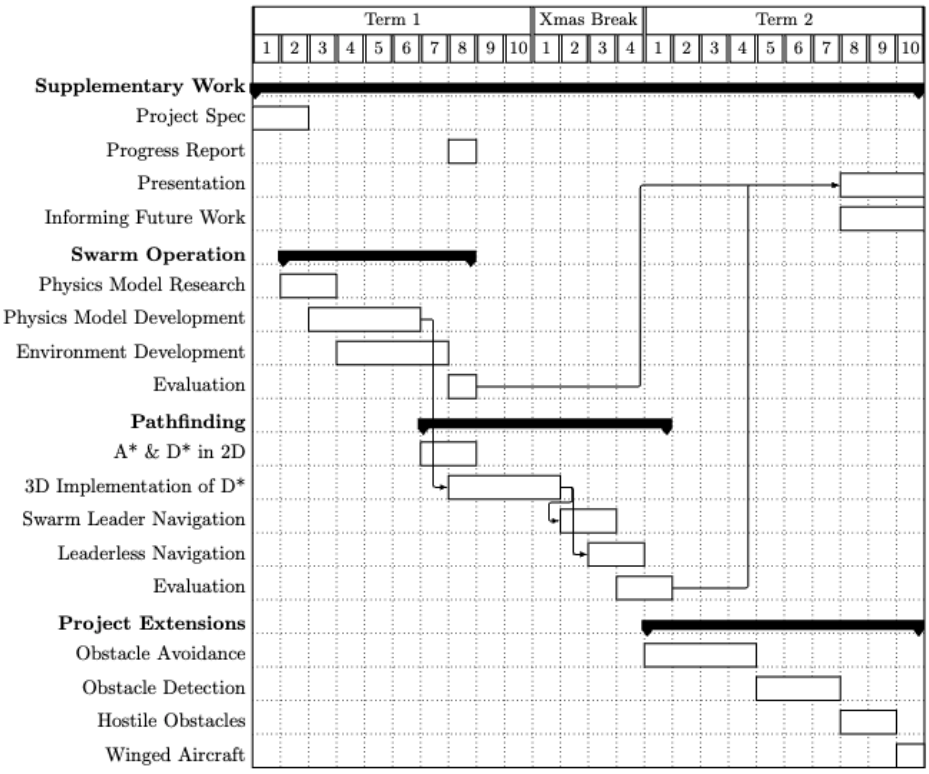


Figure 43: Gantt Chart

Breaking down the key sections of the project into smaller tasks allowed for a more manageable project. Looking at the Gantt chart, we note the modifications in the

project timeline below.

8.1.1 Project Timeline Modifications

The largest change to the timeline was the shift in focus over the Christmas break. Initially, the focus after finishing the operation of the Boids model and the agent control system was to focus on exploring and comparing pathfinding algorithms for the Boids model.

The reasoning behind this initial plan was to explore how different pathfinding algorithms, such as A*, could affect the behaviour of the agents in the Boids model. Additionally, the exploration of 'leadership' in the flock was planned. Instead, the focus shifted to optimising the emergent behaviour of the Boids model. This was due to the complexity of the pathfinding algorithms, particularly in three dimensions.

The Gantt chart was planned in a way that allowed for plenty of contingency time in the latter half of the project, in case of any delays or issues. Whilst this meant that some of the 'extensions' to the project were late, such as the exploration of obstacle detection and avoidance, whilst others were not completed, such as the exploration of hostile agents and environments, the project was completed on time. As such, this was a successful strategy.

The actual progress of the project can be seen in Fig. 44.

8.1.2 Methodology

The project was conducted in a way that allowed for a clear progression from the initial exploration of the Boids model to the optimisation of the model. As a single-person project, the development process followed a waterfall model, with each stage of the project being completed before moving on to the next stage. This was a successful strategy, as it allowed for a clear progression of the project. In a multi-person project, this method may not be suitable, as some team members may have a lot of idle time whilst waiting for others to complete their tasks.

