Biologically inspired UAV guidance: Using reinforcement learning to optimize flocking behaviour

Oscar Meunier

University of Glasgow, Glasgow, UK

Abstract

This Project uses reinforcement learning (RL) to develop flocking behaviour in drones to organise groups of UAVs using biologically inspired behavioural models. This explores how a multi-agent model is simulated and how the loss function can be constructed to achieve emergent behaviour focusing on how the reward function can be scaled with N agents. The model used is an actor-critic model, as this allows for continuous action and observation states. The 2D environment is constructed by layering potential field functions and boid flocking behavior. The agent controls drones using gains that amplify certain behaviours and is trained to maximise long-term cumulative reward. The reward function is constructed using variations on the exponential function to flock and arrive within a certain number of steps. As a result, flocking behaviour was achieved and can be scaled although computational resources meant that this was limited. Issues in the behaviour do arise however where some drones fail to arrive at the destination, prioritising flocking. This is most likely due to the design of the reward function as well as some limitations in the simulation. This dissertation demonstrates how reinforcement learning can be used with multi-vehicle UAV applications. By changing the (RL) structure to a more suitable multi-agent particle environment such as the ones available in the openAI database this could be applied at different scales whilst being much more conservative with computational requirements.

Keywords *Unmanned aerial vehicles (UAVs), Drones, Quadcopters, Flocking behavior, Boids algorithm, Potential field functions, Reinforcement learning (RL), Actor-critic model, Multi-agent systems, Emergent behavior, Reward function design, Simulation, Navigation, Guidance, Control, Machine learning, Neural networks, Policy gradients, Deep learning*

Acknowledgements

I would like to express my gratitude to my advisor, Dr Kevin Worrall for providing me with advice and steering me towards information that would prove essential to this dissertation. Dr Worrall has also advised me further and is one of the reasons I have chosen to pursue a MSc of robotics and artificial intelligence at the university of Glasgow.
Dr Dezong Zhao's, my professor for autonomous guided systems has also been an enormous help in creating the model that was used in this dissertation. I would like to thank him for his clear and well-formulated lectures from which I have derived a lot of the code for the simulation.
I would also like to thank Daniel Shiffman, the author of the nature of code, who's tutorials have been an enormous help in learning how to code multiple agents efficiently.[34]
In addition, I would like to thank Dr Lex Fridman, a professor of the university of MIT, who has provided his lectures on deep neural networks for free. [46]

Contents

1 Introduction

In recent decades UAV technology is becoming increasingly more accessible as today's manufacturing enables cheaper, smaller, and more powerful electric motors. Companies such as Drone up[8], Wing[9] and DHL[10] are developing autonomous drones to enable companies to deliver packages. Other projects such as Europe's *SKYOPENER* [11] aim to expand civilian applications of Remotely Piloted Aircraft Systems (RPAS).
There is some discussion over the efficiency of drones in comparison to electric vehicles. A 2019 study [12] found that switching entirely to drones would not be desirable in terms of total efficiency but could be useful in a last-mile approach for light and short distance deliveries. *RAND Corporation*, a research organization,[13] estimates that if 20% of deliveries were replaced by drones this could decrease diesel consumption of the United Parcel Service (UPS) by 5.7%, reducing the carbon footprint due to trucking as they are converted to electric substitutes.

Drone flight is already somewhat controlled via legislation, figure 1 shows restricted zones in the UK[14]. One can see how this would become a complex guidance control problem especially if a drone needs to interact with other UAVs, with which it does not have direct communication. A city council or other similar entity might want more localised control over which airspace is used and when this is the case. As the drone industry expands and evolves so does the control problem. By using reinforcement learning drones might be able to adapt.

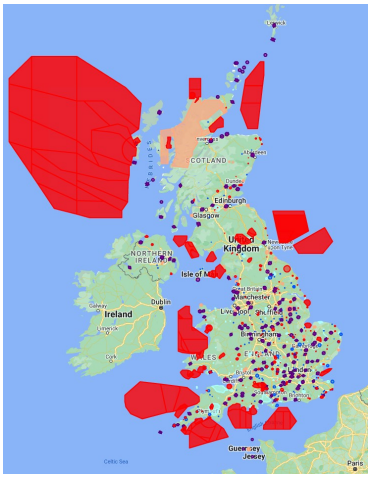


Figure 1: UK restricted airspace

This essay looks at a hypothetical future where drones are used on a much larger scale. Being as easily manoeuvrable as they are and some drones having a range of up to 100km [15] (although high-end consumer drones manage about 10-15km [16]) the risk of malicious use of drones is a threat that should not be taken lightly.[17] If drones are used more frequently, we may be dependent on the network they provide. Cyber-attacks could be a way of causing havoc and affecting supply chains. For this reason, measures should be prepared to mitigate these risks to the maximum extent. Anti-drone technology is available[18] and companies such as D-fend [19] are already working towards the solution although a full consensus is yet to be arrived at as different systems each has risks and limitations[20].

While fixed-wing UAVs benefit from flocking in the same way as migratory birds do quadcopter drones do not. Therefore, this application focuses on how to control drone traffic of general UAVs. As it allows for a general agent to be created which can adapt in real-time. If a general agent can be created this could reduce the problem of scaling as all computing could be achieved by individuals. In this investigation, a reinforcement learning agent will be trained to optimise drone traffic by layering boid flocking behaviour and potential functions. This will mean that multiple agents will come to a general consensus on a trajectory through sensing the aggregate of proximal agent headings and positions.

2 Guidance methods

In this simulation, drones will be controlled using a combination of boid flocking behaviour[2] and artificial potential field functions[30].

2.1 Potential field functions

In a similar fashion to conservative forces such as gravity and electrostatic force, the guiding force acting on each drone can be represented as a potential field function U. An important property of conservative forces is:

$$\vec{F} = -\Delta U = -\begin{bmatrix} \frac{\partial U}{\partial x} & \frac{\partial U}{\partial y} \end{bmatrix}^T \tag{1}$$

We can define the attractive potential U_a of a point as:

$$U_a = \frac{1}{2} K_a \rho_a^2 \tag{2}$$

Where, $\rho_a = \sqrt{(x-x_a)^2 + (y-y_a)^2}$ is the distance to that point and K_a is the attractive constant. This is depicted in figure 2 where x_a and y_a are the coordinates of the destination and x and y are coordinates of the drone. This will be used as the guiding force towards individual destinations. Using equations 1 and 2 the attractive force can be found as:

$$\vec{F}_a = \nabla U_a = \nabla \left(\frac{1}{2} K_a \left[(x-x_a)^2 + (y-y_a)^2 \right] \right)$$

so,

$$\begin{aligned} F_{ax} &= K_a(x_a - x) \\ F_{ay} &= K_a(y_a - y) \end{aligned} \tag{3}$$

which can be represented as:

$$\vec{F}_a = K_a |\vec{p}_a - \vec{p}_d| \tag{4}$$

Where $\vec{p}_a$ and $\vec{p}_d$ are the attractive and drone position vectors respectively. For later simplification $|\vec{p}_a - \vec{p}_d| = \vec{V}_E$. Where $\vec{V}_E$ is the desired velocity vector in the direction of the potential field.

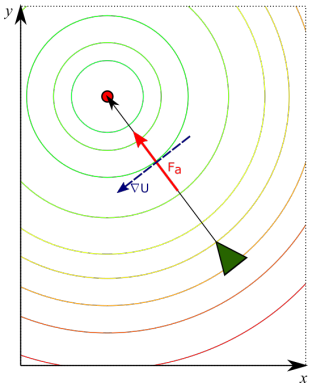


Figure 2: Potential field of destination of one drone

From figure 2 it can be seen that the attractive potential is such that the force always acts towards the destination.

2.2 Steering Force

Steering force is what gives autonomous agents the ability to steer via their own coordinate system. This simple guidance law established by Craig Reynolds [5] is the difference between the actual velocity $\vec{v_a}$ of the drone and the desired velocity heading $\vec{v_d}$ determined by the behavioural model. So the steering force is:

$$\vec{F} = K(\vec{v_a} - \vec{v_d}) \tag{5}$$

Where K is a gain which determines the weight of a specific behaviour.

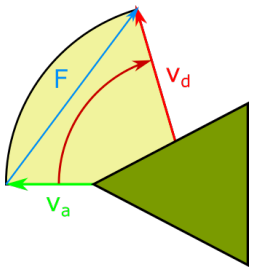


Figure 3: Steering force

2.3 Flocking behaviour

The flocking behaviour that will be introduced is from Craig Reynold's work on boids [2] and model formulae are established from derivative work [4]. This uses boid steering behaviour to mimic the aggregate motion of group herding/flocking dynamics. This motion that we recognise as flocking depends upon a localised perception of the world as in nature where, in large flocks or herds, individuals are only aware of proximal members. The local flock is determined by the perception radius, r_p, of each individual. Flockmates will be used to refer to drones within the perception radius of each subject drone as demonstrated in figure 4.

