UNIVERSITY OF BATH

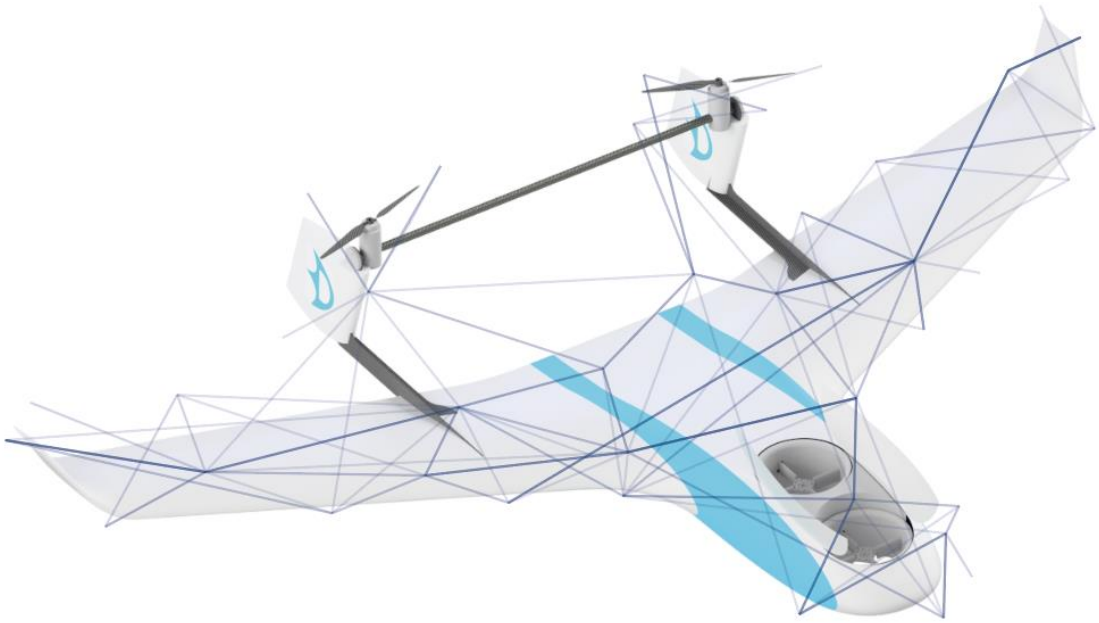
Faculty of Engineering and Design

Machine Learning for Aircraft Control

Author: Michael Morris

Supervisor: Dr P. Iravani
Assessor: Dr R. Dunn

Date: 09/05/19



Summary

The aim of this project is to control a tilt rotor UAV through its transition between vertical and horizontal flight using modern Artificial Intelligence (AI) techniques. Traditional controllers are expert systems which require the designer to program the controller to work in every potential scenario. These are limited and tend to fail catastrophically when an unforeseen event occurs. Learning systems learn solutions from first principles with no prior knowledge. This makes them more robust than hard coded equivalents, and they can be more generalised to solving tasks which we may not ourselves know how to solve. This project uses reinforcement learning as a tool to control a winged, tilt-rotor, UAV(1). To enable this a flight dynamics model of the aircraft capable of multirotor and fixed wing flight was created, and used to interact with, and as a training tool for, the AI algorithm. The goal of the AI is to act as a pilot providing suitable controller inputs and weighting outputs through an accelerating transition.

Many different methods of dynamic programming were considered for controlling the aircraft. Ultimately reinforcement-learning algorithms were found to be the most suitable candidate for this problem, although challenging to implement. Deep deterministic policy gradients (DDPG) are a reinforcement-learning algorithm that works in both continuous state and action space, a key requirement for this aircraft. DDPG was selected as the best candidate for this application as it is a state-of-the-art technique which does not require discretisation: it was implemented using MATLAB's (Beta) reinforcement learning toolbox.

Initial tests highlighted the importance of stability in the simulation of the aircraft, as initial exploration in reinforcement learning often leads to highly unusual control inputs that can cause the simulation to slow down or crash. Despite working adequately with conventional control inputs, considerable debugging was required of all aspects of the simulator to make it fast enough, and robust enough, to be used with reinforcement learning.

Testing of the DDPG algorithm highlighted how difficult to implement it is. The majority of the maths is handled by MATLAB, yet considerable work was required to get the hyper parameters suitable for training. To simplify what was required of the agent it was first tested on a multirotor only version of the aircraft. This initially had one degree of freedom in the x-axis and could only control forward velocity. Following success in one dimension, the problem was gradually made more complex by adding more degrees of freedom and reverting to attitude controllers. Ultimately the agent was trained to fly between points in multirotor mode, although not as successfully as a well-tuned PID controller. When the agent was as capable as reasonably possible in multirotor-mode, the transition was attempted.

The transition is by far the most complex part of the flight: to simplify it the agent was started off as the best performing multirotor agent, and the ability to control the tilt angle of the motors and flight controller weights was added. The aircraft was then tasked with flying a trajectory which is only possible in forward mode, with negative rewards given for any attempts as a multirotor. This open-ended approach gave novel solutions to the transition, and ultimately was comparable to a known flight controllers approach.

The project gave good results but did not manage to improve on a well-tuned conventional controller. The reinforcement learning agent successfully learnt good policies with no prior experience and demonstrated examples of 'intelligent' behaviour such as forward planning.

(1) Throughout this report 'aircraft' is used to describe a winged unmanned aerial vehicle (UAV) capable of both vertical and horizontal flight.

List of Figures

List of Tables

1. Introduction

Unmanned Aerial Vehicles (UAVs) are becoming ever more prominent in society. This relatively new technology is predominantly used for cinematography, with large-scale quadcopters carrying cameras for movie and television. The limited applications are often put down to regulations which restrict many potential operations such as surveying, aid delivery and conservation [1, 2, 3]. Many companies are preparing for a future where more sophisticated regulation will allow drones to play a role in these areas [4, 5].

UAVs typically fall into two distinct categories; those which use wings as their lifting surfaces and those which use powered lift. Fixed wing UAVs fly quickly and efficiently over long distances and are often similar to conventional aeroplanes and flying wings. Powered lift UAVs can fly precisely at low and zero speed, but are not suited to long distance or high-speed flight due to the inherent inefficiencies which come from generating lift with motors. The third, less common type of UAV are those that combine both modes: they take off and land vertically (VTOL), and transition to forward flight using wings as their lifting surfaces for efficiency and speed. These are a 'best of both worlds' solution with many advantages. The main disadvantage is the complexity introduced in finding an optimum combination of two very different aircraft types.

Team Bath Drones is the University of Bath's student lead UAV team, which in 2019 is attempting to make a VTOL tilt rotor aircraft for entry in the Institution of Mechanical Engineers (IMechE) UAS challenge. This aircraft, 'Caelus', uses two rotors to provide propulsion in forward flight that can tilt to vertical and, in combination with two coaxial ducted fans, provide lift to take off and land vertically. The two front motors can be approximated to a single lift force as shown below:

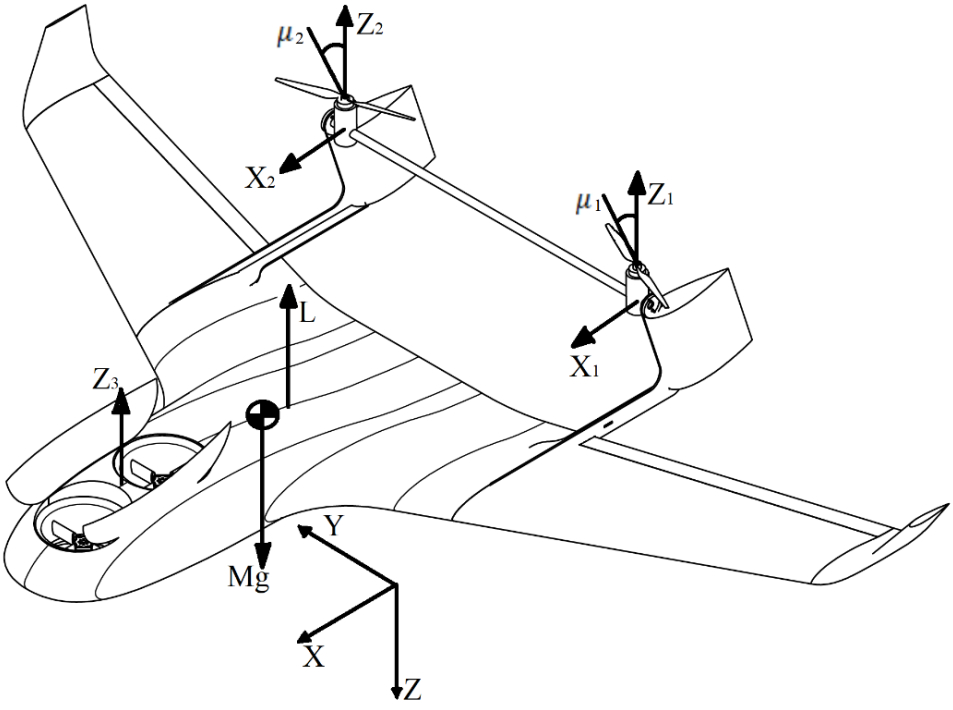


Figure 1: Aircraft Force Diagram

As the aircraft transitions from vertical to forward flight the two tilting motors go from controlling yaw and roll by varying their angles and thrust respectively, to varying thrust and angle respectively instead. This is summarised in Table 1:

Control Output	Function in Helicopter Mode	Function in Aeroplane Mode	Function in Transition
Tilt left motor	Yaw	Roll + Pitch	Pitch + Yaw + Roll
Tilt right motor	Yaw	Roll + Pitch	Pitch + Yaw + Roll
Thrust left motor	Roll + Pitch	Yaw	Pitch + Yaw + Roll
Thrust right motor	Roll + Pitch	Yaw	Pitch + Yaw + Roll
Thrust Duct	Pitch	-	Pitch
Left elevon	-	Roll + Pitch	Roll + Pitch
Right elevon	-	Roll + Pitch	Roll + Pitch

Table 1: Control Functions in Different Flight Modes [6]

This provides a complex control problem with coupling between different control-modes. They are controlled by using two distinct controllers for helicopter mode and forward flight, with outputs that are weighted based on the aircraft's stage of flight. The weights would usually be found via a look up table based on aircraft speed and tilt angle. The goal of this project is to control the aircraft through this challenging transition using methods based on artificial intelligence. The controller will set the control weights and generate an RC signal, effectively behaving as a pilot.

This control problem needs a simulation of the aircraft; it does not need to exactly model the physical aircraft because:

- The primary objective is to do with machine learning (ML), and not simulation; therefore, a working simulation with similar characteristics will suffice.
- The aircraft configuration has not been finalised. Material changes to the aerodynamic and propulsion systems of the physical aircraft have been made in parallel with this project. These changes have been modelled where necessary and where time permitted.
- Reinforcement learning algorithms take an extremely long time to train; adding complexity to the simulation increases training time
- Reinforcement learning should be applicable to a broad variety of uses. As long as the simulation is similar to true aircraft, the agent should be able to react to changes and tune itself to adapt to them.

With a working simulation the machine learning control problem can be tackled. It is not possible to interface Python and MATLAB/Simulink directly due to key differences in the way in which they handle data. This means that machine learning libraries such as TensorFlow cannot be used. The ML algorithms are therefore more challenging to implement with less documentation. The selection of the best algorithm is paramount. To do this a clear understanding of the problem is needed.

For the AI to be useful it must be superior at completing an accelerating transition to a conventional controller. 'Caelus' contains a Pixhawk flight controller which runs ArduPilot firmware. This controller is the benchmark which the AI must improve on. The ArduPilot transition is implemented in the flight dynamics model and it is tuned as if it were a physical aircraft. The aircrafts pitch and yaw are coupled while the motors are tilting, which makes the aircraft more unstable in these axes. A good controller would account for these instabilities, for instance by losing altitude during the transition to increase speed. To a skilled human pilot this would be intuitive, for a computer with no prior experience it is extremely challenging. The work done in this report starts to solve these problems but much more testing and validation would be necessary before implementing an ML based controller into a physical aircraft.