The breakdown of the time spent on the project, broken down per week, can be seen in Table 16. Weeks 11-14 in both terms were the Christmas and Easter breaks, respectively. The table shows an average of ≈ 10 hours per week spent on the project, with a total time of 250 hours spent on the project. This does not include reading time, supervisor meetings, peer consultations or relevant lectures. The addition of these would likely bring the total time spent on the project to ≈ 300 hours.

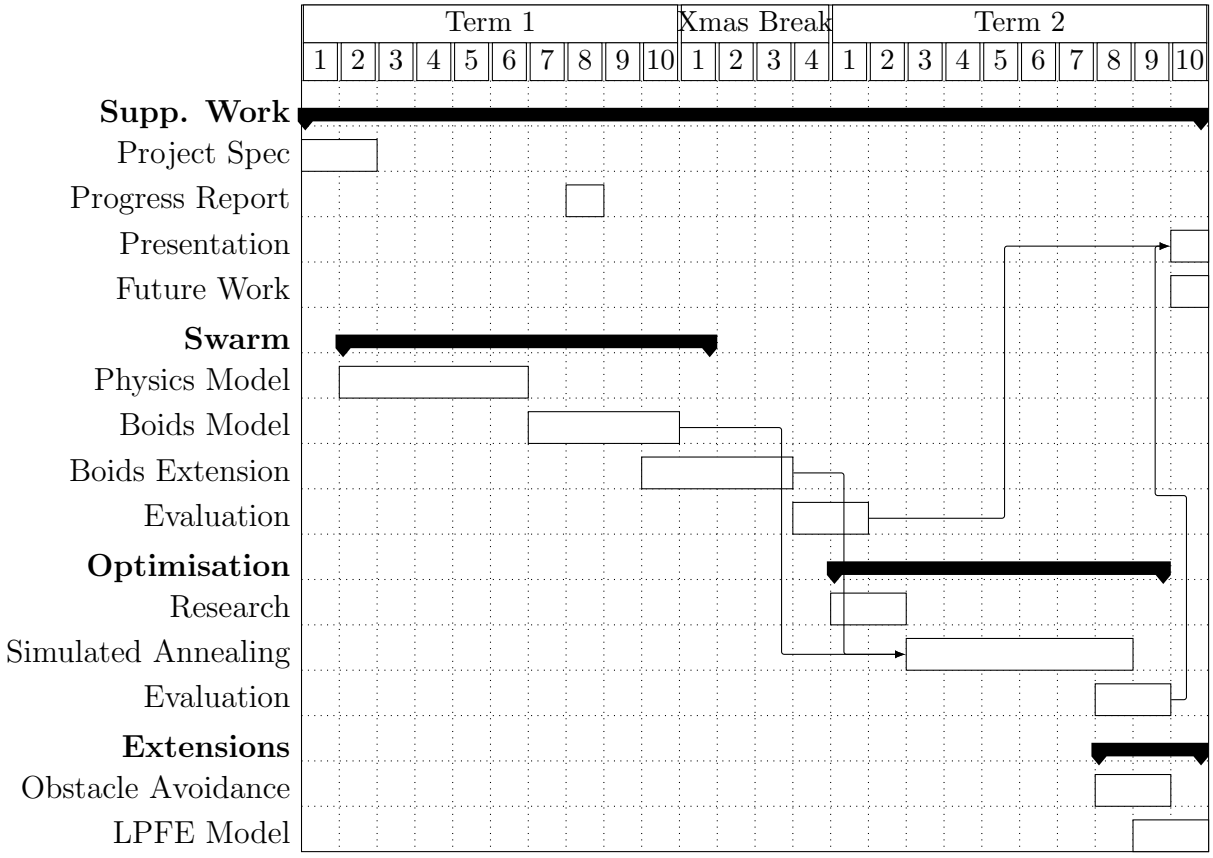


Figure 44: Gantt Chart of Actual Progress

8.1.3 Contingency Planning

As noted previously, the planned timeline allowed for plenty of contingency time in the latter half of the project. Complementary to this was the use of Github, which allowed for easy tracking of changes and the ability to revert to previous versions of the code. Github also allowed for the entire project to be stored remotely. This meant that the project was not lost in the event of a hardware failure, misplacement or theft; a real concern, as part of the project was completed abroad and required frequent travel between locations.