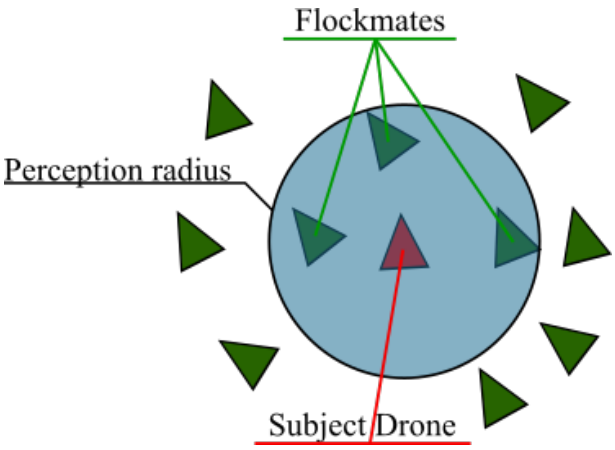


Figure 4: Drone flocking

The direction and speed of every subject drone A at position $\vec{p}_A$ depends on the direction and speed of all flock-mates within its perception radius r_p. So for all other drones in with a position $\vec{p_X}$ if:

$$\|\vec{p}_X - \vec{p}_A\| \leq r_p \tag{6}$$

Then the drone is a flock-mate and therefore is a member of the set $\mathscr{F}_i \subset \{1,2,...N\}/\{i\}$. Otherwise, it is ignored. $\mathscr{F}_i$ then represents the set of all drones that agent i is aware of.

The localised flocking will be comprised of 3 steering behaviours that describe how the drone moves based on the velocities and positions of flock-mates. Note that ‖ denotes a scalar vector magnitude and | is a directional vector. The behaviours are as follows:

2.3.1 Separation

The entity must keep its distance from other drones to avoid collisions. To achieve this the drone manoeuvres away from other drones with a force inversely proportional to the distance between them:

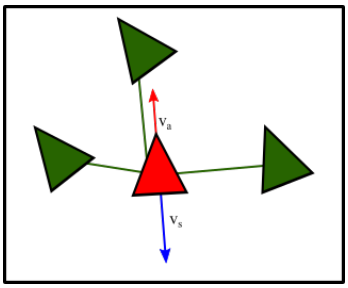


Figure 5: Separation force

$$\vec{v}_{s_i} = \sum_{j=\mathscr{F}_i} \frac{|\vec{p}_j - \vec{p}_i|}{\|\vec{p}_j - \vec{p}_i\|}$$
$$so,\ \ \vec{F}_{Si} = K_s \vec{V}_{DS} = K_s(\vec{v}_{a_i} - \vec{v}_{s_i}) \tag{7}$$

2.3.2 Alignment

By taking the average of all velocity vectors of its flock-mates, each drone steers into alignment with its flock-mates. The flock comes to a consensus of heading through velocity matching.

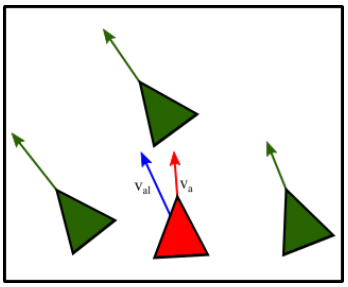


Figure 6: Alignment force

$$\vec{v}_{al_i} = \sum_{j=\mathscr{F}_i} \vec{v}_j$$
$$so,\ \ \vec{F_{Al_i}} = K_{al} \vec{V}_{DA} = K_{al}(\vec{v}_{a_i} - \vec{v}_{al_i}) \tag{8}$$

2.3.3 Cohesion

This is what keeps the flock grouped together. Members of the flock move towards the average position of the flock. This includes the subject drone position.

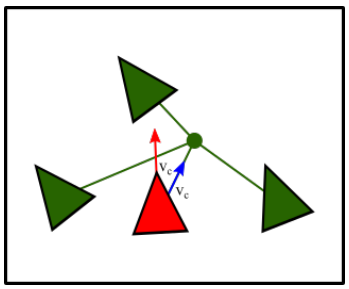


Figure 7: Cohesion force

$$\vec{v}_{c_i} = \vec{p}_i + \sum_{j=\mathscr{F}_i} \vec{p}_j$$
$$so,\ \ \vec{F}_{C_i} = K_c \vec{V}_{DC} = K_c(\vec{v}_{a_i} - \vec{v}_{c_i}) \tag{9}$$

K_s, K_{al} and K_c are the separation gain alignment gain and cohesion gain respectively. They drive the flocking behaviour of the agent by increasing the magnitude of velocity vectors $\vec{V}_{DS}$, $al\vec{V}_{DA}$ and $\vec{V}_{DC}$.

3 Machine learning problem

Machine learning is a subset of artificial intelligence, and deep learning is a subset of Machine learning. Deep learning classifies machine learning that uses deep neural networks. The neural network used in this scenario is an actor-critic, reinforcement learning (RL) approach that reaches an optimal solution through the simulation of multiple episodes.[28]

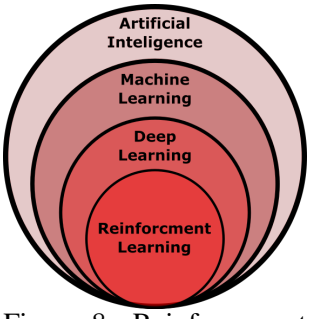


Figure 8: Reinforcement learning subset

3.1 Neural networks and deep learning

Artificial neural networks (ANN) are comprised of the input layer, hidden layers and the output layer. There can be many hidden layers constructed using perceptrons, the building block of an ANN, that map the inputs to the outputs. ANNs can have multiple inputs and outputs behaving as a MIMO controller and can map highly nonlinear input functions to any output signal. This is why they are commonly termed universal approximators.

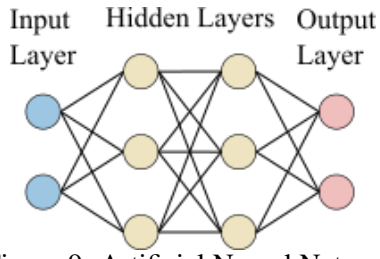


Figure 9: Artificial Neural Network

3.1.1 Perceptron

The perceptron, shown in figure 10 is the activation function of a weighted sum.

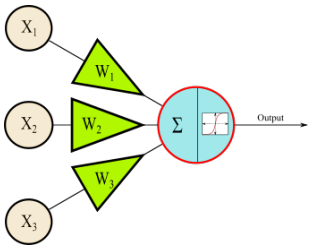


Figure 10: Perceptron

So with inputs and weight are:

$$\boldsymbol{x} = [x_1, x_2, ..., x_k], \boldsymbol{w} = [w_1, w_2, ..., w_k]$$

then,

$$\boldsymbol{x} \cdot \boldsymbol{w} + b = 0$$

The hidden layers made up of perceptrons change their weights and biases through back-propagation in the direction suggested by the loss function covered later on. In this case, the actor-critic network is used.

3.1.2 Activation function

An activation function is used since this allows the agent to map complex relationships of output and input. It can vary but, in this case, the sigmoid function is used as it can map continuous functions from 0 to 1.

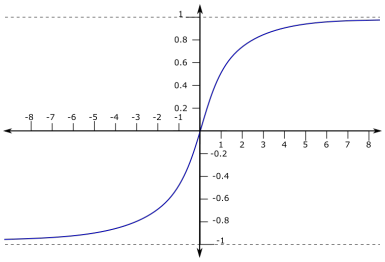


Figure 11: Sigmoid activation Function

$$\sigma(z) = \frac{1}{1+e^{-z}} \tag{10}$$

As the input goes from $-\infty$ to $+\infty$ the output ranges from 0 to 1. This means large differences in the input are normalised which allows the network to learn highly non-linear functions.

3.2 Reinforcement Learning

Reinforcement learning (RL) [27] is used to optimise complex or unpredictable control. The agent interacts with an environment to maximise reward. At each discrete time step t, with a given state $s \in S$ the agent chooses an action $a_t \in A$ with respect to the policy function $\pi : S \rightarrow A$. The agent then receives a reward r_{t+1} and a new state s_{t+1} for that action[37]. This model is known as a Markov Decision Process(MPD)[36], which means the agent acts in a discrete, stochastic, sequential environment.