2. Related Works

VTOL aircraft are inherently difficult to control; they have multiple flight modes and often coupling between their control outputs. Tilt rotors in particular have poor stability due to non-linear coupling between their control surfaces [7]. During the transition between flight modes there are two controllers outputting weighted actions, each tuned for different dynamics [8]. Both of these must be understood with balanced weights for the controller be effective [9]. As the aircraft accelerates in its forward transition it becomes more like an aeroplane and less like a multirotor or helicopter [10]. Balancing the controllers by adjusting weightings is a serious challenge that has been approached in many ways [11, 12, 13]. 'Caelus', the aircraft being controlled in this paper, uses a Pixhawk 2 flight controller running ArduPilot firmware in tilt rotor mode. In the transition the aircraft is gradually accelerated while being held as stable as possible, it accelerates past its stall speed and the motors at the rear quickly rotate to horizontal and it continues in aeroplane mode [14].

Issues with controlling an aircraft's transition are discussed at length in the AV-8B Harrier pilot's manual [15]. Fundamentally the aircraft requires precise and delicate control inputs, and at low speed can quickly start to side slip and become uncontrollable. This has made it the most dangerous military jet in history [16].

Like the Harrier, 'Caelus' is optimised neither for horizontal nor vertical flight, as optimising for one would worsen the other flight mode. Controlling VTOL aircraft requires a novel control approach and computational methods provide this. Modelling of the aircraft enables a computational controller to be taught the aircraft's dynamics without flying, and likely crashing, the physical aircraft [17, 18]. Having this simulation enables an agent to be able to fly the aircraft by learning by imitation [19], by reinforcement [20] or otherwise [21, 22].

In the project planning and background review [6] several methods of applying machine learning to this aircrafts control problem were identified. The controller has to generate RC signals for the aircraft, effectively replacing the pilot. Previously model predictive control was identified as potential candidate for this [21, 22]. It would require a physical aircraft to be 'learnt' using neural networks and supervised learning in a variety of control tasks which could later be optimised for the transition [23]. This was impossible due to the lack of a physical aircraft and does not produce a controller as adaptable as others.

A better solution was to use learning by imitation to copy an experienced pilot in a simulation and then attempt to replicate these inputs to behave in a similar way to how a pilot would in a variety of situations. This has been used with success [19, 24] but is limited to modes where a pilot is able to fly the aircraft. With novel aircraft such as 'Caelus' this is not possible. This approach would be better suited to controlling the aircraft in forward flight mode for tracking pre-determined flight paths generated via intelligent methods [25, 26, 27] in a more human like way [28].

The project planning report identified reinforcement learning (*RL*) as the best control method for this project. With RL, an agent interacts with an environment and attempts to maximise a reward. It explores and attempts to find the best path through the environment. This makes it a branch of machine learning well suited for control. It is used to find 'optimal controllers of systems with nonlinear, possibly stochastic dynamics that are unknown or highly uncertain' [29, 30]. There are countless ways of applying reinforcement learning RL to control problems and these are discussed further in section 5.

3. Flight Dynamics Model

3.1 Introduction

The physical model uses a six degrees of freedom (6DOF) aerospace block from Simulink; this receives forces and moments about the centre of gravity (cg) of the aircraft. These forces are produced by aerodynamic forces (lifting surfaces and control surfaces), propulsive forces, ducts at the front and center of the aircraft, tilting rotors at the rear and the mass effects of the aircraft. The orientations of these forces are described by three references frames. The equations of motion are then calculated with respect to these. The frames are chosen by convention, as described in [31].

i. Body Axes – Positioned on the cg, with X facing forward, Z pointing down, and Y to the aircrafts starboard. Rotating about these axes are roll, pitch and yaw.
ii. Stability Axes – This is a North-East-Down frame on the earth's surface, with X pointing north, Y pointing east and Z pointing towards the earth's centre.
iii. Wind axis – This axis always follows the wind direction with respect to the aircraft. This rotation is described by the aircraft's angle of attack (α) and sideslip angle (β).

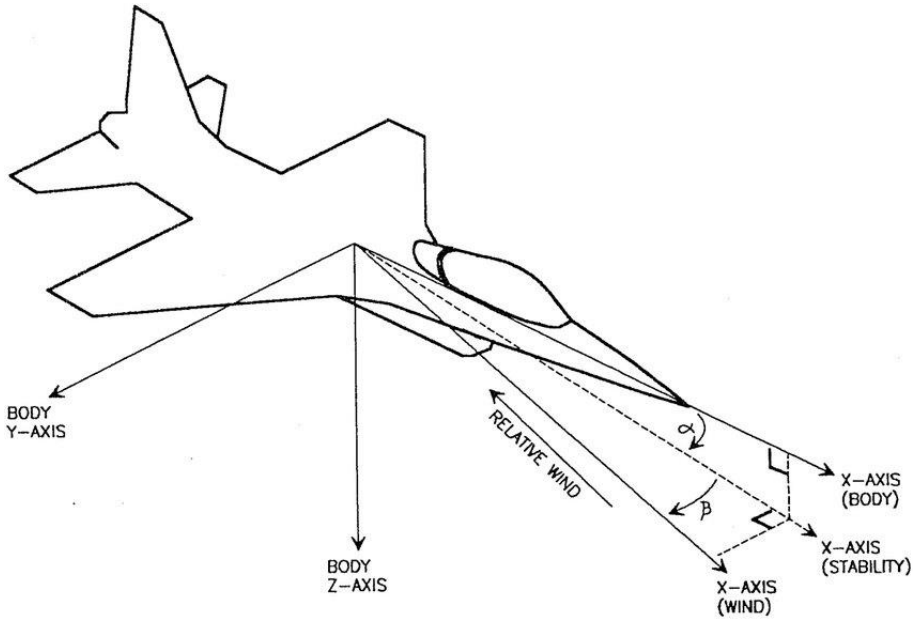


Figure 2: Aircraft Reference Frames [31]

These reference frames allow forces and moments to be calculated with respect to each inertial frame which are then rotated with respect to one another. These forces are:

$$\sum F = m(\dot{v}_b + \omega \times v_b) \tag{3.1.1}$$

$$\sum M = I\dot{\omega} + \omega \times I\omega \tag{3.1.2}$$

3.2 Aerodynamic Forces and Moments

The aerodynamic forces are related to the angle of attack (α) and moments come from the lift, drag and pitching moments. These are calculated by the following:

$$L = \bar{q}SC_L(\alpha) \quad (3.2.1)$$

$$D = \bar{q}SC_D(\alpha) \quad (3.2.2)$$

$$M = \bar{q}S\bar{c}C_M(\alpha) \quad (3.2.3)$$

The dynamic pressure is given by:

$$\bar{q} = \frac{1}{2}\rho V_T^2 \quad (3.2.4)$$

Where:

$\rho =$ air density (1.225$\frac{kg}{m3}$ at sea level)
$V_T =$ free stream airspeed
$c =$ wing mean geometric chord
$b =$ wing span
$S =$ wing reference area

The dimensionless lift, drag, and moment coefficients (C_l, C_d, C_m) describe how effective the aerofoil is at producing lift, minimising drag and resisting changes in α [31]. These forces are implemented in the Simulink model using the aerospace block set to take care of the majority of the maths. The coefficients are calculated from computational fluid dynamics (CFD) data.

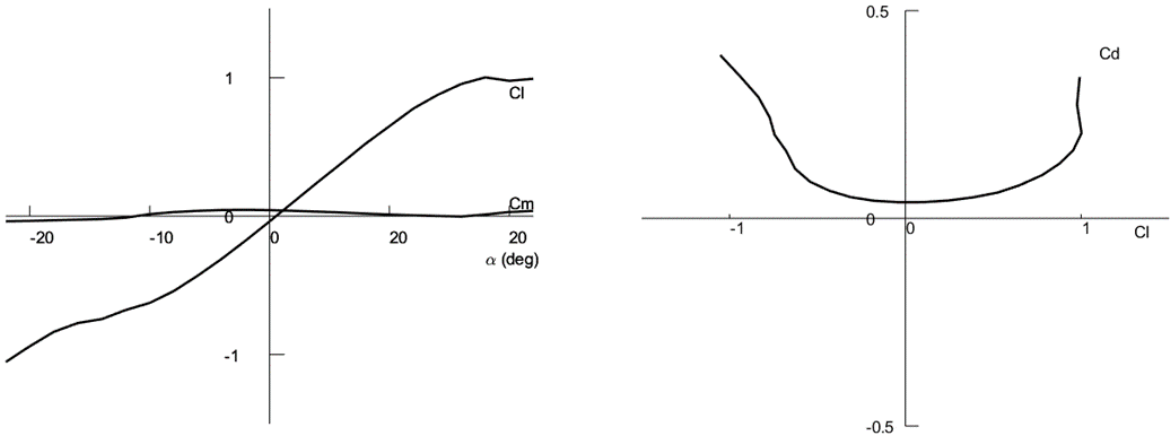


Figure 3: Plots of lift, drag and moment coefficients

The side forces can be calculated in a similar way using the sideslip angle β. The side force, rolling moment, and yawing moments are calculated as follows:

$$Y = \bar{q}SC_Y(\beta) \quad (3.2.5)$$

$$\bar{L} = \bar{q}SbC_l(\beta) \quad (3.2.6)$$

$$N = \bar{q}SbC_N(\beta) \quad (3.2.7)$$

These were implemented in the Simulink model but it was found that their effects are minimal in the majority of conditions and they dramatically slow down the simulation. This would not usually be an issue but as it must be run hundreds of times during training, it is useful to remove these forces from the simulation.

i. Elevons

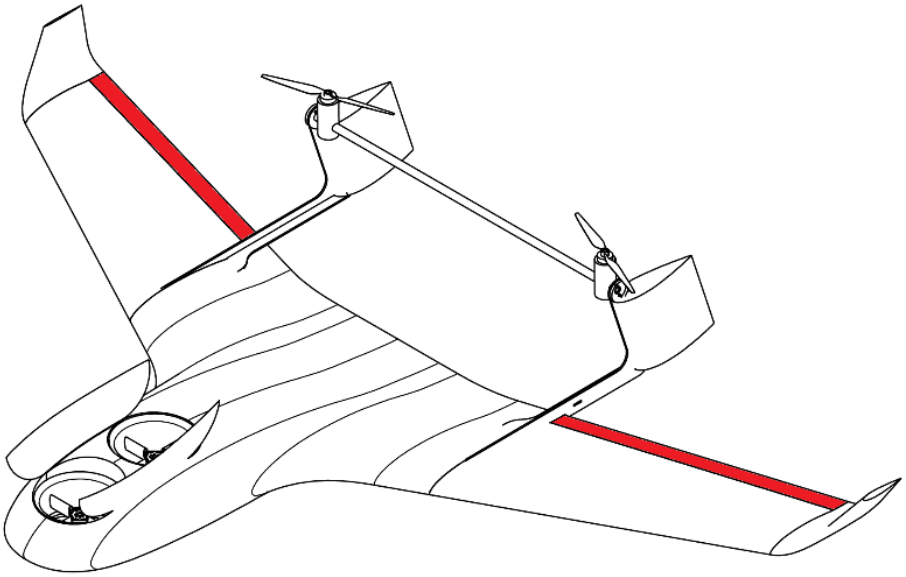


Figure 4: Elevon locations

The aircraft has two elevons for pitch and roll control in forward flight mode. These produce a force proportional to the velocity squared, elevon area, sum of the deflection angle (δ) and angle of attack (α):

$$F_x = \bar{q}AC_d(\delta + \alpha) \tag{3.2.8}$$

$$F_z = \bar{q}AC_l(\delta + \alpha) \tag{3.2.9}$$

This in turn produces a moment about the cg:

$$M = L \times F \tag{3.2.10}$$

$$\begin{bmatrix} L_L \\ L_R \end{bmatrix} = \begin{bmatrix} -0.438 & -0.505 & 0 \\ -0.438 & 0.505 & 0 \end{bmatrix}$$

These force and moment equations were taken from another flying wing Simulink model [32]. The elevons produced excessive drag in simulation so this was scaled by 0.1. An alternative to this would be to use CFD data and a look-up table to update the coefficients from the previous section. This would be more accurate, but for the purposes of this simulation is not required as the primary goal of the project is to control the aircraft using reinforcement learning and not to create a highly accurate Simulink model.