Keeping track of the tasks completed weekly meant this reflection was possible. This allowed for a clear understanding of the project progression and the ability to identify areas where more time could have been spent.

Week	Hours	Week	Hours
1	3	1	6
2	10	2	7
3	25	3	9
4	13	4	8
5	10	5	10
6	12	6	8
7	7	7	11
8	2	8	17
9	5	9	16
10	5	10	6
11	0	11	44
12	0	12	16
13	0	13	0
14	16	14	0

Table 16: Time Breakdown

8.2 Remarks: Assumptions

Though this report has focused on the optimisation of a parameter space to achieve emergent behaviour in a Boids model, several assumptions were left untweaked to simplify the model.

Much of the parameters of the flock generation were kept constant. Exploring how the flock size, avoidance radius and other parameters affect the emergent behaviour of the flock would be an interesting extension to this project. Likewise, the PID controller used to control the agents was kept constant. Exploring how the parameters of the PID controller affect the emergent behaviour of the flock could be explored.

Generally, there will always be some trade-off between some of these parameters we have kept constant. Reducing the number of agents in the simulation would likely reduce the number of collisions, but the effect of the cohesion behaviour, for example, would be more difficult to explore. Likewise, increasing the maximum tilt in the PID controller could have led to a faster convergence to the goal, but could have led to more collisions, oscillations and likely a more noisy simulation.

8.3 Author's Assessment of the Project

Overall, this project has been a success. The project has explored the optimisation of a Boids model using simulated annealing, a relevant and important topic in the field

of artificial intelligence and the wider field of computer science. The project has also introduced a novel extension to the Boids model, the Local Potential Field Emergence model.

The project has not explored the optimisation of the LPFE model, which would have been a natural extension of the project.

Hopefully, the project can be used as a basis for further exploration into the optimisation of emergent behaviour in agent-based models, specifically in a world-like environment.

Acknowledgments

This project's success would not have been possible without the support of my supervisor, Nathan Griffiths, who provided guidance and support throughout the project. Additionally, Dr. Claire Rocks provided a welcome second opinion on some initial ideas for the project's direction and Arpan Mukhopadhyay, who sparked valuable ideas from his questions and comments. I would also like to thank my peers, who provided valuable explanation, feedback, discourse and support throughout the project, namely Christopher Wilkinson, Ethan Larkin, Luke Sarfas and Maddie Sangway.

References

[1] 2024. [Online]. Available: https://libnoise.sourceforge.net/glossary/#coherentnoise

[2] K. H. Ang, G. Chong, and Y. Li, "Pid control system analysis, design, and technology," *IEEE transactions on control systems technology*, vol. 13, no. 4, pp. 559–576, 2005.

[3] D. Mandloi, R. Arya, and A. K. Verma, "Unmanned aerial vehicle path planning based on a* algorithm and its variants in 3d environment," *International Journal of Systems Assurance Engineering and Management*, vol. 12, no. 5, pp. 990–1000, Jul 2021. [Online]. Available: https://link.springer.com/article/10.1007/s13198-021-01186-9

[4] T. Chen, G. Zhang, X. Hu, and J. Xiao, "Unmanned aerial vehicle route planning method based on a star algorithm," in *2018 13th IEEE Conference on Industrial Electronics and Applications (ICIEA)*, 2018, pp. 1510–1514.