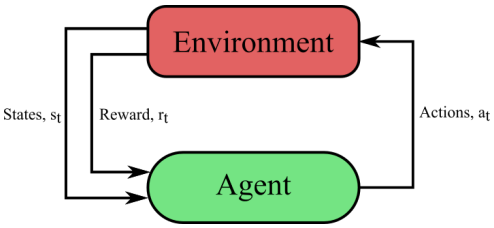


Figure 12: Reinforcement learning Agent

3.2.1 Policy

Policy $\pi_\theta(s_t, a_t)$ can be either Stochastic or deterministic:

- **Deterministic** - A policy is deterministic if there is a clear action for any state. This will not explore new options and is referred to as greedy. This would be a pre-trained agent which has a consistent behaviour.
- **Stochastic** - This policy means that there is some statistical element of variation is included. $\pi_\theta(s_t, a_t)$ represents the conditional probability density at a_t associated with the policy.[38]

In this case, the policy $\pi_\theta(s_t, a_t)$ behaves stochastically with the MPD environment such that:

$$s_0, a_0, r_1, s_1, a_1, r_2, ..., s_{k-1}, a_{k-1}, r_n, s_k$$

The policy $\pi_\theta(s_t, a_t)$ is a neural network with policy parameters θ which are initialised at arbitrary values. The policy parameter are then changed using the policy gradient theorem covered later[28] to maximise long cumulative reward.

3.2.2 Reward

The return at time step t is is defined as the discounted sum of rewards:

$$r_t^\gamma = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + ... + \gamma^{k-t} r_k$$

Return is often expressed as G_t and can be represented as:

$$G_t = r_t^\gamma = \sum_{k=t}^{T} \gamma^{k-t} r(s_k, a_k) \tag{11}$$

Where discount factor $0 < \gamma < 1$ determines the priority of the short term rewards. This pushes the agent to finish the simulation faster as it decreases since the short term reward is less valuable than long term reward.

3.2.3 Value function and bellman optimality

A value function is defined as the estimated total discounted reward. In this case, a Q-actor-critic network is used which means the **state-action** value function [38] is used:

$$Q^{\pi}(s_t, a_t) = \mathbb{E}[G_t | s_t = s, a_t = a; \pi]$$

This equation provides a function that estimates the future reward with a given initial state following the current policy. Since this is a maximisation problem the optimal state-value [39] the optimal state-action value function indicates the maximum reward:

$$Q_*(s,a) = \max_{\pi} Q_{\pi}(s,a)$$

Bellman Optimality Equation

Bellman proved that optimal action value function Q_* satisfies:

$$Q_*(s,a) = \mathbb{E}[r_{t+1} + \gamma \max_{a'} Q_*(s',a')] \tag{12}$$

This means that at time t, for any state-action pair (s,a), the expected return from a given state s and action a, using the optimal policy function will be the sum of the expected reward r_{t+1} achieved through action a and the maximum expected discounted reward, achievable for any (s',a'). (s',a') are the expected next state-action pair. s' will be the state from which the best action a' can be taken at time $t+1$.[40]

Loss function

To converge to an optimal solution the agent then acts to find the actual $Q(s,a)$. The loss is defined as the difference between the optimal Q-value and the optimal Q-value so the loss is $Q_*(s,a) - Q(s,a)$. They are compared iteratively until the value function converges to the optimal solution:

$$Q'(s,a) = (1-\alpha) \underbrace{Q(s,a)}_{\text{Old value}} + \alpha \overbrace{[r_{t+1} + \gamma \max_{a'} Q_*(s',a')]}^{\text{Learned value}} \tag{13}$$

Where α is the learning rate which determines how much of the information the previously computed Q-value will remain in the next iteration.

3.2.4 Policy Gradient theorem

The reinforcement learning objective is to maximise long term cumulative reward. this can be formulated as:

$$J(\theta) = \mathbb{E}[\sum_{t=0}^{T-1} r_{t+1}] \tag{14}$$

In every iteration, the policy parameter is changed through back-propagation in the direction of gradient ascent as this is a maximisation problem:

$$\theta \leftarrow \theta + \frac{\partial}{\partial \theta} J(\theta)$$

Derived in [23], the policy gradient is then.

$$\nabla_\theta J(\theta) = \mathbb{E}[\sum_{t=0}^{T-1} \nabla_\theta log \pi_\theta(a_t|s_t) Q(s,a)] \tag{15}$$

The reinforce algorithm is a Monte-Carlo[35] variant of policy gradients. This is a statistical method that chooses random samples so that agents can explore different policies.

3.2.5 Actor-Critic Methods

The overall aim of this optimization function is to reduce the loss function through back-propagation and gradient ascent[37].The Q-actor-critic achieves this with 2 parametrised neural networks:

- **Value critic** - estimates the state-action value from Bellman equation 12.
- **Policy actor** - This updates the policy in the direction suggested by the critic through gradient ascent in equation 15.

4 Method

The RL training is performed using the MATLAB reinforcement learning toolbox [44]. The agent runs a simulation until it ends due to predetermined conditions, which, in this case is if the maximum number of steps is reached or all drones arrive at final locations, this is classified as an episode. The agent calculates the cumulative reward over one episode. Agents with the highest cumulative reward are saved.

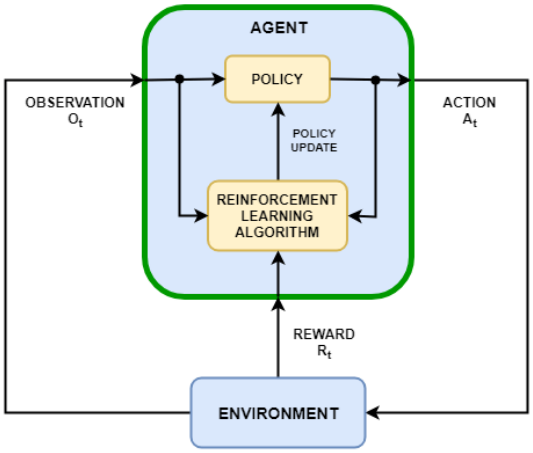


Figure 13: Rl Environment-Agent Interaction [29]

1. Construct a reset and step function that will be used to construct the environment in which the agent will be in.

2. Define action and Observation information to define how the agent and environment will interact.

3. The environment was achieved using *rlFunctionenv()* which requires action and observation information as well as the step and reset function.

$$\text{Environment} = \text{rlFunctionenv}(\text{ActInf},\text{ObsInf},\text{Stepfunc},\text{Resetfunc})$$

4. The agent was constructed using the *rlACAgent()* with the same action and observation options.

$$\text{Agent} = \text{rlACAgent}(\text{ActInf},\text{ObsInf})$$

5. Training options are set Using *rlTainingOptions()* which are used to evaluate the success of an agent, then Agent is trained using *train()*

$$\text{TrainingData} = \text{train}(\text{Agent},\text{Environment})$$

6. Agents with the highest cumulative reward are kept.

4.1 Reset function

The reset function initiates each episode. It sets the initial positions, velocity, acceleration and endpoints for n drones drone. Figure 14 shows the environment 3 drone objects are created at random initial coordinates and 3 corresponding endpoints.

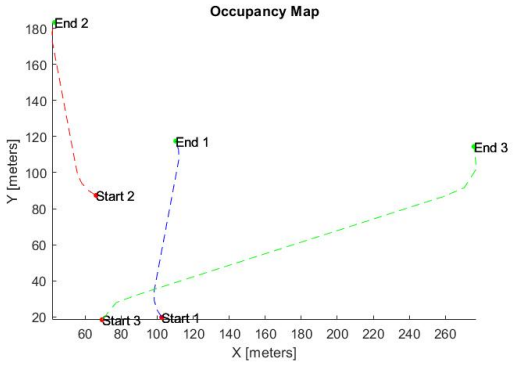


Figure 14: Environment initial conditions

4.2 Step Function

4.2.1 Actions

From equations, 3, 7, 8 and 9 in section 2. The control input is the input force to the system so:

$$u_i = K_{a_i}\vec{V}_{E_i} + K_{s_i}\vec{V}_{SC_i} + K_{al_i}\vec{V}_{AlC_i} + K_{c_i}\vec{V}_{CD_i} - K_{d_i}\vec{V}_i$$

where:

$$\mathbf{U}(t) = [u_1(t), u_2(t), ..., u_N(t)]$$

Control signal $U(t)$ is constructed with gains such that:

$$K_i(t) = \begin{bmatrix} K_{a_i} & K_{s_i} & K_{al_i} & K_{c_i} & K_{d_i} \end{bmatrix} V_i(t) = \begin{bmatrix} \vec{V}_{E_i} \\ \vec{V}_{SC_i} \\ \vec{V}_{AlC_i} \\ \vec{V}_{CD_i} \\ \vec{V}_{d_i} \end{bmatrix}$$

where:

$$\begin{aligned} \mathbf{A}(t) &= [K_1(t), K_2(t), ..., K_N(t)] \\ \mathbf{V}(t) &= [V_1(t), V_2(t), ..., V_N(t)] \end{aligned} \tag{16}$$

The agent is able to change gains to vary the behaviour of the system, prioritising certain behaviours. $\mathbf{A}(t)$ is the action at each step. The information is constructed using rlNumericSpec() which is used to set the action as a $5 \times N$ array of continuous values such that:

```
ActionInfo = rlNumericSpec([5 n]);
ActionInfo.LowerLimit = zeros(5,n);
ActionInfo.UpperLimit = ones(5,n)*20;
```

Reasonable Upper and lower limits were set based on other boid simulations parameters.