The Simulink aerodynamic forces block is validated by comparing it to known inputs:

- At zero velocity there are no forces or moments
- At 25m/s there is an expected drag force of 13N, which increased to 18N at 30m/s. The lift produced also increases from 13N to 18N
- Using the elevons adds a moment to the pitch axis
- Vertical movement produces high forces in both X and z.

3.3 Propulsive Forces

The propulsive forces from the motors are:

$$[F_x \quad F_y \quad F_z] = F[\sin\theta \quad 0 \quad -\cos\theta] \tag{3.3.1}$$

Where θ is the angle of tilt of the motor. The ducted fans cannot tilt: their force acts entirely in the $-Z$ direction. The moments are calculated by:

$$[M_x \quad M_y \quad M_z] = [L_x \quad L_y \quad L_z] \times [F_x \quad F_y \quad F_z] \tag{3.3.2}$$

For the three motors these lengths are:

$$\begin{bmatrix} L_1 \\ L_2 \\ L_3 \end{bmatrix} = \begin{bmatrix} -0.544 & -0.372 & 0 \\ -0.544 & 0.372 & 0 \\ 0.392 & 0 & 0 \end{bmatrix}$$

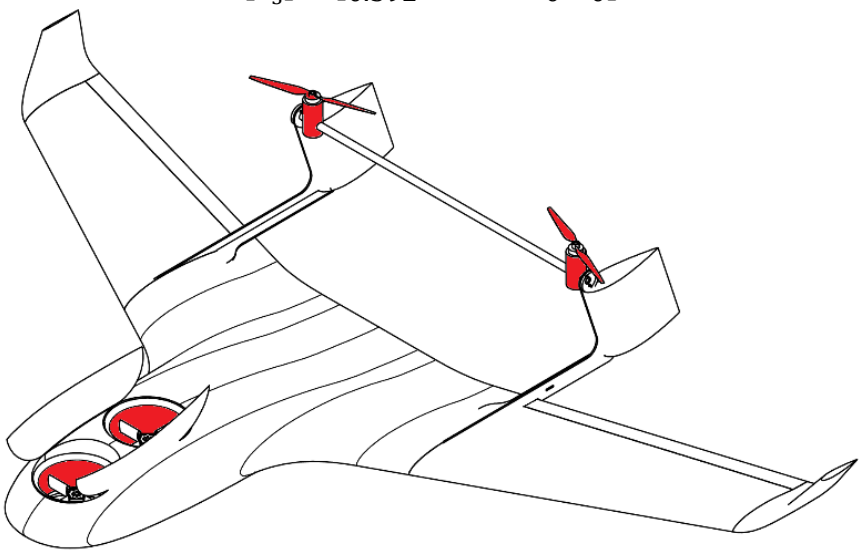


Figure 5: Propulsive forces

The motors at the rear produce a force proportional to the velocity as follows:

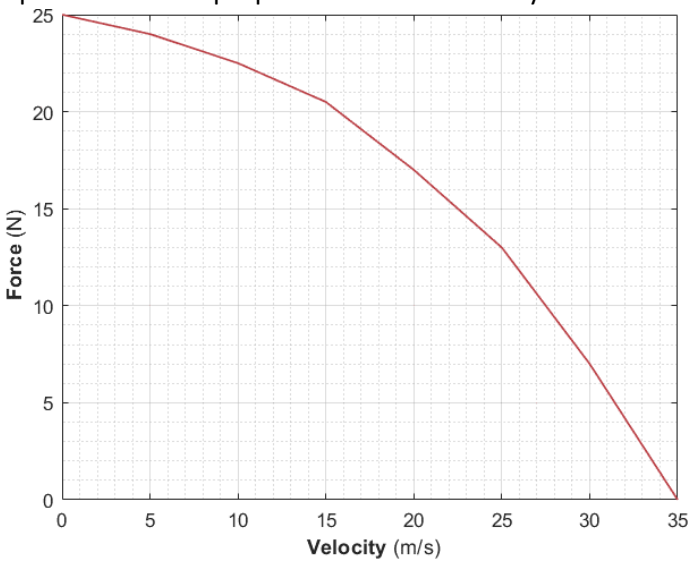


Figure 6: Tilt-rotor force at different airspeeds

The air velocity over the propellers is related to the aircraft velocity:

$$\sqrt{{V_x}^2 + {V_z}^2}\cos\left(\tan\left({}^{V_z}/_{V_x}\right) + \theta\right) \tag{3.3.3}$$

The front ducted fans produce the same force at all velocities, as the ducts make the induced velocity much higher and less susceptible to movement. The front ducts produce twice the maximum force from the tilting rotors to give a zero-pitching moment when the control pitch PID controller is 0. There are moments caused by the inertia of the propellers which is used to control yaw in quadcopters. These forces could be modelled as a proportion of the force about the rotational axis of the propeller: as the motors at rear turn at the same speed in a hover, and the motors at the front are counter rotating, this force is trivial and has been ignored for the sake of speeding up the simulation. It has been shown that the effects of this force are minor with similar aircraft; during testing of the multirotor-only version of the aircraft one of the counter rotating propellers broke, but this did not result in a noticeable reduction in flight performance and the aircraft was still able to hover as shown in Figure 7.

Figure 7: Multirotor mode aircraft after losing a propeller.

The propulsive force Simulink block is validated by comparing it to known inputs:

- At zero velocity, with all motors on full and the motors 1 and 2 vertical, they produce a total force of 100N vertically, 0 in other directions and no moments.
- With the motors tilted to horizontal at 25m/s the motors make 13N each and push the aircraft forward with the expected moments.
- With the motors tilted to horizontal at 25m/s, vertically the motors make 25N each and push the aircraft forward with expected moments.
- With both motors at 45 degrees and the aircraft moving at 25m/s in the same direction, the motors produce the expected forces in X and Z with the expected moments.

4. Flight Controller Model

4.1 Multirotor Controller

The multirotor controller has three levels, a rate controller, attitude controller and velocity controller. The rate controller receives desired rates and outputs motor forces and angles. There are PID loops for vertical velocity V_{b_z} (throttle) and roll, pitch, and yaw rates $(\emptyset_r, \theta_r, \psi_r)$. These PID outputs are mixed to motor and angle outputs, as shown below:

Force 1	Force 2	Force 3	Angle Motor 1	Angle Motor 2
$-V_{b_z}$	$-V_{b_z}$	$-V_{b_z}$	-	-
$\emptyset_r$	$-\emptyset_r$	-	-	-
$-\theta_r$	$-\theta_r$	θ_r	-	-
-	-	-	$-\psi_r$	$-\psi_r$

Table 2: Multirotor rate controller mixing

Each output is held between two limits, as shown below, so that the forces do not saturate and remove control authority from another controller.

Control	V_{b_z}	$\emptyset_r$	θ_r	ψ_r
Lower limit	0	0	0	-2°
Upper limit	0.6	0.2	0.2	2°

Table 3: motor force and angle upper and lower limits

The forces are limited to $\in [0,1]$, so that the motor will always be turning in the same direction and will not generate negative lift that would cause it to go unstable. The minimum and maximum angles are constrained to $\in [-2,2]$, as it was found that the closer to zero these are the faster the simulation could run. It is assumed that these move instantly to the desired position, in practice this would not be true. Adding a delay would make the yaw PID controller harder to tune and the simulation run more slowly.

The attitude controllers are based on the ArduPilot controller although with constraints on the maximum rate and with no feed forward controller:

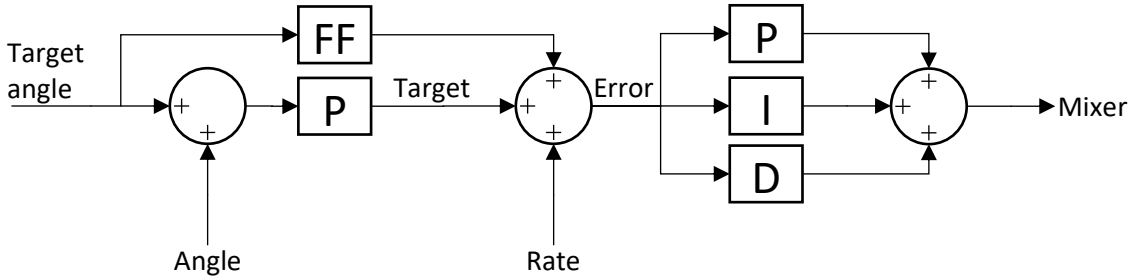


Figure 8: ArduPilot attitude controller [50]

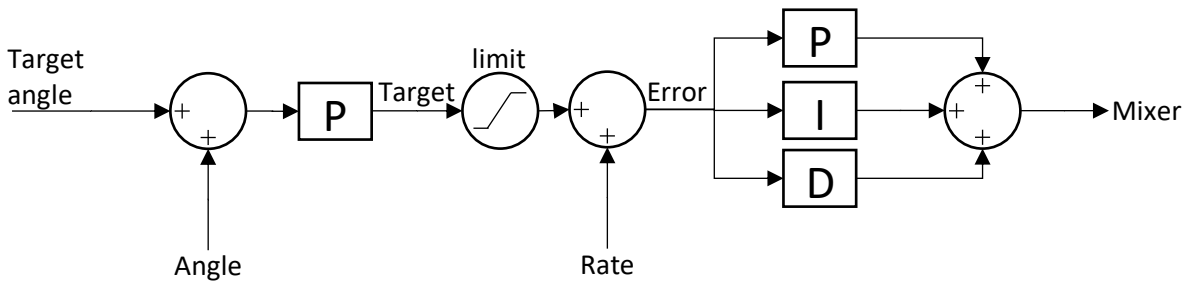


Figure 9: Simulink attitude controller

i. Rate Controller

The rate controller was tuned to a step response with a maximum rate of $30°/s$ for $\emptyset_r, \theta_r, \psi_r$ and $1m/s$ for V_{b_z}. These rates were chosen as reasonable limits for the aircraft that would ensure the simulation is stable. The step response is much faster on pitch and roll than on yaw, due to the increased authority caused by the way the aircraft is controlled. This is expected and is true on the real aircraft:

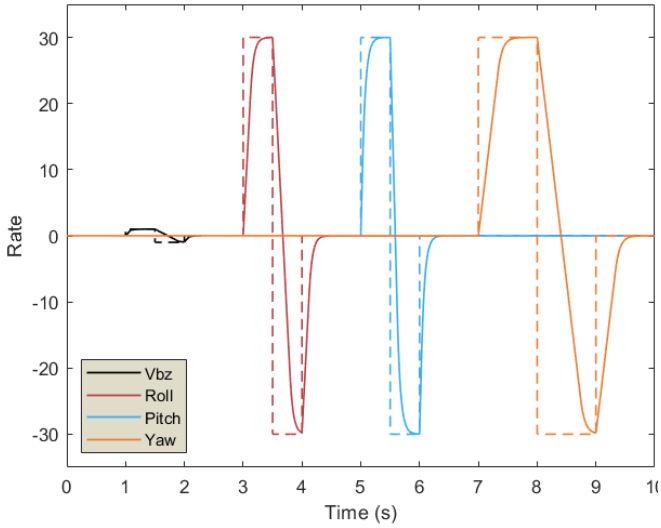


Figure 10: Multirotor mode rate controller step response

ii. Attitude Controller

The attitude controller receives desired attitudes as inputs and outputs rates to the rate controller. This only controls the roll and pitch, and is a proportional controller only; yaw is conventionally a rate controller to enable the aircraft to spin round indefinitely. The throttle is controlled only by the rate controller as it sets the vertical velocity:

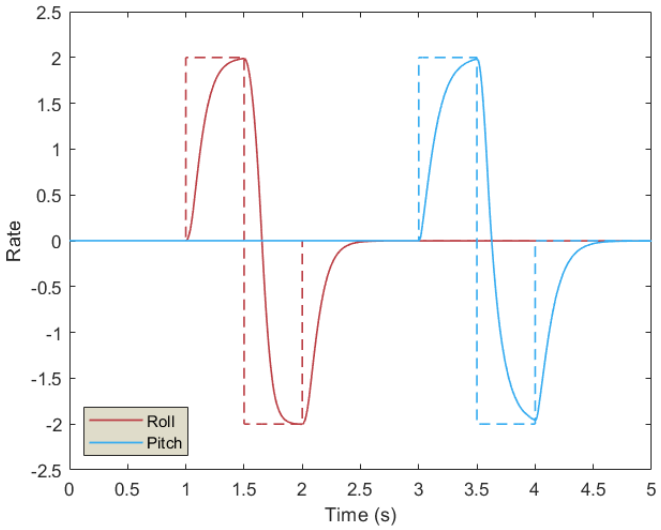


Figure 11: Multirotor mode attitude controller step response

The attitude is limited to 2 degrees as it has been found to be the most stable; it can be increased but, in some situations has been found to cause instability that causes the simulation to slow down and sometimes terminate.

iii. Velocity Controller

As the control inputs to the attitude controller are all very different (linear velocity, angular position, angular velocity) the aircraft is very hard to control in multirotor mode using a learning agent. A solution to this was to take out another layer of control by include a velocity controller for X and y. These control the attitude of the aircraft to make it move forwards, backward, left and right. With a further attitude controller to keep yaw at 0 degrees the learning agent has a much simpler task that makes tuning the learning parameters much easier:

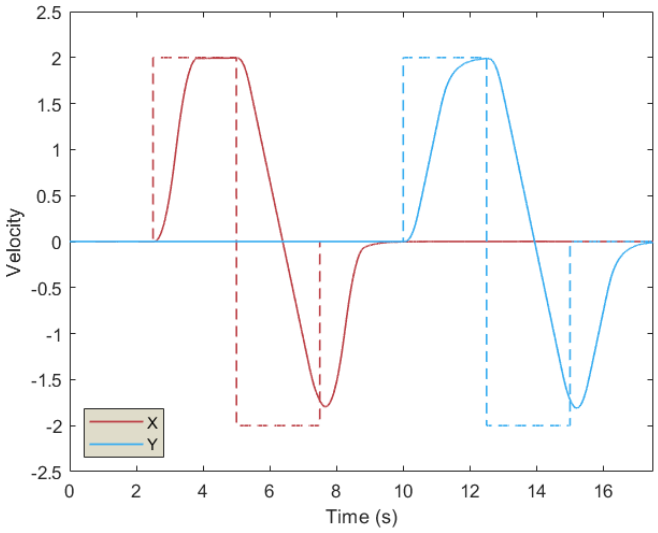


Figure 12: Multirotor mode velocity controller step response

4.2 Forward Flight Controller

The forward flight controller is simpler than the multirotor controller and only has two levels; one for rate and one for attitude. The rate controller outputs motor forces and elevon angles. It would be possible to use three-dimensional thrust vectoring and do away with elevons but this is unlikely to be the case in the final aircraft. Roll $\emptyset_r$, and pitch θ_r are controlled by elevons and yaw ψ_r is controlled by differential thrust on the rear motors. As before these are set by PID loops and then mixed together. The mixing rules are as follows:

Force 1	Force 2	Elevon Left	Elevon Right
V_x	V_x	-	-
-	-	$\emptyset_r$	$-\emptyset_r$
-	-	$-\theta_r$	$-\theta_r$
ψ_r	$-\psi_r$	-	-

Table 4: Forward flight mode rate controller mixing

The outputs of the controller are limited to avoid saturating the motors or breaking the elevons. For this controller one motor is capable of rotating backwards in a glide, but the forward flight controller has a hard-coded lower limit on speed of 10m/s which prevents this situation arising:

Control	V_x	ψ_r	Elevon L	Elevon R
Lower limit	0	-0.2	-20°	-20°
Upper limit	0.8	0.2	20°	20°

Table 5: Motor force and elevon upper and lower limits

i. Rate Controller

The rate controller was tuned to a step response with a maximum rate of $30°/s$ for $\emptyset_r, \theta_r, \psi_r$. The maximum velocity of the simulated aircraft is 50m/s, higher than the real aircraft (30m/s), and this would be corrected given more time. The controller input is $\in [10,50]$ to prevent the FF controller being used in multirotor mode. The proportional controllers are tuned to give the following step response:

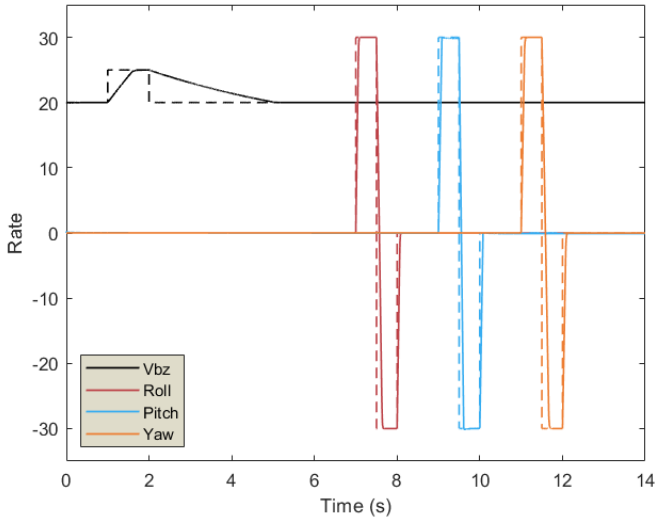


Figure 13: FF Rate controller step response

ii. Attitude Controller

The attitude controller is identical to its multirotor counterpart other than having higher limits ($\in [-30,30]$), it only controls pitch and roll. This has a step response as follows:

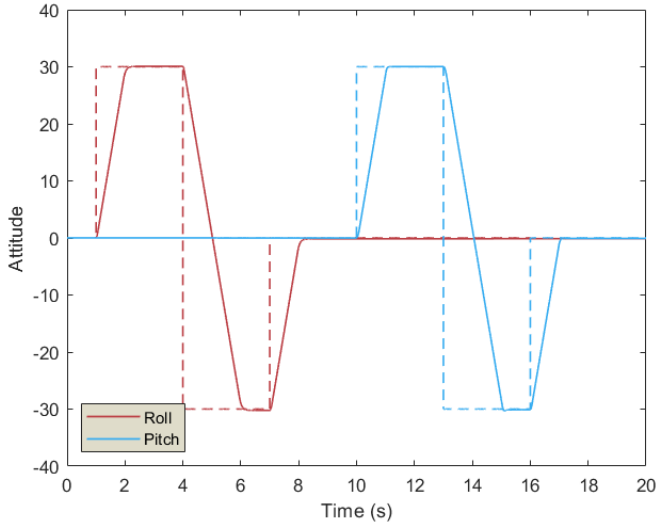


Figure 14: FF Attitude controller step response

Due to complexities in training the aircraft to control yaw, for many of the simulations yaw was fixed at 0. For this to work an attitude controller is needed for yaw for both multirotor and forward flight modes. These are simply copies of the pitch controller with the desired position fixed at 0.

With this working forward flight controller, the aircraft responds as expected to inputs. Changing the attitudes moves the aircraft as expected. The result is aircraft control inputs that cause it to fly to the desired location at speeds and rates comparable to a physical aircraft.

5. Reinforcement Learning

5.1 Algorithm Overview and Selection

The problem of control and how to map situations to actions fundamentally comes down to a Markov Decision Process (MDP) where the goal is to move from state to state in an optimal way. In the case of controlling a quadcopter the state represents its position in space and time, and the transition between states are the controller inputs. The problem is that the quality of each state is unknown and neither are the means of transitioning between them. To a human these are intuitive, pitching a multirotor forward will cause it to move forwards etc. Reinforcement learning agents have no a priori knowledge and instead the agent must interact with an environment to find how to optimise a numerical reward. This is both a blessing and a curse, since the algorithm may find novel optimisations, but equally is likely to be computationally intense. The agent must have a goal that relates to the environment: it should have three fundamental aspects at each time step (t); sensation, often called observation (x_t); action (a_t); and goal, often called reward or cost function (r_t) [33]. These observation-action pairs $s_t = (x_1, a_1, \cdots, a_{t-1}, x_1)$ are used to describe the state: the history of the observation may be required to fully describe the state.

The quality of a state is the sum of current and future reward $R_t = \sum_{i=t}^{T} \gamma^{(i-t)} r(s_i, a_i)$ with a discounting factor for future steps $\gamma \in [0,1]$. The return depends on future actions and hence the policy π which maps states to a learnt probability distribution of actions $\pi: \mathcal{S} \rightarrow \mathcal{P}(\mathcal{A})$. The action-value function describes the expected return of action a_t in state s_t [20]:

$$Q^{\pi}(s_t, a_t) = \mathbb{E}_{r_{i \geq t}, s_{i \geq t} \sim E, a_{i > t} \sim \pi}[R_t | s_t, a_t] \tag{5.1.1}$$

The Bellman Equation is key to many reinforcement learning algorithms: it is used to look ahead and map rewards in the future to actions now:

$$Q^{\pi}(s_t, a_t) = \mathbb{E}_{r_t, s_{t+1} \sim E}\left[r(s_t, a_t) + \gamma \mathbb{E}_{a_{t+1} \sim \pi}[Q^{\pi}(s_{t+1}, a_{t+1})]\right] \tag{5.1.2}$$

If the target policy is deterministic, as is the method used in this report, it is described by $\mu: \mathcal{S} \leftarrow \mathcal{A}$:

$$Q^{\mu}(s_t, a_t) = \mathbb{E}_{r_t, s_{t+1} \sim E}\left[r(s_t, a_t) + \gamma Q^{\mu}\left(s_{t+1}, \mu(s_{t+1})\right)\right] \tag{5.1.3}$$

Here the expectation is dependent only on the environment and can learn off-policy, i.e. it can learn the optimal policy independent of the agent's actions. On policy agents learn the value of the current policy and include the exploration steps [34].

The first algorithm considered for this problem was Q-Learning [35], in which a computational agent moves around in a discrete, finite world and chooses actions from a finite set at each time step. The world is a Markov process and the agent its controller. Initially there is a high degree of randomness in the agent's actions, but with time and experience the degree of randomness is reduced. A table of state-action transactions is updated according to the greedy policy $\mu(s) = argmax_a Q(s, a)$. Selecting the action with the highest *Q-Value* at each time step guarantees an optimal policy. The issue with this process is that it is finite in both space and action space. Controlling an aircraft could be discretised, however doing so would result in a huge 'Q-table' taking an extremely long time to train, or a very crude discrete action space giving correspondingly crude actions.

DeepMind famously overcame the problem of scalability in Q-Learning when one agent successfully played all original Atari games to an expert level after only observing the pixels from the game screen [36]. This method replaces the Q-Table with *deep Q networks* (DQN), convolutional neural networks

which approximate the Q function for a given state and action. Although extremely powerful this method still only works in discrete action space, in an aircraft these could be things such as 'increase roll', 'increase throttle' etc. Although potentially powerful this does not work in continuous state and action spaces.

A critical change which DeepMind made was to use a *replay buffer* which stores previous experiences. These are randomly sampled in training and used to update the networks and select actions, thus massively speeding up the training process. It can also potentially be modified either with a prioritisation function to select the best experiences [37], or by filling the experience buffer with pre-set experiences, giving the agent the possibility to learn from demonstration [38]. Both of these would be a potential performance increase for future work or on a similar project.

A good method of avoiding the issues of only using discrete action with DQN is to use *deterministic policy gradients (DPG)*, a method which is capable of operating with continuous actions. This is the algorithm which will be used throughout the rest of the paper. It works by using the expected gradient of the action-value function rather than attempting to find a global maximum at each time step. It is an off-policy actor critic algorithm which learns a target policy from exploratory behaviour. It has been proven to converge to the optimum controller in many cases [39] and convergence to a local optimum is guaranteed [40]. Recently this method been shown to outperform stochastic methods in high dimensional action spaces [41]. It can be challenging to apply due to the considerable knowledge of a system which the user must have in order to make policy decisions [40]. 'Deep' deterministic policy gradients choose actions from a stochastic policy which adjusts weights to tend towards an optimum solution, usually using some modified form of the Bellman Equation.

