Unmanned Aerial Vehicle Swarm Intelligence

An exploration of pathfinding and control for UAV swarms using swarm algorithms.

Antonio Brito

University of Warwick

March 13, 2024

Unmanned Aerial Vehicles

UAVs have a wide range of scientific applications:

- Area Reconnaissance & Surveying
- Mapping & Environmental Monitoring
- Agricultural
- Military
- Search & Rescue

UAVs are cheap & swarms introduce redundancy and the ability to work together.

UAV Swarms

UAVs have a wide range of scientific applications:

- Area Reconnaissance & Surveying: Increased coverage and accuracy
- Mapping & Environmental Monitoring: Faster
- Agricultural: Increased coverage and accuracy
- Military: Increased redundancy
- Search & Rescue: Increased redundancy

UAVs are cheap & swarms introduce redundancy and the ability to work together.
Can we utilise emergent, natural behaviour as part of autonomy?

Unity

We will use Unity, a game development platform, to model our UAV swarm and the surrounding environment.
Inspiration: *Boids* by Sebastian Lague

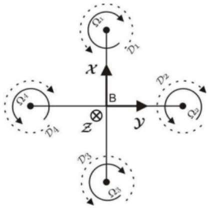

Figure: Quadcopter Model

[1]

Unity

We will use Unity, a game development platform, to model our UAV swarm and the surrounding environment.
Inspiration: *Boids* by Sebastian Lague

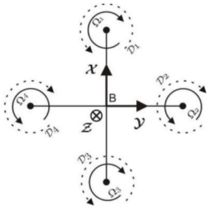

Figure: Quadcopter Model

[1]

Our first challenge is controlling the movement of the quadcopters.

PID Control

We will use a PID controller to control the movement of the quadcopters. We can define a goal orientation and the PID controller will adjust the motors to achieve this orientation.

Proportional	Integral	Derivative
$P = K_p e(t)$	$I = K_i \int_0^t e(t)dt$	$D = K_d \frac{d}{dt} e(t)$

Swarming Algorithm

1987: Craig Reynolds' Boids

Simulates the emergent behaviour present in flocks of birds.

- Square Avoidance Radius R_{av}^2
- Neighbourhood N
- Agent Position $\vec{a}$

Avoidance

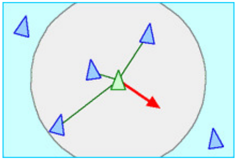

Figure: Avoidance Rule[3]

The avoidance rule ensures that agents do not collide with each other. This can be done by calculating the average position of the agents in the neighbourhood.

The avoidance move vector $\vec{A}$ can be represented as follows:

$$\vec{A} = \begin{cases} 0 & \text{if } |N| = 0 \\ \frac{1}{n_{avoid}} \sum_{n=1}^{|N|} \vec{a} - \vec{n}, \vec{a} - \vec{n} < R_{av}^2 & \text{otherwise} \end{cases}$$

with n_{avoid}, the number of neighbors for which the adjustment is non-zero.

Alignment

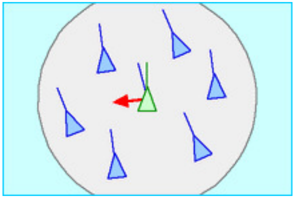

Figure: Alignment Rule[3]

The alignment rule ensures agents match the heading and velocity of agents in their neighbourhood.
Considering the agents $n \in N$ with heading $\vec{n}$,

$$\vec{B} = \frac{1}{|N|} \sum_{n=0}^{|N|} \vec{n}$$

Cohesion

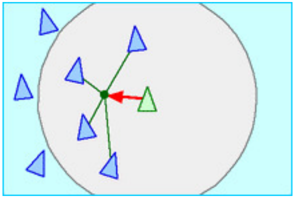

Figure: Cohesion Rule[3]

The cohesion rule ensures agents stick together.
The cohesion move vector $\vec{C}$ can be represented as follows:

$$\vec{C} = \begin{cases} 0 & \text{if } |N| = 0 \\ \frac{1}{n_{avoid}} \sum_{n=1}^{|N|} \vec{a} - \vec{n} & \text{otherwise} \end{cases}$$

Boids!