4.2.2 Observations

As actor-critic networks are a model free approach more than just states spaces can be observed. The observations made are the 5 desired velocities calculated from the boids model as well as position, endpoint and number of flock-mates. This is constructed as follows:

$$O_i = \{V_i^T, \vec{p}_i, \vec{p}_{end}, j \in \mathscr{F}_i\}$$

where:

$$V_i^T = \left[\vec{V}_{E_i}, \vec{V}_{SC_i}, \vec{V}_{AlC_i}, \vec{V}_{CD_i}, \vec{V}_{y_i}\right]$$

These are all 2 dimensional vectors except for $j \in \mathscr{F}_i$ which is the number of flock-mates for agent i. The x and y coordinates are separated to give:

$$O_i = \left[V_{E_{ix}}, V_{SC_{ix}}, V_{AlC_{ix}}, V_{CD_{ix}}, V_{d_{ix}}, p_{ix}, p_{end_{ix}}, V_{E_{iy}}, V_{SC_{iy}}, V_{AlC_{iy}}, V_{CD_{iy}}, V_{d_{iy}}, p_{iy}, p_{end_{iy}}, j \in \mathscr{F}_i\right]$$

The full system observation at time step t is constructed as:

$$\mathbf{O}(t) = [O_1(t), O_2(t), ...O_N(t)]$$

This dimension is defined in MATLAB as:

ObservationInfo = rlNumericSpec([15 n])

4.2.3 States: multi-agent double integrator

A Multi-agent double integrator model[42] was used to simulate environment dynamics. Let N drones be moving in a 2-Dimensional plane. The linear state-space representation of the double integrator model is then:

$$\begin{aligned} \dot{\xi}_i(t) &= A\xi_i(t) + Bu_i(t) \\ v_i(t) &= C\xi_i(t) \end{aligned} \tag{17}$$

Where $\xi_i = [x_i, \dot{x}_i, y_i, \dot{y}_i]^T$ and:

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \quad B = \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix}$$

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Here, v_i is the measured output which in this case is the position (x_i, y_i) of the i^{th} drone. This is constructed using the perception radius from equation 6 section 2.3. In this multi agent system agents are aware of relative speed:

$$z_i(t) = \sum_{j \in \mathscr{F}_i} (v_i(t) - v_j(t)) \tag{18}$$

For $i = 1...N$, the flock consists of the set $\mathscr{F}_i \subset \{1,2,...N\}/\{i\}$.This then denotes the other drones that the agent i is aware off. The system is:

$$\dot{X}(t) = (I_N \otimes A)X(t) + (I_N \otimes B)U(t) \tag{19}$$

where,

$$\begin{aligned} X(t) &= (\xi_1(t), \xi_2(t), ...\xi_N(t)) \\ U(t) &= (u_1(t), u_2(t), ...u_N(t)) \end{aligned} \tag{20}$$

at the network level:

$$Z(t) = (\mathscr{F} \otimes C)X(t) \tag{21}$$

4.3 Reward function

The reinforcement learning objective from equation 14 section 3.2.4:

$$J(\theta) = \mathbb{E}[\sum_{t=0}^{T-1} r_{t+1}]$$

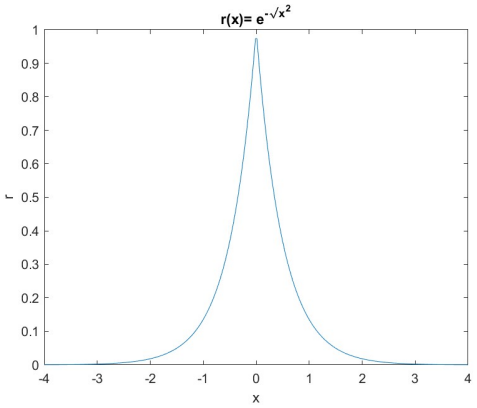
The reward therefore needs to maximise in the direction of the desired emergent behaviour.[43].

- **Step reward** provided at every step.
- **Episodic reward** is provided at the end of every simulation which occurs at once all agents have arrived or the maximum iterations have been reached.

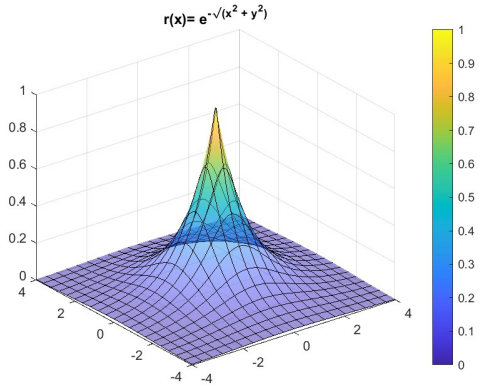
In this scenario it is the agent should move towards the destination but also flock with other agents as long as this does not sway it too much from its original objective. If the agents are only rewarded for discrete observations such as arriving at a destination or flocking with a member the reward system may be sparse, where the agent receives no signal for long series of actions and observations. To reduce the sparsity of reward, a continuous reward function is designed to push certain behaviours which is incorporated into the step reward function. [7]

4.4 Shaping step reward

The function $e^{-k_r\sqrt{x^2}}$ is used to construct the reward function.



(a) Exponential decay 2D



(b) Exponential decay 3D function

As shown in figures 15a and 15b, The maximum reward is given at the roots of the function, which in this case are the coordinates of the agent, and tends to 0. This is ideal for a multi-agent system as it allows the control of the size of the horizon of influence of the agents with the constant k_r which is found by applying constraints to the radius of effect and threshold.

Let the reward function for a point be:

$$r(x,y) = e^{-k_r\rho}$$

Where $\rho = \sqrt{(x-x_{pos})^2 + (y-y_{pos})^2}$. When agent coordinates are equal to the point coordinates:

$$e^0 = 1$$

The maximum area of effect is defined as ρ_{max} and the threshold reward r_t is the reward at $\rho = \rho_{max}$ so:

$$r_t = e^{-k_r\rho_{max}}$$

so:

$$k_r = \frac{ln(r_t)}{\rho_{max}}$$

The reward function of the drones is set to a smaller radius as it is only beneficial to flock when close to another agent. The maximum area of influence is expressed as relative to the size of the map by using the map hypotenuse $h = \sqrt{width^2 + height^2}$. The threshold was set to $r_t = 0.05$ as this meant the signal was only 5% of the original signal at that point and so had a low effect on change in reward.

Flocking reward

The flocking reward was chosen so that the effective area of the drones needed to be smaller than the endpoint effective area so:

$$max(\rho_f) = 0.2h$$

then,

$$k_f = \frac{ln(0.05)}{0.2h}$$

The flocking reward is then the average of all flocking rewards thus,

$$r_{f_i} = \frac{1}{n}\sum_{j=1}^{f} e^{-k_f\sqrt{(x_i-x_j)^2+(y_i-y_j)^2}} \tag{22}$$

Endpoint Reward

$$max(\rho_e) = 0.6h$$

so,

$$k_e = \frac{ln(0.05)}{0.6h}$$

$$r_{e_i} = e^{-k_e\sqrt{(x_i-x_e)^2+(y_i-y_e)^2}} \tag{23}$$

The sum of these two reward functions is represented in figure 16. See appendix A1 for more details.