The deterministic policy gradient method is used to update an 'off policy actor-critic' topology. Two neural networks operate simultaneously, the actor network creates commands while the critic network judges them based on the effect they have in the action space in comparison to the target policy averaged over the state distribution of the behaviour policy [42, 43]. The actor chooses the next value based on the gradients of the critics output; this allows it to create continuous outputs.

In continuous action spaces greedy policy is difficult as it requires global maximisation at every time step. DPG instead moves the policy in the direction of the gradient of Q. The policy parameters θ^{k+1} are updated in proportion the gradient $\nabla_\theta Q^{\mu^k}\big(s, \mu_\theta(s)\big)$. As each state gives a different gradient, they are averaged over time with an expectation from the critic network of the state distribution [41]. The critic $Q(s,a)$ is updated according to the Bellman equation and the actor is updated by applying the chain rule to the expected return from the start distribution J with respect to other parameters [20, 41].

$$\begin{aligned}\nabla_{\theta\mu} J &\approx \mathbb{E}_{s_t\sim\rho^\beta}\left[\nabla_{\theta_\mu} Q(s,a|\theta^Q)|_{s=s_t,a=\mu(s_t|\theta^\mu)}\right] \\ &= \mathbb{E}_{s_t\sim\rho^\beta}\left[\nabla_a Q(s,a|\theta^Q)|_{s=s_t,a=\mu(s_t)} \nabla_{\theta_\mu}\mu(s|\theta^\mu)|_{s=s_t}\right]\end{aligned} \tag{5.1.4}$$

This deterministic policy gradient method is then combined with the modifications of function approximators, replay buffers etc. from deep Q networks to create the deep deterministic policy gradient algorithm DDPG [20] which is used in this project.

The replay buffer is updated continually in training with the oldest samples being discarded as new ones are created. Mini-batches are sampled uniformly from the replay buffer to update the network. These mini-batches are used to update both actor and critic networks: as the algorithm is off policy the replay buffer is large and therefore can contain a wide variety of uncorrelated experiences.

A common issue for all neural networks is that different components of the inputs are scaled very differently. To address this issue for this work the inputs are scaled to $\in [-1,1]$, but in the future implementing batch normalisation would be a useful change to make, where each mini-batch is normalised so that different experiences may be seen by the networks in a similar way [44].

To aid learning, noise is added to the actor policy; this is critical to ensure the agent explores the action space. The exploration can be treated independently of the learning, as it is an off-policy algorithm. The noise $\mathcal{N}$ is added to actor policy and based on the Ornstein-Uhlenbeck process; this is a common approach often used in literature [20, 45, 46]. In this process the noise is generated at a given time step and is correlated to previous noise. This causes the noise to tend to continue in one direction that is useful as an aid top exploration. $\mathcal{N}$ is chosen to suit the environment for which some experience is required. For this reason, the noise parameters were initially copied from another paper [47] that tackled a similar problem with some success.

The pseudo-code for the complete DDPG algorithm as described in [20] is:

Algorithm 1: Deep Deterministic Policy Gradients

Randomly initialise critic network $Q(s,a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ.
Initialise target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
Initialise replay buffer $\mathcal{R}$
for episode = 1, M **do**
    Initialise a random process $\mathcal{N}$ for action exploration
    Receive initial observation state s_1
    for t = 1, T **do**
        Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the policy and noise
        Execute action a_t and observe reward $r_t$and observe new state s_{t+1}
        Store transition (s_t, a_t, r_t, s_{t+1}) in $\mathcal{R}$
        Sample a random minibatch of $\mathcal{N}$ transitions (s_i, a_i, r_i, s_{i+1}) from $\mathcal{R}$
        Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
        Update critic by minimising the loss: $L = \frac{1}{N}\sum_i \left(y_i - Q(s_i, a_i|\theta^Q)\right)^2$
        Update the actor policy using the sampled policy gradient:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N}\sum_i \nabla_a Q(s,a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu}\mu(s|\theta^\mu)|s_i$$

        Update the target networks:

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$
$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$

    end for
end for

5.2 Initial Testing

The transition is the most complex part of the flight having a high number of inputs, a complex cost function, but more importantly having the ability to go unstable. The agent must control weights for the forward flight and multirotor controllers while balancing the tilt angle to give the most efficient transition possible. It is very easy for the agent to get these weights wrong, causing the simulation to become unstable and making it very difficult to train.

To avoid some of this complexity the first reinforcement learning experiments were done using the aircraft in multirotor mode only, with the angle of the rear motors fixed. This makes it possible to constrain the aircraft so that there are no combinations of control inputs which will cause it to become unstable.

For the first experiments the attitude controller was used with no rate controller and very limited maximum angles. The agent's observations were as follows:

Parameter	Unit	Max	Min	Quantity
Euler angles	°	180	−180	3
Euler angles	°/s	50	−50	3
X, Y, Z position	m	∞	−∞	3
X, Y, Z Velocity	m/s	∞	−∞	3

Table 6: Multirotor first test observer parameters

The code was set based on two MATLAB examples: a flying robot and the swinging and balance of a pendulum. These both use large convolutional neural networks which were modified for the correct inputs and outputs for both actor and critic:

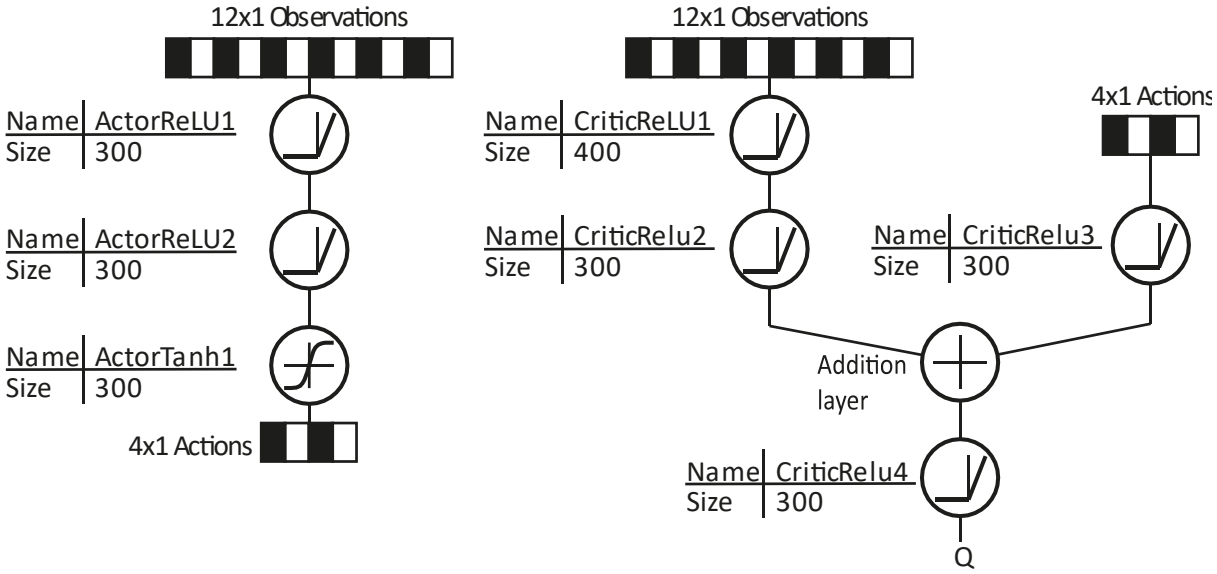


Figure 15: Multirotor first test actor and networks

These networks caused many issues, and were later found to be far too big for this use case. They slow down training and can 'fight' each other. They don't converge because the networks are large with many redundant perceptrons. This was not realised quickly: section 5.3 describes other means of optimising the simulation and training process that were explored.

5.3 Optimisations

i. Processing

The computer processor was found to make a huge difference to simulation speed. Three were trialled. A laptop using a 2.8GHz Intel i5-6200U was used first; with a fixed step size the simulation took approximately 15 minutes per episode for a 100s long simulation. This was slower than expected because of the laptop's tendency to thermal throttle and reduce clock speed under high load. Typical training sessions require hundreds or thousands of episodes to converge to a good solution so clearly optimisations were needed. An overclocked 4.5GHz Intel i5-4690k was used instead of the laptop; it was anticipated this would also thermal throttle but instead the episode time was reduced to approximately 5 minutes. Both of the CPUs were tested using single and multi-threading. Multi-threaded simulations run multiple agents in parallel which are used to update the experience buffer after a set number of samples. As both processors have 4 cores this means a potential 4 agents updating instead of one. This is faster than the single threaded simulation, but as the agents have to open and close threads when returning their data the performance increase is limited. Further, the computer cannot be used for any other purpose whilst it is running, a significant disadvantage for a process that runs for hours at a time. With a 4 core CPU this is not a price worth paying, but a massively parallel system with tens or hundreds of computing cores would make more sense. MATLAB's GPU accelerator was used with an NVidia GTX970 for updating the neural networks. It was anticipated that this would speed up the training dramatically; but with the sizes of neural networks being used the majority of computation time is due to the Simulink model and the GPU is never utilised above 3%.

ii. Sample Time

The simulations were accelerated greatly by reducing the sample rate. This is usually the same as the time step on the simulation, but was later made independent with variable step solvers. By doing this the sample rate was reduced from 0.001s for a fixed step solver to 0.4s with variable time stepping. 0.4s was chosen to match the sample rate from the MATLAB flying robot example [48]. The only downside found from using a variable step is that when the simulation becomes unstable the solver does not always recognise it, and can run extremely slowly until the end of the simulation.

iii. Flight Controllers

When using continuous PID controllers the integral gains can cause the simulation to slow down dramatically. The Simulink built-in PID controller did not work well for this application, and often gave unstable step responses or responses that were far too slow. Once again, the integral gains were the key cause of this problem, and were tuned to be $1E13$ when other gains were in the order of tens. After tuning these gains the step responses from Figure 10 to Figure 12 were obtained while using discrete PID controllers at 100Hz. This made the simulation more stable and much faster running than before.

iv. Reverse aerodynamic forces

It was found that the aircraft moving backwards is a big cause of instability; for example, when pitched back in multi rotor mode. The drag equation (2.2.2) contains a velocity squared term, so the force is always in the same direction. When the aircraft starts to move backwards the drag continues to push it backwards and it accelerates to infinity. To fix this there is a catch statement which sets the drag force to zero when the aircraft is moving backwards. The drag could have been made to slow the aircraft down but it wasn't deemed important as the focus is for an accelerating transition where the aircraft will be moving forwards.

5.4 Cost Functions

The cost function defines what the agent learns as good and bad behaviour; the goal of the cost function is to reward good actions and punish bad actions by giving positive and negative reward values for each. For the multirotor mode aircraft flying between points this can be as simple as the Euclidean distance to the target:

$$R_t = -\sqrt{\left(X_{goal} - X_t\right)^2 + \left(Y_{goal} - Y_t\right)^2 + \left(Z_{goal} - Z_t\right)^2} \tag{5.4.1}$$

Or it could be more complex, giving high rewards for being close to the goal, high negative rewards for being far away, but not differentiating between the path to the goal. In these reward functions the energy required to accomplish each action is often taken into account [48]:

$$R_t \sum_{i=1}^{n} r_i \tag{5.4.2.1}$$

$$r_1 = 10\left(\left({X_t}^2 + {Y_t}^2 + {Z_t}^2\right) < 0.5\right) \tag{5.4.2.2}$$

$$r_2 = -100\left(|X_t| \geq 20 \left|\right| |Y_t| \geq 20 \left|\right| |Z_t| \geq 20\right) \tag{5.4.2.3}$$

$$r_3 = -(0.2(R_{t-1} + E_{t-1})^2) \tag{5.4.2.4}$$

Throughout this project many cost functions were tested to try and promote the best behaviour. This had varied success as often the agent would behave in unexpected ways. A common approach for the agent when only negative rewards are given is to attempt to activate the simulation's termination function as quickly as possible. This will give a highly negative reward, so to improve the next time it will crash it faster! To stop this, rewards are given for simply surviving each time step: if these are too high then the agent learns very slowly, but if too small it reverts back to crashing as fast as possible.