[5] K. Burwell, "Multi-agent pathfinding for unmanned aerial vehicles," *Upc.edu*, Oct 2019. [Online]. Available: https://upcommons.upc.edu/handle/2117/176279

[6] F. Mondada, G. Pettinaro, A. Guignard, I. Kwee, D. Floreano, J.-L. Deneubourg, S. Nolfi, L. M. Gambardella, and M. Dorigo, "Swarm-bot: A new distributed robotic concept: Swarm robotics (guest editors: Marco dorigo and erol Şahin)," *Autonomous Robots*, vol. 17, 01 2004.

[7] Y. Zhou, B. Rao, and W. Wang, "Uav swarm intelligence: Recent advances and future trends," *IEEE Access*, vol. 8, pp. 183 856–183 878, 2020.

[8] M. Verdoucq, G. Theraulaz, R. Escobedo, C. Sire, and G. Hattenberger, "Bio-inspired control for collective motion in swarms of drones," in *2022 International Conference on Unmanned Aircraft Systems (ICUAS)*, 2022, pp. 1626–1631.

[9] C. W. Reynolds, *Flocks, Herds, and Schools: A Distributed Behavioral Model*. New York, NY, USA: Association for Computing Machinery, 1998, pp. 273–282. [Online]. Available: https://doi.org/10.1145/280811.281008

[10] S. Hayat, E. Yanmaz, and R. Muzaffar, "Survey on unmanned aerial vehicle networks for civil applications: A communications viewpoint," *IEEE Communications Surveys & Tutorials*, vol. 18, pp. 2624–2661, 2016. [Online]. Available: https://api.semanticscholar.org/CorpusID:424353

[11] I. Perez, A. Goodloe, and W. Edmonson, "Fault-tolerant swarms," in *2019 IEEE International Conference on Space Mission Challenges for Information Technology (SMC-IT)*, 2019, pp. 47–54.

[12] J. Hu, H. Niu, J. Carrasco, B. Lennox, and F. Arvin, "Fault-tolerant cooperative navigation of networked uav swarms for forest fire monitoring," *Aerospace Science and Technology*, vol. 123, pp. 107 494–107 494, Apr 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1270963822001687

[13] L. M. Pyke and C. R. Stark, "Dynamic pathfinding for a swarm intelligence based uav control model using particle swarm optimisation," *Frontiers in Applied Mathematics and Statistics*, vol. 7, Nov 2021. [Online]. Available: https://www.frontiersin.org/articles/10.3389/fams.2021.744955/full#B18

[14] P. Basu, J. Redi, and V. Shurbanov, "Coordinated flocking of uavs for improved connectivity of mobile ground nodes," in *IEEE MILCOM 2004. Military Communications Conference, 2004.*, vol. 3, 2004, pp. 1628–1634 Vol. 3.

[15] E. Falomir, S. Chaumette, and G. Guerrini, "A mobility model based on improved artificial potential fields for swarms of uavs," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018, pp. 8499–8504.

[16] A. Juliani, V.-P. Berges, E. Teng, A. Cohen, J. Harper, C. Elion, C. Goy, Y. Gao, H. Henry, M. Mattar, and D. Lange, "Unity: A general platform for intelligent agents," 2020.

[17] A. G. Madey and G. R. Madey, "Design and evaluation of uav swarm command and control strategies," in *Proceedings of the Agent-Directed Simulation Symposium*, 2013, pp. 1–8.

[18] N. Watson, N. John, and W. Crowther, "Simulation of unmanned air vehicle flocking," in *Proceedings of Theory and Practice of Computer Graphics, 2003.*, 2003, pp. 130–137.

[19] S. A.-A. Alaliyat, H. Yndestad, and F. Sanfilippo, "Optimisation of boids swarm model based on genetic algorithm and particle swarm optimisation algorithm (comparative study)," in *European Conference on Modelling and Simulation*, 2014. [Online]. Available: https://api.semanticscholar.org/CorpusID:16528976

[20] G. Hattenberger, M. Bronz, and J.-P. Condomines, "Evaluation of drag coefficient for a quadrotor model," *International Journal of Micro Air Vehicles*, vol. 15, p. 4, 2023. [Online]. Available: https://doi.org/10.1177/17568293221148378

[21] M. Figliozzi, "Multicopter drone mass distribution impacts on viability, performance, and sustainability," *Transportation Research Part D: Transport and Environment*, vol. 121, p. 3, 2023.