So we have our three fundamental behaviours.

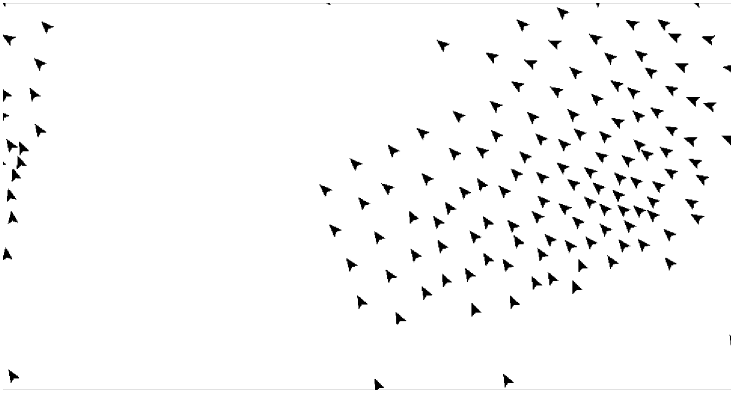

Figure: Emergent Behaviour[2]

Seeking

We now look to add the ability to seek a given goal to our agents. We will assume the location of the goal is known.

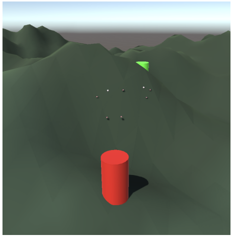
Figure: Start Platform

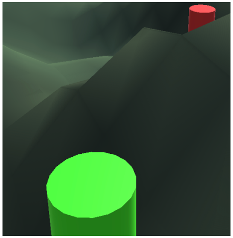
Figure: Goal Platform

This can be done by calculating the vector between the goal position and the current position of the agent.

$$\vec{D} = \vec{goal} - \vec{agent}$$

Terrain Generation

Terrain was generated using *Perlin Noise*, introduced by Ken Perlin in 1983.

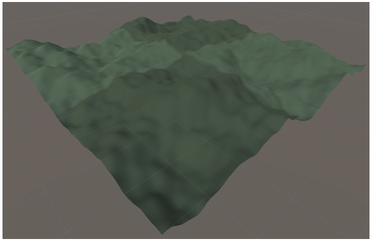

Figure: Perlin Noise Terrain

Terrain Avoidance

To implement the avoidance behaviour, we can simulate a *ranging* sensor ray on the agents.
We can then apply an upward thrust when the distance to the terrain is below a certain threshold.

$$vec(E) = \begin{cases} 0 & \text{if } d > R_{avoid} \\ (0, c_{thrust}, 0) & \text{otherwise} \end{cases}$$

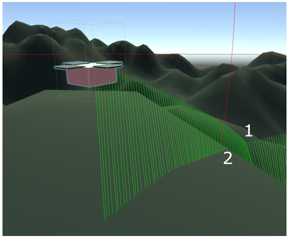

Figure: Agent with incident rays

Composite Behaviour

Optimisation

We can use a bounded linear combination of the rules to control the behaviour of our UAVs.

$$\vec{New} = \alpha\vec{A} + \beta\vec{B} + \gamma\vec{C} + \delta\vec{D} + \epsilon\vec{E}$$

Problem: How can we calculate the best values for $\alpha, \beta, \gamma, \delta, \epsilon$?

Simulated Annealing

We can define three ‘scenarios’ and a cost function to optimise the behaviour of the drones. We can then use a simulated annealing algorithm to find the best parameter values.
The scenarios chosen were:

- Shortest First-Arrival Time
- Reduced Spread (Cohesivity)
- Reduced Collisions

A modular cost function was defined to calculate the cost of the parameters in each scenario.