For flying to a location, the most successful cost function used was the Eq. $5.4.1$ + a reward for staying alive. For the transition this had some success: the desired location was moved away at a desired speed for the aircraft to prevent backwards transition or unrealistic targets. The issue with this was that the agent learnt to optimise itself in multirotor mode to fly extremely quickly using the ducted fans for pitch control with the motors tilted back – not a full transition but nonetheless an excellent solution to the problem given. To rectify this energy was added to the reward function:

$$E = -0.02 \sum_{i}^{n} {F_i}^2 \tag{5.4.3}$$

This encourages the agent to fly in the most efficient way possible and minimise use of the ducted fans by flying in forward flight mode as soon as possible.

6. Multirotor Mode Testing

6.1 Setup

The forward flight controller was removed for testing in multi rotor only mode. This makes the simulation run faster and removes agent outputs. After applying all of the optimisations from 5.3. the agent was still unable to converge to a solution. Quadcopters and other multirotors have been trained to fly using DDPG before. 'Learning Control Policies for Quadcopter Navigation with Battery Constraints' [47] is a paper where a similar issue of flying a multirotor from A to B was solved; critically, all of the hyper-parameters are given in the paper. Most important to training success are the noise settings added to actor policy using the Ornstein-Uhlenbeck process; these give good exploration characteristics for the multirotor mode. As shown in Figure 17:

Variable	Symbol	Value
Mean	σ	0
Mean attraction constant	σ_{ac}	0.15
Variance	v	0.3
Decay Rate	v_{dr}	0.4

Table 7: Noise Parameters

$$\mathcal{N}_t = \mathcal{N}_{t-1} + \sigma_{ac}(\sigma - \mathcal{N}_{t-1})\delta_t + rand\big(\epsilon(-\sigma, \sigma)\big)v\sqrt{\delta_t}(1 - v_{dr})^{t-1}$$

The problem is simplified by constraining the aircraft's vertical velocity and yaw so that the agent only controls X and Y velocity. The goal of the agent was to fly to a point in two-dimensional space. Hovering at that point until the end of the simulation maximises the reward. As this is a simple problem, the critic network was shrunk to single layers with 20 perceptrons using ReLU as their activation function (150 perceptrons in [47]).

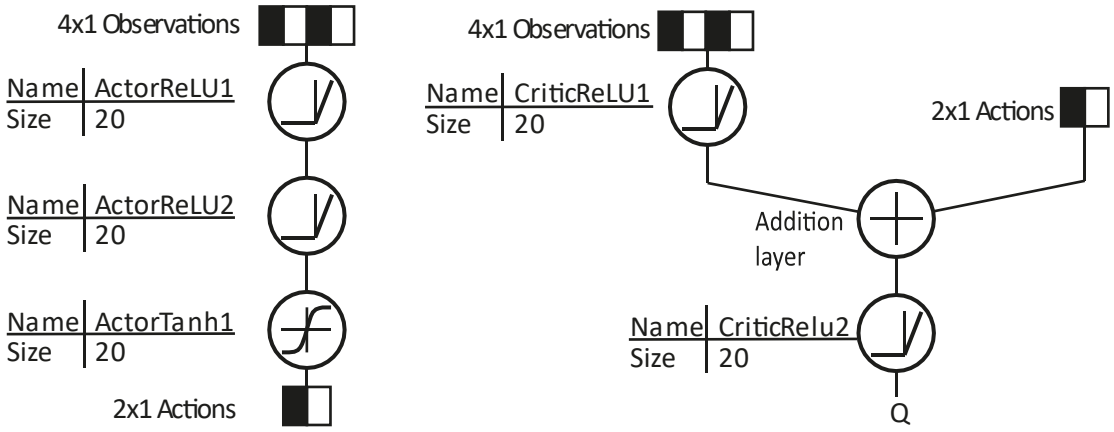


Figure 16: Multirotor 2D actor and critic networks

6.2 2D with Velocity Control

This was the first experiment to give good results, and the agent converged after approximately 500 episodes. Initially the desired location was fixed to make it more obvious how the training was going, but randomising it at the start of each episode gives the same result:

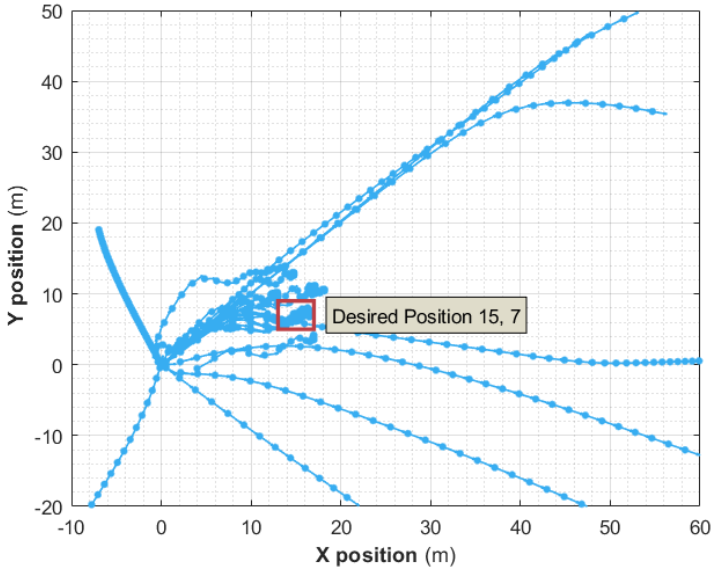


Figure 17: Exploration of aircraft in multirotor mode (2D)

The above plot shows the aircrafts trajectories through time. At the start of its training the aircraft has randomised neural networks and moves somewhat randomly as a result. Over time the networks are updated, but due to the noise added to the actor policy the aircraft will initially move in a random direction. Throughout training the agent converges to flying closer and closer to the desired goal. The plot shows the training over 500 episodes, sampled at intervals of 20, the agents that finish in the red box are the best agents from towards the end of training. The rewards at each episode are shown in Figure 18.

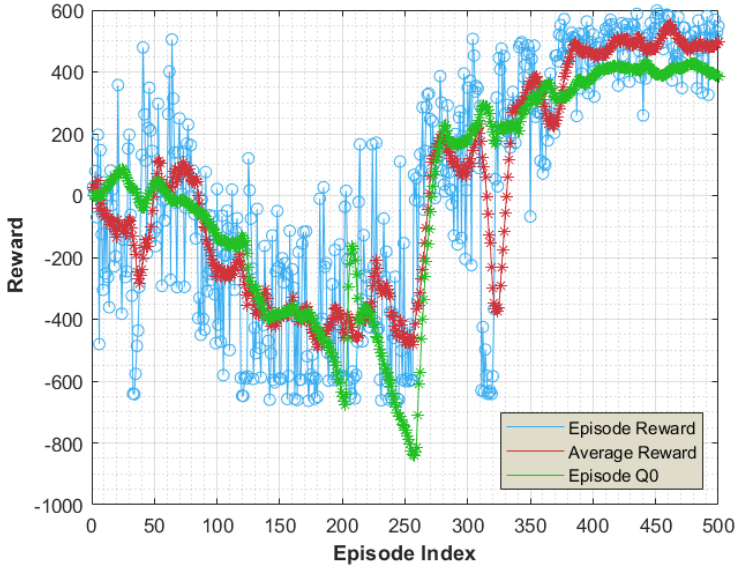


Figure 18: Episode reward of aircraft in multirotor mode (2D)

The episode reward is extremely noisy due to the Ornstein-Uhlenbeck noise. The average reward is a better indication of how effectively the agent is learning. Initially it is expected that the actor and critic will be very different, and will not immediately start to converge. The critic learns a close approximation of the reward function while the actor learns a good policy. This is shown by the Episode Q0 – the discounted long-term reward at the start of each episode. As the training progresses, this should approach the true discounted long-term reward. The average reward initially gets worse, but this is not a cause for concern as initially the replay buffer will be empty, and the agent will not have any good experiences on which to base training. After 500 episodes, the agent has found a reasonable solution; but comparison to a conventional PID controller on the position of the aircraft shows there is still considerable room for improvement:

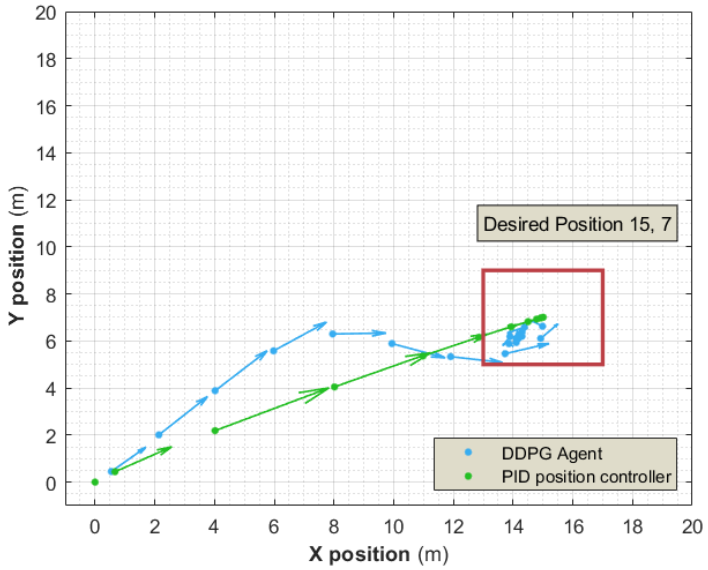


Figure 19: Comparison of PID and DDPG agent (2D)

To test whether the agent can improve from this point the agent is trained further, this time starting from the previously trained agent but looking more steps ahead (increased from 4 to 20), and with the maximum controller output increased from 2m/s to 5m/s. The agent is chasing smaller improvements in its reward and is expected to train more slowly, and for that reason the number of training episodes is increased from 500 to 1500. However, this was later found to be pointless. The increased controller output initially causes the agent to do far worse than before, reaching rewards as low as -5000. This quickly improves and it is noticeable how quickly the critic converges to the true reward. After 150 episodes the agent has improved to the point where it started the previous training cycle, and after 350 episodes (30 minutes training time) has matched the best reward from before. But beyond that there is no further improvement:

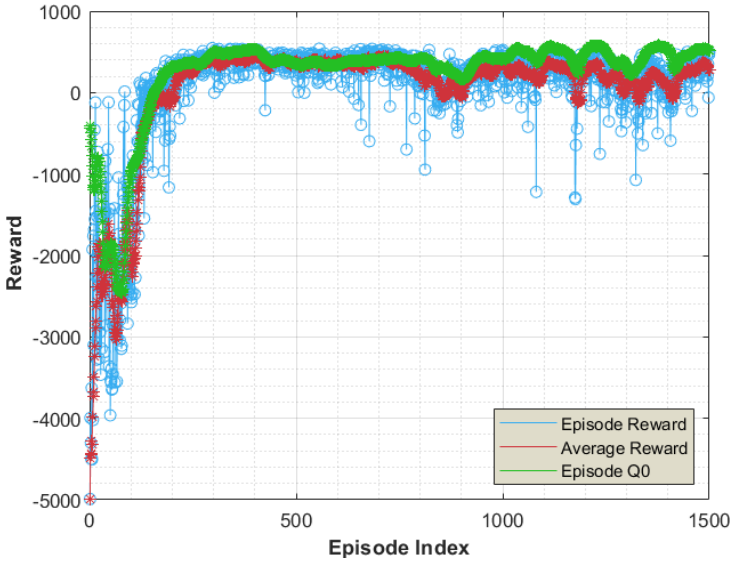


Figure 21: Multirotor training with look-ahead (2D)

Comparison to the PID controller shows that the aircraft now moves as quickly as the PID controller, but does not track the desired position any better. The agent travels more smoothly, and stops much more quickly when it gets to its final position. This could be improved by increasing the reward more quickly as it gets closer to the desired position but further tests in 2 dimensions would not be useful at this stage:

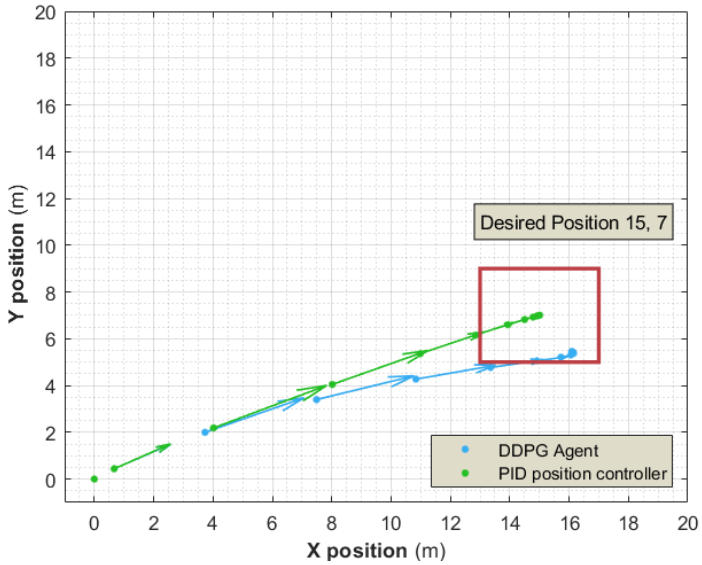


Figure 20: Comparison of PID and DDPG agent with look-ahead (2D)

6.3 3D with Attitude Control

The next simulation was to train the agent to control the aircraft in 3 dimensions so that it can fly to randomly selected points. Once again the agent is first trained with a low number of look-ahead steps: as the agent improves this number is increased. It was found that the actor was converging to a good solution while the critic was not. To improve this the noise settings were adjusted as follows:

Variable	Symbol	Value
Mean	σ	0
Mean attraction constant	σ_{ac}	0.6
Variance	v	0.15
Decay Rate	v_{dr}	0

Table 8: 3D multirotor testing noise parameters

These changes increase the amount of noise added to the policy, and increase exploration, thus helping to avoid local minima in exploration. Plotting the noise shows that it is far more constant and with a much higher magnitude:

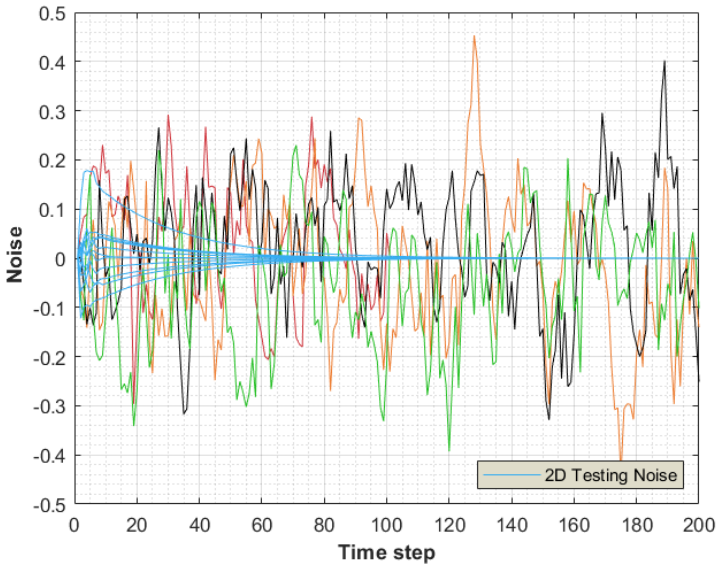


Figure 22: Policy noise comparison

When added to the actor policy this requires the agent to have much greater control outputs and therefore more exploration. This reduced the training time but unfortunately did not improve the critic as much as anticipated.

This agent has control outputs for the whole transition:

Output	Tri Min	Tri Max	FF Min	FF Max
Throttle	1.5	-1.5	25	-25
Roll	5	-5	25	-25
Pitch	10	-10	25	-25
Multirotor controller weight	0	1	0	1
Forward flight controller weight	0	1	0	1
Tilt angle	0	90	0	90

Figure 23: Transition controller agent outputs

yaw is fixed at zero as it was found to be extremely difficult to control. During the multirotor only tests the controller is fixed in multirotor mode to reduce the number of variables that can cause it to fail. In the future, when it is trained to transition, it will have control over these parameters again, whilst already having the capability to fly as a multirotor.

Choosing appropriate hyper parameters for the aircraft to fly without velocity controllers was much more challenging than in the previous simulation. Many different noise settings and network configurations were tried. The agent needs redundant neurons in its networks so that later on when it is trained for the transition it has some overhead which it can use in training.

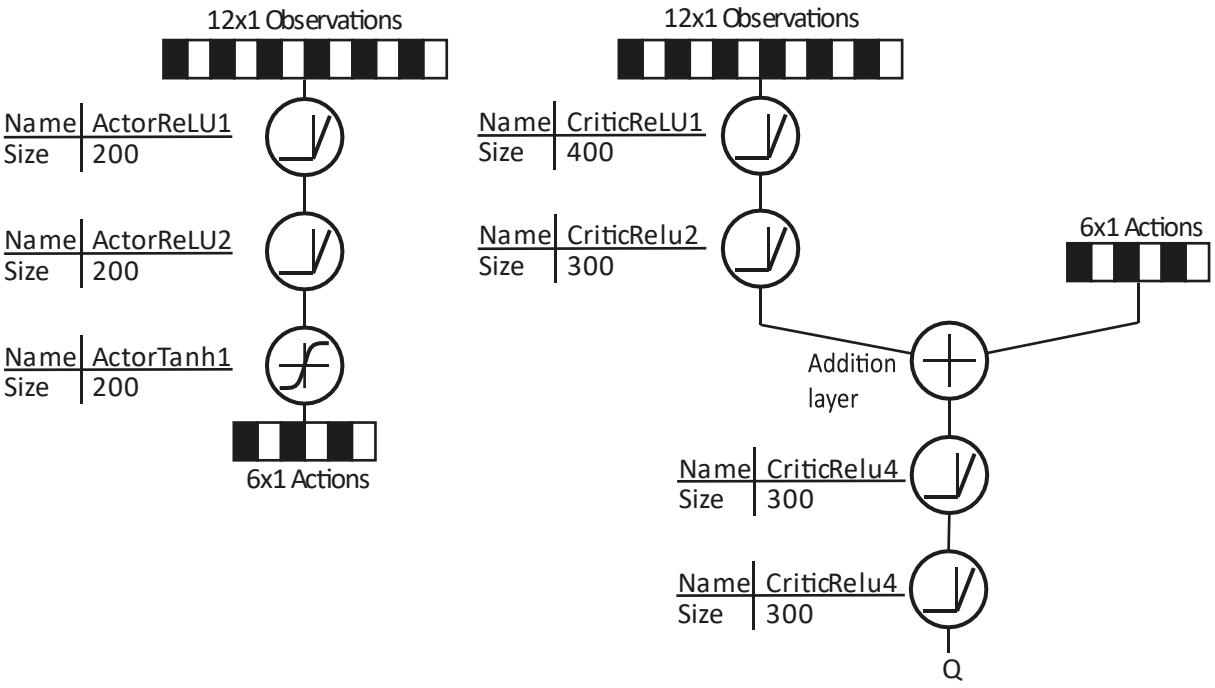


Figure 24: 3D Multirotor actor and critic networks

The aircraft is trained flying to a random position, maximising its reward by hovering there. Through the aircrafts training there is considerably more exploration than in the 2-dimensional case:

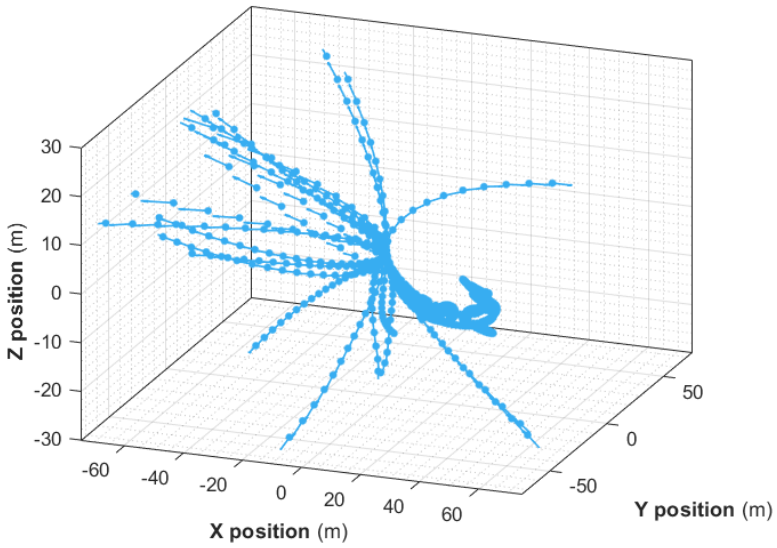


Figure 25: 3D multirotor exploration

The critic does not converge well, but the actors training is good nonetheless. This is unexpected as the actors training is based on the critic output. The increased noise should increase the variance in the critic output; although that has not worked it has improved the aircrafts training speed.

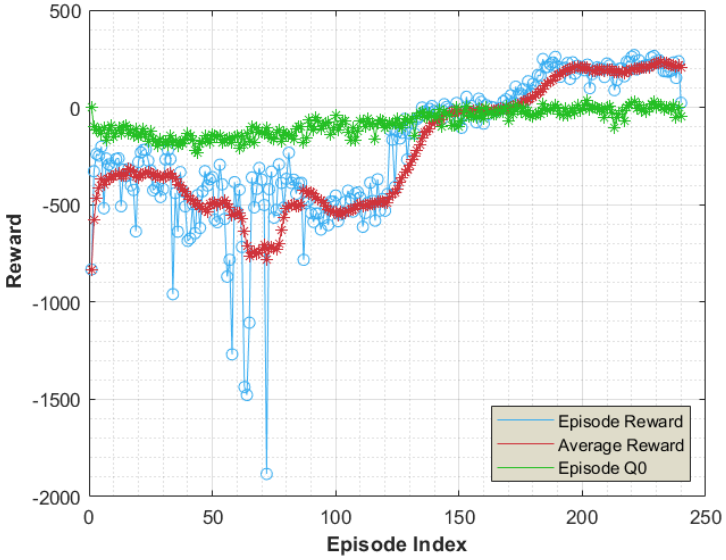


Figure 26: Training Stats 3D

By the end of the training the aircraft reliably flies to any point in 2D space and once again is less efficient than a PID controller. By further training and careful setting of the parameters it is possible that the agent could improve further, but there is little need and in practical terms the agent is accurate enough for anything other than extremely precise flying.

7. Transition Testing

7.1 ArduPilot Controller Evalutation

ArduPilot is the flight controller firmware used by 'Caelus'. The flight controllers in the simulation are designed to be as close a representation of ArduPilot as possible. Historically it has proven to be a very capable firmware and it has been used with success on all of Team Bath Drones' aircraft. For VTOL aircraft, ArduPilot is relatively untested. The transition is crude and is done by setting the tilt of the two rotors at a fixed angle until a pre-set airspeed is reached. After this the motors quickly go to horizontal and the aircraft behaves as a fixed wing. During the transition the only tuneable parameters are the tilt rate of the motors and the angle which they are at while the aircraft accelerates.

Parameter	Value	Unit
Q_TILT_RATE_UP	15	°/s
Q_TILT_MAx	45	°

Table 9: ArduPilot transition parameters

In the simulation below the transition is started after 10 seconds: before this the aircraft is in multirotor mode and is attempting to gain altitude. The transition then takes 8 seconds. The motors tilt to $45°$ over 3 seconds and during this time the forward velocity increases from 1 to 9m/s. The motors stay at 45° for a further 2 seconds, and the forward velocity is then 15m/s. As this is greater than the stall speed (approximately 12m/s) the rear motors quickly go to 90 degrees with a desired velocity of 25m/s reached 8 seconds after the start of the transition.