[22] F. P. Thamm, N. Brieger, K. P. Neitzke, M. Meyer, R. Jansen, and M. Mönninghof, "Songbird - AN Innovative Uas Combining the Advantages of Fixed Wing and Multi Rotor Uas," *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. XL1, p. 345, Aug. 2015.

[23] S. Skogestad, "Probably the best simple pid tuning rules in the world," in *AIChE Annual Meeting, Reno, Nevada*, vol. 77. Citeseer, 2001, p. 276h.

[24] A. McCormack and K. Godfrey, "Rule-based autotuning based on frequency domain identification," *IEEE Transactions on Control Systems Technology*, vol. 6, no. 1, pp. 43–61, 1998.

[25] U. Technologies, "Unity - scripting api: Vector3.sqrmagnitude," 2022. [Online]. Available: https://docs.unity3d.com/ScriptReference/Vector3-sqrMagnitude.html

[26] H. Liu, X. Li, M. Fan, G. Wu, W. Pedrycz, and P. N. Suganthan, "An autonomous path planning method for unmanned aerial vehicle based on a tangent intersection and target guidance strategy," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 4, pp. 3061–3073, 2020.

[27] M. De Petrillo, J. Beard, Y. Gu, and J. N. Gross, "Search planning of a uav/ugv team with localization uncertainty in a subterranean environment," *IEEE Aerospace and Electronic Systems Magazine*, vol. 36, no. 6, pp. 6–16, 2021.

[28] K. Perlin, "An image synthesizer," in *Proceedings of the 12th Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH '85. New

York, NY, USA: Association for Computing Machinery, 1985, pp. 287–296. [Online]. Available: https://doi.org/10.1145/325334.325247

[29] T. R. Etherington, "Perlin noise as a hierarchical neutral landscape model," *Web Ecology*, vol. 22, no. 1, pp. 1–6, 2022. [Online]. Available: https://we.copernicus.org/articles/22/1/2022/

[30] T. Srinath, P. Can, and P. Mitch, "Minimum time trajectory generation for surveying using uavs," 2022.

[31] G. Deng, "Simulation-based optimization," 01 2007.

[32] Reynolds, 2001. [Online]. Available: https://www.red3d.com/cwr/boids/

[33] J. R. Caldwell, R. A. Watson, C. Thies, and J. D. Knowles, "Deep optimisation: Solving combinatorial optimisation problems using deep neural networks," 2018.

[34] F. Zhao, Y. Zeng, B. Han, H. Fang, and Z. Zhao, "Nature-inspired self-organizing collision avoidance for drone swarm based on reward-modulated spiking neural network," *Patterns*, vol. 3, no. 11, 2022.

[35] Y.-W. Chen, K. Kobayashi, H. Kawabayashi, and X. Huang, "Application of interactive genetic algorithms to boid model based artificial fish schools," in *Knowledge-Based Intelligent Information and Engineering Systems: 12th International Conference, KES 2008, Zagreb, Croatia, September 3-5, 2008, Proceedings, Part II 12*. Springer, 2008, pp. 141–148.

[36] C. Delgado-Mata, J. I. Martinez, S. Bee, R. Ruiz-Rodarte, and R. Aylett, "On the use of virtual animals with artificial fear in virtual environments," *New Generation Computing*, vol. 25, pp. 145–169, 2007.

[37] M. Jiang, B. S. Felzer, U. N. Nielsen, and B. E. Medlyn, "Biome-specific climatic space defined by temperature and precipitation predictability," *Global ecology and biogeography*, vol. 26, no. 11, pp. 1270–1282, 2017.

[38] R. H. Whittaker, "Communities and ecosystems." 1970.