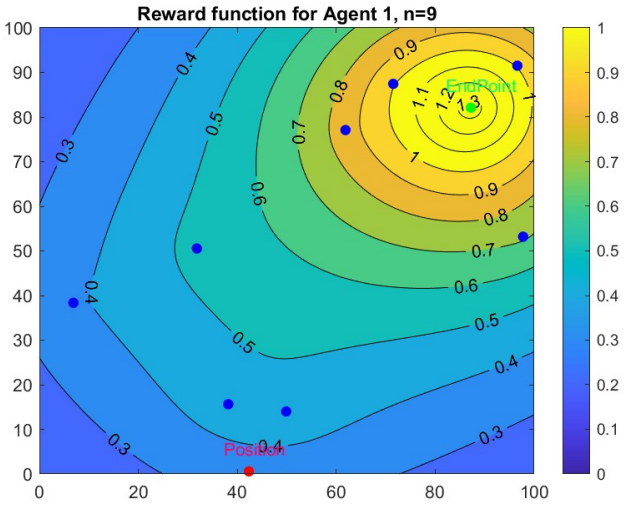


Figure 16: Reward function with 9 agents

Penalty

From equation 11 cumulative return for time step k is:

$$r_t^{\gamma} = \sum_{k=t}^{T} \gamma^{n-t} r(s_k, a_k)$$

If $0 < \gamma < 1$ and the $r > 0$ then it can be seen that r^{γ} tends to 0. This implies the maximisation problem is impossible. To solve this, a penalty is added to the step function to push the agent to arrive within a set number of steps defined as T_{max}. The exponential function $ae^{-k_p x}$ is used as constants a and k_p can be changed to form the appropriate signal. For the agent to arrive within T_{max} steps the following definition is formulated:

$$r_t = 0 \text{ for } t = T_{max}$$

Total step reward r_t is the sum of equations 22 and 23 minus the penalty.

$$r_{t_i} = r_{f_i} + r_{e_i} - r_{p_i} = 0$$

where:

$$r_{p_i} = ae^{k_p T_{max}}$$

and maximum possible values for r_{f_i} and r_{e_i} is 1. Through substitution this yields:

$$ae^{k_p T_{max}} = 2$$

Applying initial conditions a is found:

$$ae^0 = a \text{ where a is set so that } a << 2,$$

Applying terminal conditions:

$$k_p = \frac{ln(\frac{2}{a})}{T_{max}} \tag{24}$$

Note that r_{f_i} the sum of r_{e_i} is only equal to 2 when all reward functions are at the same point which is unlikely. It is better to overestimate, however, as this prevents exponential growth of the reward.

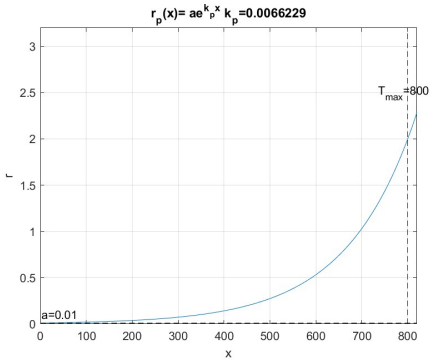


Figure 17: Penalty function

Figure 17 shows the penalty function used. The reward for each agent step is then:

$$r_{t_i} = r_{f_i} + r_{e_i} - r_{p_i}$$

The total reward for each simulations time step n is then the average reward of the N agents:

$$r_{step_n} = \frac{1}{N} \sum_{i=1}^{N} r_{t_i}$$

4.5 Episodic reward

The episodic reward takes into account the overall success of the simulation, providing this reward once a maximum number of steps is achieved or all agents have arrived at their destinations. This is represented as:

$$R_{total} = r_{episodic} + \sum_{n=1}^{T} r_{step_n}$$

The episodic reward is based of the terminal state of the system. It takes number of individuals that have arrived at their destinations such that if agent i has arrived within a set arrival distance:

$$r_{a_i} = 1$$

And if agent i hasn't arrived:

$$r_{a_i} = 0$$

Then total episodic reward is

$$r_{episodic} = \frac{1}{N} \sum_{i=1}^{N} r_{a_i}$$

The overall maximisation problem becomes:

$$MaxJ(\theta) = max(R_{total})$$

4.6 Training Options

Knowing the maximum cumulative reward that is achievable is important as it sets limits as to what outcome to expect. The reinforcement learning toolbox provides the $rlTrainingOptions()$ allows us to define the criteria for a well-trained agent as well as limit the number of steps per episode.

```
trainOpts = rlTrainingOptions('MaxStepsPerEpisode',800,...
'SaveAgentCriteria','EpisodeReward','SaveAgentValue',800);
```

This limits the maximum number of steps per episode as well as specifies that an agent with a cumulative reward of $J(\theta) = 1050$ is a successful agent.

5 Results

5.1 No agent

To provide a baseline for what flocking behaviour looks like figure 18 demonstrates flocking with constant gains for each step. the flocking can be seen to occur as drones are in proximity to each other and all drones are arriving at their individual destinations.

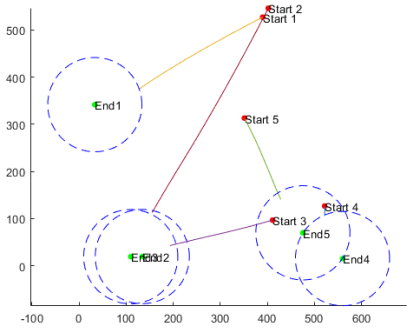


Figure 18: No agent Flocking

5.2 Agent training

During the training, the agents would behave in a stochastic fashion to facilitate exploration. The training was stopped once a satisfactory average reward was achieved and the subsequent agents were saved.

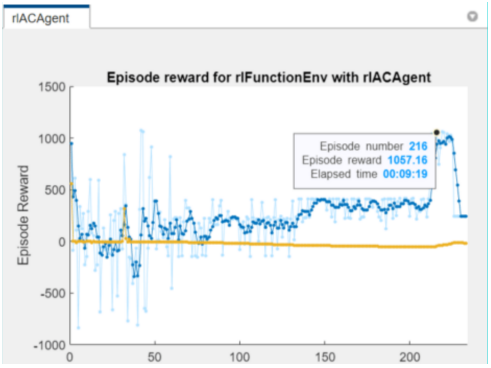


Figure 19: Agent Training

Figure 19 demonstrates this training where 3 values are shown: episode reward, an average taking into account the last 5 episodes and the Q0 estimated reward. The average reward, calculated over the last 5 steps, steadily increases as the agent performs more episodes. The agent learns steadily until it reaches a certain amount of episodes and suddenly loses all its learned behaviour. This is a common issue in reinforcement learning classified as "catastrophic forgetting"[45]. This might be corrected by varying the learning rate once a certain agent is achieved. This would mean the agent would have a more robust policy function although learn slower.

5.3 Trained agent

Once trained the agent behaves in a deterministic way as it has established the optimal policy function. This means for every state input the agent has a definite action to perform. The simulation is then repeatable as there is a definite action for every state.

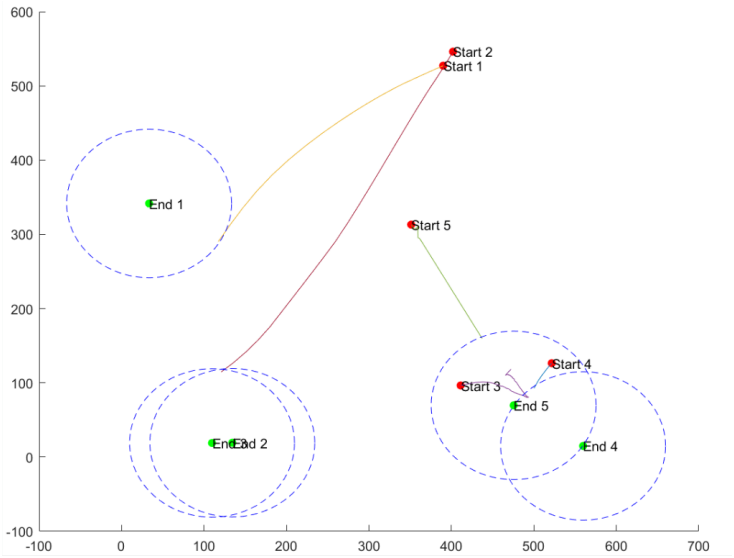


Figure 20: Trained Agent

5.4 Emergent Behaviour

Emergent behaviour is a qualitative analysis that is useful in understanding how the reward affects agent behaviour. If certain undesirable behaviours emerge this is a sign that the reward function is promoting the wrong actions. From the trained agent's performance in figure 20 some useful behaviours arise as well as some problematic behaviours:

- Flocking is achieved which shows drones 1 and 2 arriving at designated locations from sides that benefit both members, this might be useful if multiple drones need to deliver packages but also avoid certain areas.
- Some drones are not arriving at their destination which is a definite example of a scenario that is a detriment to the system. Drone 3 is choosing to flock with drones 4 and 5 rather than arrive at the destination.

6 Discussion

The actions that the agent can perform are what determine how the agent controls the system. Providing the agent with a guidance system constrains the possibilities of actions it can take which has positive and negative influences on the system:

+ The agent can learn quickly as there are fewer possible actions.