Optimisation: Shortest First-Arrival Time

We can see the effect of the parameters on the first-arrival time by the optimised parameter values:

Parameter	Value
Avoidance	8.5
Alignment	2.5
Cohesion	0.5
Seeking	17
Terrain Avoidance	16.5

Table: Optimised Parameters for Shortest First-Arrival Time, $c = 0.268$

Optimisation: Reduced Spread

We can consider the case where we want to reduce the spread of the agents' arrival time at the goal. We have found the optimised parameter values:

Parameter	Value
Avoidance	12.5
Alignment	4
Cohesion	15
Seeking	18.5
Terrain Avoidance	0.5

Table: Optimised Parameters for Reduced Spread, $c = 0.229$

Optimisation: Reduced Spread

We can also explore the trajectory of the simulations to see how the annealing schedule affects the paramters of the cost function.

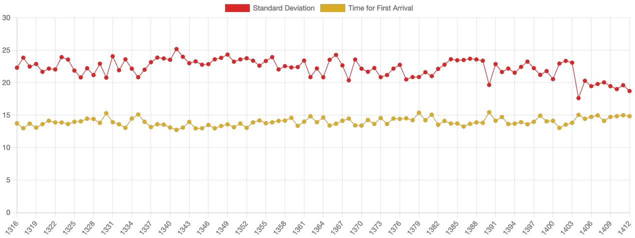

Figure: One SA Simulation Run

Issue 1: The chosen parameters may not have the intended effect on chosen observations.

Optimisation: Reduced Collisions

Looking at minimising collisions, we notice that the effect of the parameters on the cost function is not always as would be expected.

1. Collision numbers are not bounded.
2. Random spawn points mean a large range of initial conditions.

For example, for one of the lowest cost functions:

Parameter	Value
Avoidance	3
Alignment	15
Cohesion	15.5
Seeking	4.5
Terrain Avoidance	2.5

Table: Optimised Parameters for Reduced Collisions, $c = 0.202$

Issue 2: Simulated Annealing does not always find the global minimum.

Status Update

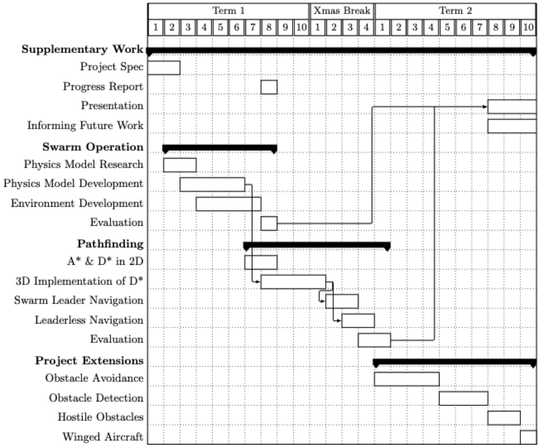

Figure: Gantt Chart

Future Work: Solving the Issues?

Issues

1. Chosen parameters may not have the intended effect on chosen observations. Curse of dimensionality?!
2. Simulated Annealing does not always find the global minimum.

Future Work: Solving the Issues?

Issues

1. Chosen parameters may not have the intended effect on chosen observations. Curse of dimensionality?!
2. Simulated Annealing does not always find the global minimum.

To solve this, we will aim to look at alternative optimisation algorithms and tweak our behaviours.

- Boids (1987) vs Boids (1999)?
- Particle Swarm Optimisation?

The focus has been on controlling specific behaviours, which contribute to a goal state. What if we control the goal state itself?

Future Work: Extensions

We have successfully implemented the pathfinding and control of a basic swarm of UAVs.
The hope is that we can introduce other kinds of obstacles into the simulation, much like terrain currently.

An interesting extension involves the introduction of hostile agents into the simulation, with the ability to avoid these agents or even ‘fight back’ in some cases.

Samir Bouabdallah and Roland Y. Siegwart.
Full control of a quadrotor.
2007 IEEE/RSJ International Conference on Intelligent Robots and Systems, pages 153–158, 2007.

RafaelKuebler.
Github - rafaelkuebler/flocking: 2d flocking in the unity3d game engine using craig reynolds' boids, 2018.

Craig W. Reynolds.
Boids (flocks, herds, and schools: a distributed behavioral model), 2024.