-The agent is constrained to certain motions and is not given enough degrees of freedom the system lacks controllability.[31]

Since the first publication in 1987[1], the boid flocking behaviour has been iterated upon to provide the agent with the ability to learn new behaviours. These include having different perception radii for separation, alignment and cohesion, limiting the field of view of boids and allowing for flock leaders.[2] These are just some examples of model variation that are possible. The application would be used to design the model. In the case of drones, lidar or some such sensor might be to detect the relative and heading of other UAVs. The simulation in this project uses a very simplistic model of the world. To reach further conclusions it would be desirable to further elaborate on the model. The reward function here focuses on only 2 characteristics: The position of the drone and the number of steps taken. The emergent behaviour exhibited by the drone is limited by this. Additional factors could be taken into account especially if the model representation expanded such as the power used to create a path. In this case, the reward function promotes drone flocking as well as arriving at individual destinations within a limited number of steps in any scenario. This simulation is in 2 dimensions, so the altitude is not considered. This would obviously play a role in deciding whether to flock if a drone is at a certain altitude or is descending towards its destination.
Positive emergent behaviour was achieved in some sense so to continue down this path it is important to establish why it is being successful and how the reward function and overall machine learning structure can be leveraged to further improve the algorithm.

6.1 Improvements

To further encourage the positive emergent behaviour as well as discourage the issues that arise multiple factors come into play

6.1.1 Machine learning structure

The structure of learning can affect the learning speed of the agent but also be used to determine the application of the agent. The current machine learning structure uses a centralised agent that controls n drones, providing a [5, N] gain matrix at each iteration step controlling the swarm. Instead of this a multi-agent structure can be defined where each agent controls a single drone. This would simulate a decentralised network which is better suited to a multi-vehicle system.

- The application could work with drones that are not in communication with each other if there is sufficient sensor information about other drones. This structure would also mean the number of drones would no longer affect the dimensions of action and state matrices creating a more generalised agent.
- Learning speed would increase as a result of agents learning in parallel, as for a simulation with N drones N agents would be trained as opposed to 1. To go even further

- Agents could be assessed individually rather than taking a weighted sum of the overall reward. Agents could be removed from every simulation and the best ones could be used to seed the next generation of agents. This would improve the learning rate as well.

6.1.2 Environment

In this case, the environment is simplistic. There are no obstacles included as well as very simple environmental dynamics. The system could be represented in 3D and additional environmental characteristics could be included such as:

- Differentiation of quad-copter[47] and fixed-wing[48] dynamics could be introduced to find how this affects the way they interact with each other.
- Consider benefits of flocking for efficiency for fixed-wing UAVs [48].
- Consider prevailing winds making use of head and tailwind.

6.1.3 Optimisation

Optimising the simulation is important in reinforcement learning as the more episodes that can be performed the more the agent can optimise its policy function. Code refactoring could be achieved through:

- Inclusion of a Quad-tree[49], which is a programming technique that compresses spacial data limiting the number of checks that need to be done when calculating whether agents are close to one another.
- Making use of specialised hardware that is optimised for RL[50].
- Using open AI platform to produce a better reinforcement learning structure.[51]

6.1.4 Reward Function

Being the key cause for emergent behaviour the reward function can be changed to take into account more of the environment. This clearly benefits from more in-depth environmental dynamics. The function that is used in this scenario is specifically tailored for the specific range. The reward function could be altered in many ways depending on the intended application, here are some possibilities:

- Make reward for position relative to the range that it will be used to make more general emergent behaviour.
- Include altitude in the 3-dimensional environment
- Include additional penalty for position, namely no fly-zones [14].

7 Conclusion

It has been demonstrated that by layering boid flocking behaviour and potential field functions a reinforcement learning agent can be trained to achieve emergent flocking behaviour which could be used to control drone traffic. Although the simulation is simplistic and the machine learning structure may not be optimal, this shows a basic framework is scale-able and could be layered with other reward functions to achieve more complex emergent behaviour.

7.1 Suggestions for further work

For further development of this idea one could look at how this system could be scaled further by using a further customised reinforcement learning algorithm. The loss function in this case is quite basic and therefore limits the learning capabilities. Using the Resources available on open AI make it easier to design a capable RL environment to suit specific requirements.[51] Making use of a Multi-Agent Particle Environment style of RL might also improve learning speeds and scale-ability.[52]

By simulating the environment in 3 dimensions and the addition of obstacles in the form of a city block plan could be used to emulate an environment closer to that intended for the agent.

References

[1] *Reynolds, C.W. (1987) Flocks, Herds and Schools: A Distributed Behavioral Model. ACM Siggraph Computer Graphics, 21, 25-34.*

[2] Craig W. Reynolds,
"Boids (Flocks, Herds, and Schools: A Distributed Behavioral Model)." Red3d.com, 2019, www.red3d.com/cwr/boids/.

[3] Bajec, I. Lebar, et al.
The computational beauty of flocking: boids revisited *"The Computational Beauty of Flocking: Boids Revisited." Mathematical and Computer Modelling of Dynamical Systems, vol. 13, no. 4, Aug. 2007, pp. 331–347, 10.1080/13873950600883485. Accessed 15 June 2021.*

[4] A.V. Moere,
Time-Varying Data Visualization Using Information Flocking Boids,
" IEEE Symposium on Information Visualization, 2004, pp. 97-104, doi: 10.1109/INFVIS.2004.65.

[5] Craig W. Reynolds Steering Behaviors For Autonomous Characters *Www.red3d.com, www.red3d.com/cwr/steer/gdc99/*

[6] Qu, Guannan. "Scalable Multi-Agent Reinforcement Learning for Networked Systems with Average Reward."

[7] Adam Daniel Laud Theory and Application of Reward Shaping in Reinforcement learning

[8] Drone Up,
"Let's Fly! | DroneUp." www.droneup.com/.

[9] Wing,
Owned by Google,
Wing. wing.com/.

[10] "DHL Drone Delivery and Parcelcopter Technology
Discover DHL." Dhl.com, 2022,
www.dhl.com/discover/en-my/business/business-ethics//parcelcopter-drone-technology.

[11] Skyopener Project.
M3 Systems, m3systems.eu/en/skyopener-project/. Accessed 16 Apr. 2022.

[12] Kirschstein, Thomas.
"Comparison of Energy Demands of Drone-Based and Ground-Based Parcel Delivery Services." Transportation Research Part D: Transport and Environment, vol. 78, 1 Jan. 2020, p. 102209, www.sciencedirect.com/science/article/pii/S1361920919309575, 10.1016/j.trd.2019.102209. Accessed 17 Mar. 2020.

[13] Gulden, Timothy R.
The Energy Implications of Drones for Package Delivery,
The Energy Implications of Drones for Package Delivery: A Geographic Information System Comparison. Santa Monica, CA: RAND Corporation, 2017.

[14] NO FLY DRONES
"No Fly Drones." No Fly Drones, www.noflydrones.co.uk/.

[15] NEXTECH
"VTOL Vertical Take off and Landing Drone." Nextechnology, shop.airbornedrones.co/products/vtol-drone. Accessed 16 Apr. 2022.

[16] DJI
"Mavic 3 - Specs - DJI." DJI Official, www.dji.com/uk/mavic-3/specs.

[17] ASSOCIATION OF THE UNITED STATES ARMY
"The Role of Drones in Future Terrorist Attacks." AUSA, 26 Feb. 2021, www.ausa.org/publications/role-drones-future-terrorist-attacks.

[18] Yaacoub, Jean-Paul, and Ola Salman.
"Security Analysis of Drones Systems: Attacks, Limitations, and Recommendations."
Internet of Things, May 2020, p. 100218, www.ncbi.nlm.nih.gov/pmc/articles/PMC7206421/, 10.1016/j.iot.2020.100218.

[19] D-fend
"Drone Threat, Counter Drone." D-Fend Solutions, www.d-fendsolutions.com/drone-threat-overview/. Accessed 16 Apr. 2022.

[20] Seongjoon Park et al.
Park, Seongjoon, et al. "Survey on Anti-Drone Systems: Components, Designs, and Challenges." IEEE Access, vol. 9, 2021, pp. 42635–42659, 10.1109/access.2021.3065926.

[21] Chao Yan et al.
Yan, Chao, et al. "Fixed-Wing UAVs Flocking in Continuous Spaces: A Deep Reinforcement Learning Approach." Robotics and Autonomous Systems, vol. 131, Sept. 2020, p. 103594, 10.1016/j.robot.2020.103594.

[22] Understanding Actor Critic Methods and A2C
Yoon, Chris. "Understanding Actor Critic Methods." Medium, 17 July 2019, towardsdatascience.com/understanding-actor-critic-methods-931b97b6df3f.,

[23] Deriving Policy Gradients and Implementing REINFORCE
Yoon, Chris. "Deriving Policy Gradients and Implementing REINFORCE." Medium, 23 May 2019, medium.com/@thechrisyoon/deriving-policy-gradients-and-implementing-reinforce-f887949bd63.,

[24] Perceptron,
Brilliant Math & Science Wiki."
" Brilliant.org, brilliant.org/wiki/perceptron/#: :text=The%20perceptron%20is%20a%20machine.

[25] Charu C. Aggarwal,
Machine Learning with Shallow Neural Networks. In: Neural Networks and Deep Learning. Springer, Cham., (2018).

[26] Deep Learning and the Game of Go
Pumperla, Max, and Kevin Ferguson. Deep Learning and the Game of Go. Shelter Island, Ny, Manning Publications Co, 2019.

[27] Sutton, Richard S,
and Andrew Barto. Reinforcement Learning : An Introduction. Cambridge, Ma ; Lodon, The Mit Press, 2018.

[28] Grondman, Ivo, et al.
"A Survey of Actor-Critic Reinforcement Learning: Standard and Natural Policy Gradients." IEEE Transactions on Systems, Man, and Cybernetics, Part c (Applications and Reviews), vol. 42, no. 6, Nov. 2012, pp. 1291–1307, 10.1109/tsmcc.2012.2218595. Accessed 27 Feb. 2020.

[29] "Create MATLAB Reinforcement Learning Environments - MATLAB & Simulink."
www.mathworks.com/help/reinforcement-learning/ug/create-matlab-environments-for-reinforcement-learning.html.

[30] Stengel, Robert F.
Optimal Control and Estimation.
Google Books, Courier Corporation, 20 Sept. 1994, books.google.co.uk/books?hl=en&lr=&id=byRgDwAAQBAJ&oi=fnd&pg=PA1&dq=Stengel+Optimal+Control+and+Estimation. Accessed 16 Apr. 2022.

[31] Control Systems/Controllability and Observability,
Wikibooks, Open Books for an Open World."
En.wikibooks.org, en.wikibooks.org/wiki/Control_Systems/Controllability_and_Observability#: :text=The%20concept%20of%20controllability%20refers.

[32] "Boids Algorithm.
" Eater.net, eater.net/boids.,

[33] "The Nature of Code."
Natureofcode.com, natureofcode.com/book/chapter-6-autonomous-agents/.

[34] "The Nature of Code."
Natureofcode.com, natureofcode.com/.

[35] Metropolis, Nicholas, and S. Ulam.
"The Monte Carlo Method." Journal of the American Statistical Association, vol. 44, no. 247, Sept. 1949, pp. 335–341, 10.1080/01621459.1949.10483310.

[36] Littman, M. L.
"Markov Decision Processes."
ScienceDirect, Pergamon, 1 Jan. 2001, www.sciencedirect.com/science/article/pii/B0080430767006148.

[37] Fujimoto, Scott, et al.
Addressing Function Approximation Error in Actor-Critic Methods.

[38] Silver, David, et al.
Deterministic Policy Gradient Algorithms.

[39] TORRES.AI, Jordi.
"The Bellman Equation."
The Bellman Equation
Medium, 15 June 2020, towardsdatascience.com/the-bellman-equation-59258a0d3fa7.

[40] *"Bellman Optimality Equation in Reinforcement Learning." Analytics Vidhya, 13 Feb. 2021, www.analyticsvidhya.com/blog/2021/02/understanding-the-bellman-optimality-equation-in-reinforcement-learning/#: :text=The%20Optimal%20Value%20Function%20is. Accessed 16 Apr. 2022.*

[41] Hood, Jordan J.
"Reinforcement Learning: Temporal Difference (TD) Learning."
www.lancaster.ac.uk/stor-i-student-sites/jordan-j-hood/2021/04/12/reinforcement-learning-temporal-difference-td-learning/.

[42] Deshpande, Paresh, et al.
Formation Control of Multi-Agent Systems with Double Integrator Dynamics Using Delayed Static Output Feedback."
" IEEE Xplore, 1 Dec. 2011, ieeexplore.ieee.org/document/6161074/figures#figures. Accessed 16 Apr. 2022.

[43] Brummerloh, Daniel.
"Ultimate Guide for AI Game Creation Part 2 — Training."
Medium, 7 Jan. 2021, towardsdatascience.com/ultimate-guide-for-ai-game-creation-part-2-training-e252108dfbd1. Accessed 16 Apr. 2022.

[44] "Create MATLAB Reinforcement Learning Environments - MATLAB & Simulink."
Www.mathworks.com, www.mathworks.com/help/reinforcement-learning/ug/create-matlab-environments-for-reinforcement-learning.html. Accessed 16 Apr. 2022.

[45] Cahill, Andy.
"Catastrophic Forgetting in Reinforcement-Learning Environments."
Ourarchive.otago.ac.nz, 2011, ourarchive.otago.ac.nz/handle/10523/1765. Accessed 16 Apr. 2022.

[46] Fridman, Lex.
"MIT Deep Learning Basics: Introduction and Overview."
YouTube, 11 Jan. 2019, www.youtube.com/watch?v=O5xeyoRL95U.

[47] "Dynamics Modeling of a Highly-Maneuverable Fixed-Wing UAV."
Researchgate, Apr. 2013, www.researchgate.net/publication/264773227_Dynamics_Modeling_of_a_Highl Maneuverable_Fixed-Wing_UAV.

[48] "Accurate Simulator for Motion of the Quadcopter; Assuming Dynamic and Aerodynamic Effects."
*Esearchgate, Apr. 2019, www.researchgate.net/publication/332980824_Accurate_Simulator_for_Motion_*

[49] Yin, Xiang, et al.
"Quadtree Representation & Compression of Spatial Data."
Www.semanticscholar.org,
www.semanticscholar.org/paper/Quadtree-Representation-%26-Compression-of-Spatial-Yin-Diintsch/d45d7f7d8817170ee0f9cec2db083a4852848621?p2df.

[50] Shiri, Aidin, et al.
"Energy-Efficient Hardware for Language Guided Reinforcement Learning."
Proceedings of the 2020 on Great Lakes Symposium on VLSI, 7 Sept. 2020,

eehpc.csee.umbc.edu/publications/pdf/2020/2020_GLSVLSI_RL_Structured_Language.pdf, 10.1145/3386263.3407652. Accessed 16 Apr. 2022.

[51] https://openai.com/api/

[52] Huichu Zhang Shanghai Jiao Tong University Shanghai, et al. "CityFlow: A Multi-Agent Reinforcement Learning Environment for Large Scale City Traffic Scenario: The World Wide Web Conference." ACM Other Conferences, 1 May 2019, https://dl.acm.org/doi/abs/10.1145/3308558.3314139?casa_token=nbe2mKJTQcEAAAAA%3AJsYLiuB
DQ4YFfC0WLX0CRlDuj1UJnyeUpdcFouJDvfeosKk64t08rUAfQKStvFnwPB0Ieg.

A Appendices

A.1 Reward Function

A.1.1 Flocking reward

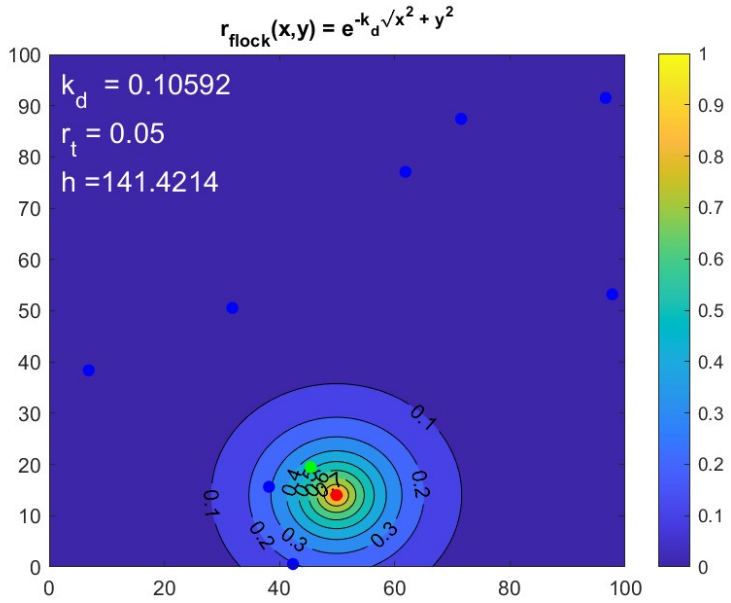


Figure 21: Reward for endpo

A.1.2 Endpoint Reward

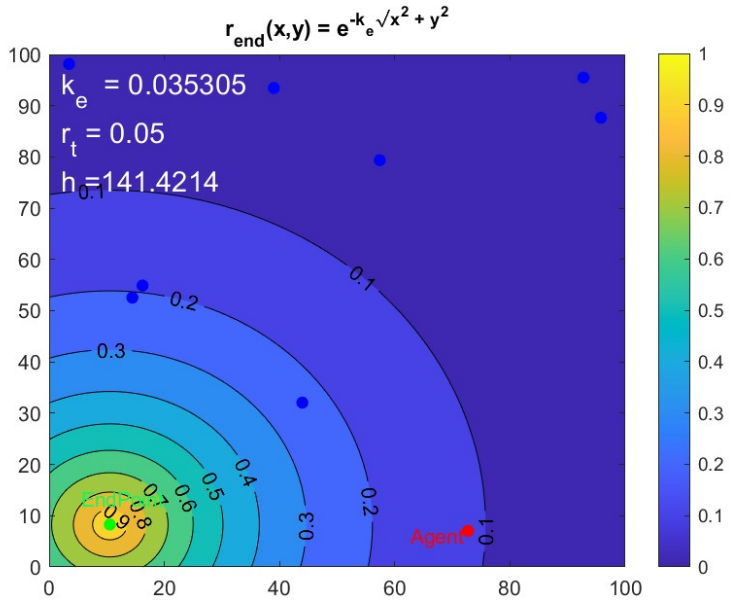
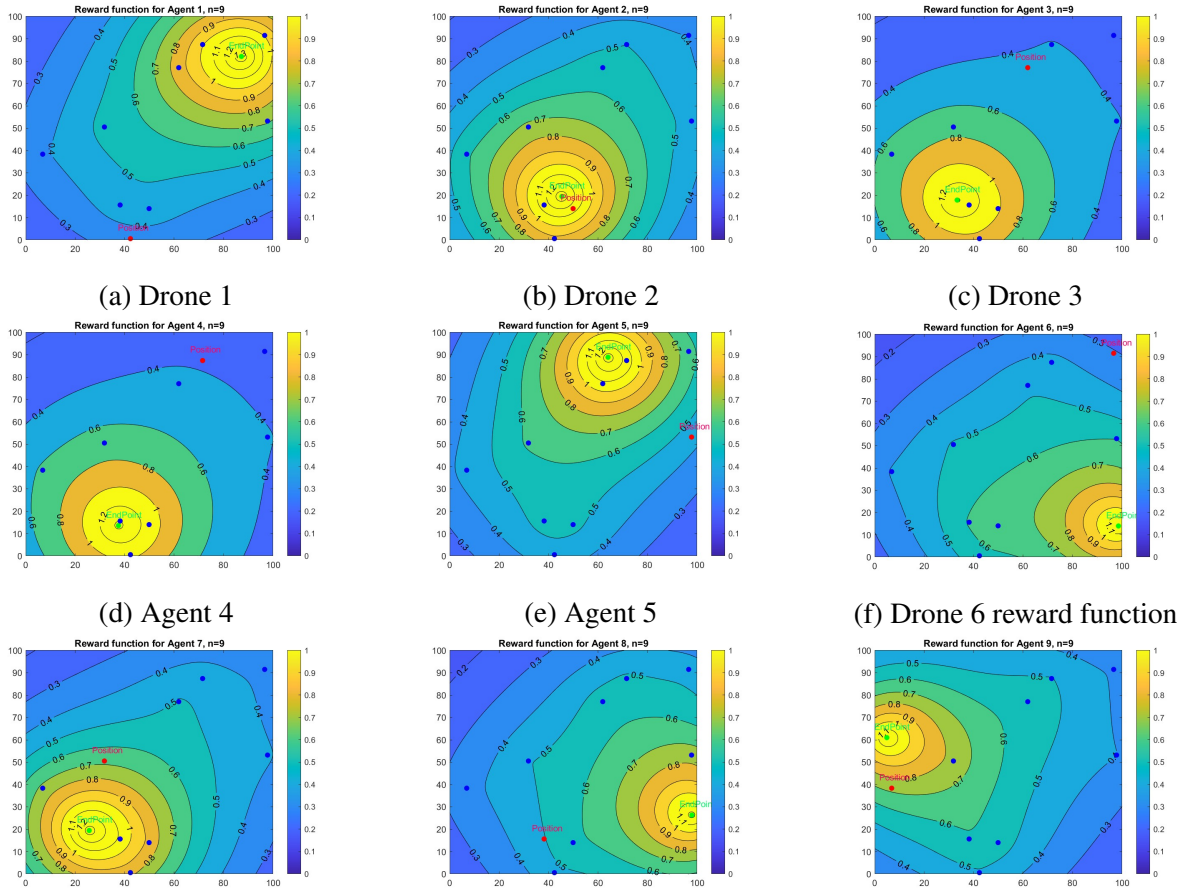


Figure 22: Caption

A.1.3 Individual reward function for 9 drones



(a) Drone 1 (b) Drone 2 (c) Drone 3

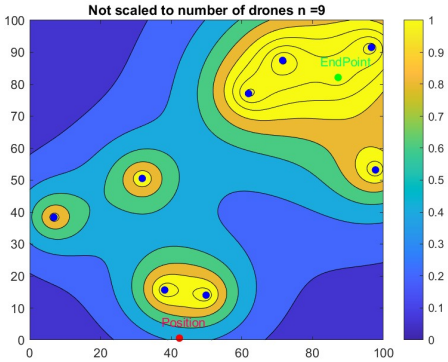
(d) Agent 4 (e) Agent 5 (f) Drone 6 reward function

(g) Agent 7 (h) Agent 8 (i) Agent 9

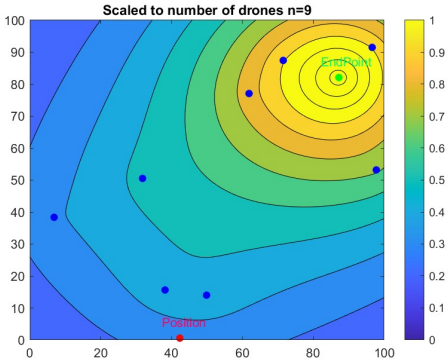
Figure 23: Reward functions for 9 different Agents

A.1.4 Scaling reward function relative to number of agents

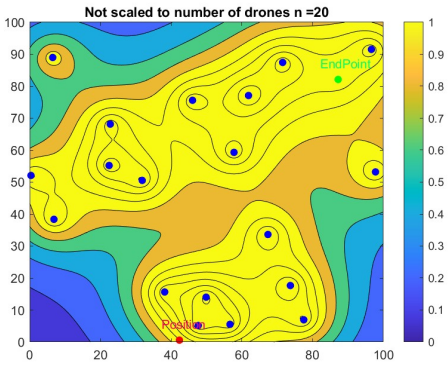
Comparison of reward function with no scaling relative to number of entities:



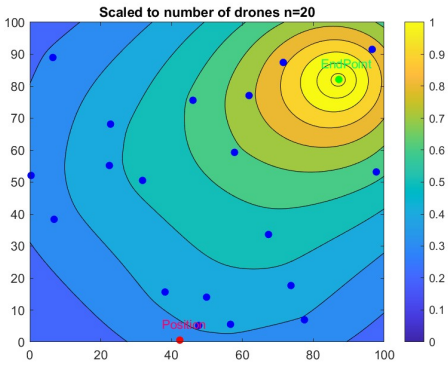
(a) Not scaled n = 9



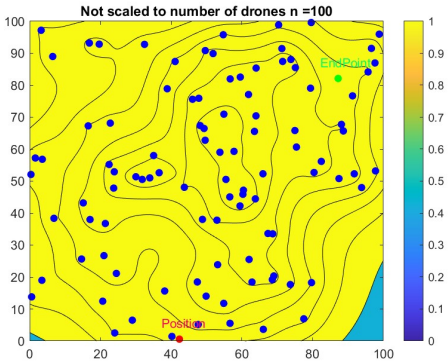
(b) Scaled n = 9



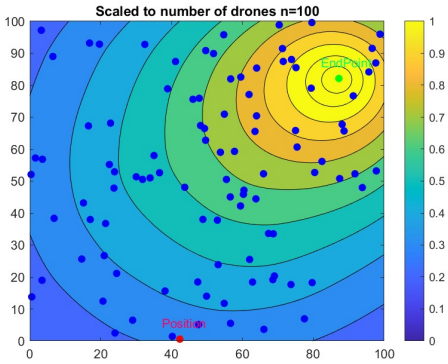
(c) Not scaled n = 20



(d) Scaled n = 20



(e) Not scaled n = 100



(f) Scaled n = 100