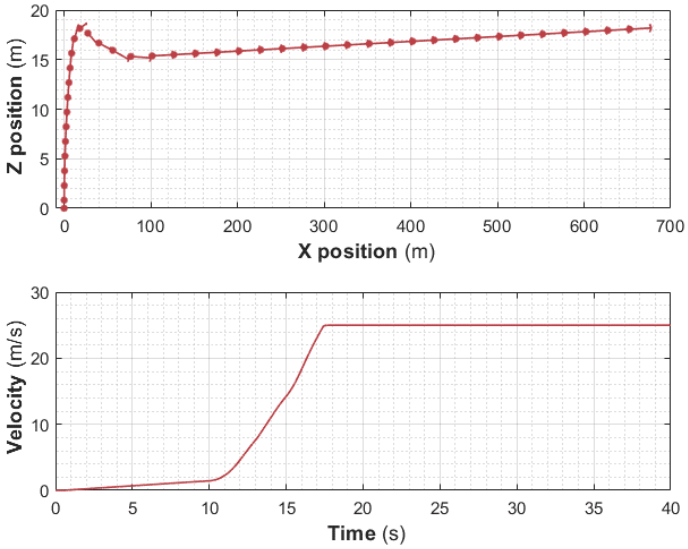


Figure 27: ArduPilot transition position and velocity

The only undesirable characteristic in this transition is the drop in altitude as the motors tilt. This is hard to avoid as the aircraft is underpowered due to battery constraints. The transition is quick however, and as soon as it has finished the aircraft pitches up to compensate.

The transition starts after 10 seconds by rotating the motors to 45 degrees; the controller weights always add up to 1 and are proportional to the tilt angle, so at 45 degrees tilt the controllers are weighted 50% multirotor and 50% aeroplane. The aircraft is level before the transition, but starts to pitch up to counteract the loss of lift from the front fans. When it is fully pitched up the aircraft is in forward flight mode with all of the lift coming from the wings and fuselage.

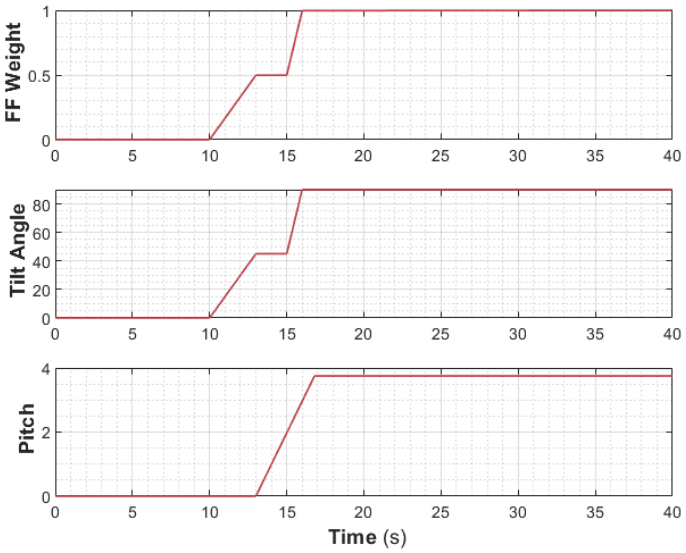


Figure 28: ArduPilot transition inputs

Overall this transition is excellent, the only negatives being that the parameters need some tuning to ensure it remains stable. In the real world these parameters would need to be tuned 'on the fly' to correct for wind or other anomalies which make the aircraft harder to control.

7.2 Reinforcement Learning Controller

i. Set Up

Initially it was intended that the multirotor mode controller would be adapted to control the transition by adding additional controller inputs and modifying the cost function. This was attempted, but following the success of the ArduPilot controller was not needed. The first attempt at controlling the transition was based on the trajectory in Figure 28. The agent used attempted to follow a trajectory which firstly flew up to 10 meters and then forwards at a rate of 25m/s. For the agent to do this it would have to transition at some point between the vertical and horizontal flight. The neural networks used are identical to Figure 24, with noise settings adjusted from before:

Variable	Symbol	Value
Mean	σ	0
Mean attraction constant	σ_{ac}	0.05
Variance	v	0.08
Decay Rate	v_{dr}	0.01

Table 10: Transition controller noise settings

Plotting the noise shows that it follows a stronger trend than before and is much further from Gaussian noise. Once again this promotes exploration and should be viewed as a progression of the understanding of the DDPG algorithm:

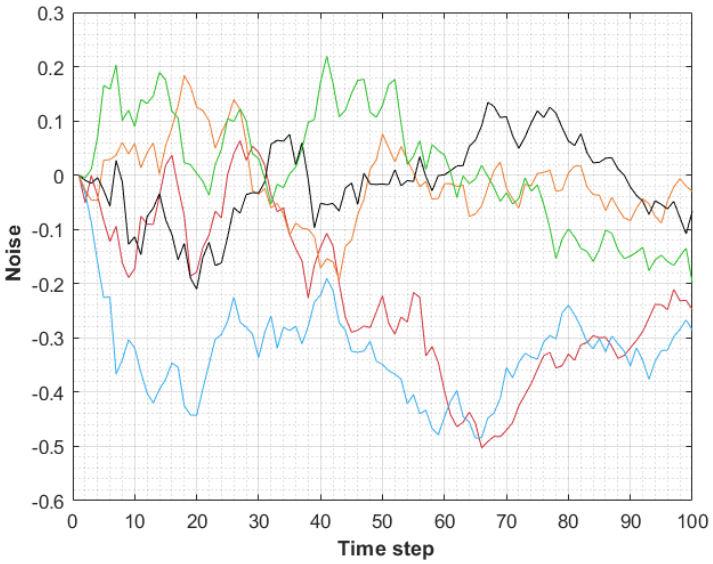


Figure 29: Transition controller noise plot

ii. Initial Results

The first transition attempt appears to converge to a good solution. The training was left to run over night through which time the agent got progressively better. The target network smoothing factor was reduced from 0.125 to 0.08; this reduces the speed of training but improves stability. The target networks are exact copies of the critic which are updated slowly over time. Reducing the rate at which they are updated reduces the probability of anomalies being picked up in them. The training continues for 4300 expisodes. Only the first 600 are shown as beyond this the agent progresses very slowly (and linearly).

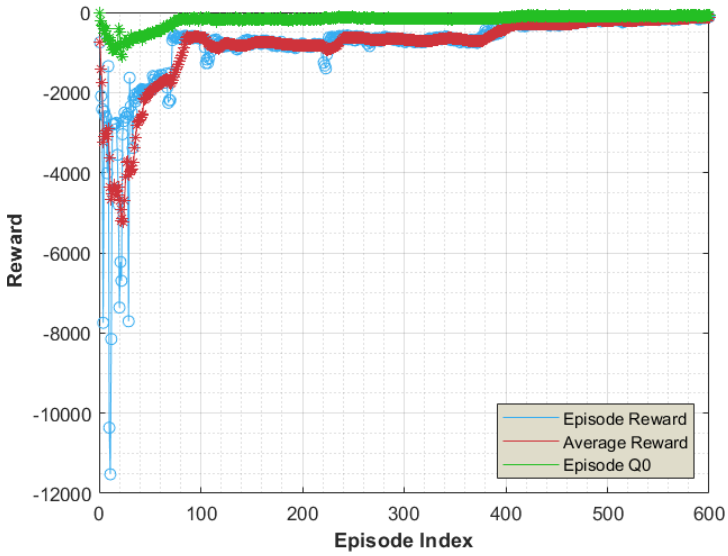


Figure 30: Transition controller training progress

The agent is attempting to fly the aircraft at 10m altitude in the positive X direction. The exploration for some of the agents is shown below. It is impossible to show all of the exploration as there are so many and the noise is random at each sample. The agent can be seen to fly in a few trajectories before converging to a solution which is optimised over time:

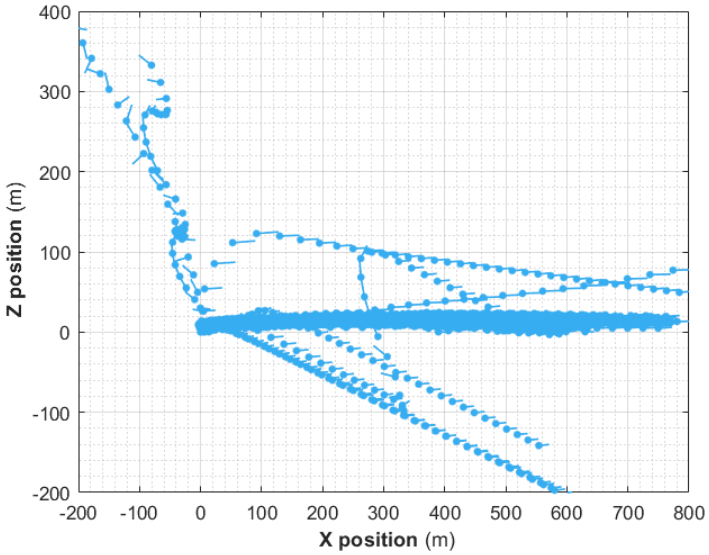


Figure 31: Transition controller exploration

The transition controller's final trajectory is shown below:

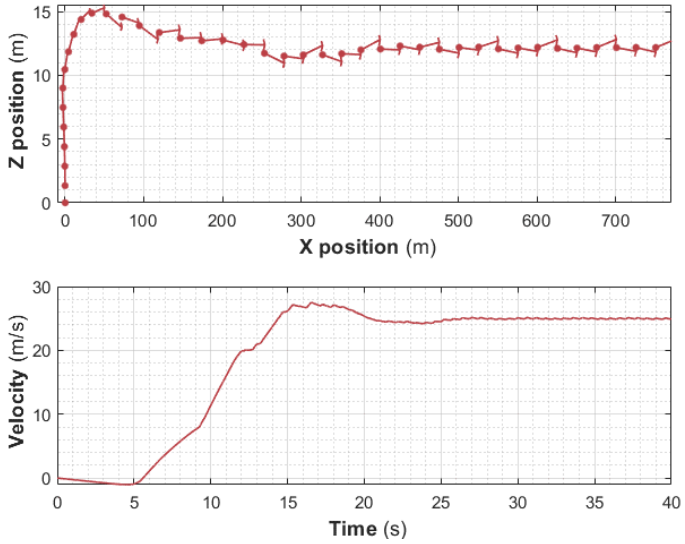


Figure 32: Transition controller trajectory

The trajectory and velocity plot are very similar to those of the ArduPilot transition, however on inspection of the controller inputs which give these results the transition has not worked as intended. Once again the forward flight controllers weight is $1 - Tri\_Weight$. It is evident that the agent has learnt to correlate tilt angle and controller weight, but instead of completing a transition it uses the ducted fans to assist with acceleration with negative pitch. It then uses a low angle of attack to balance the force from the front ducted fans and keep the aircraft level with the rear motors on full. This is an exploitation of a limitation of the flight dynamics model, which does not account for drag in the vertical direction:

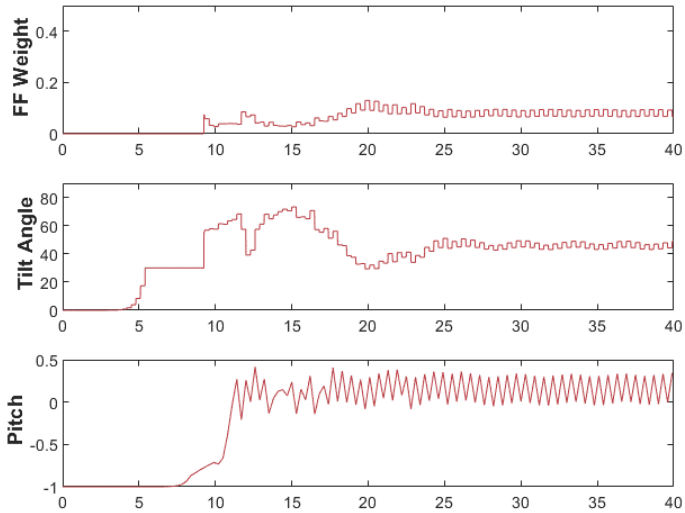


Figure 33: Transition controller inputs

Following this the aerodynamic coefficients are updated to include 360 degree rotation. The cost function is also modified to give negative rewards for actions which use a lot of energy and yaw instability – a common problem in the training of the previous agent.

$$R_t = 1 - 0.1\left(\sqrt{\left(X_{goal} - X_t\right)^2 + \left(Z_{goal} - Z_t\right)^2} + {F_1}^2 + {F_2}^2 + {F_3}^2\right) - 0.02|\theta|$$

iii. Modified Testing

The modified aerodynamics make the previous solution extremely inefficient, which is punished by the cost function. This gives rise to interesting behaviour and instead of flying the aircraft in multirotor mode it completes a full transition. With the same noise settings the training is reasonably fast and very stable:

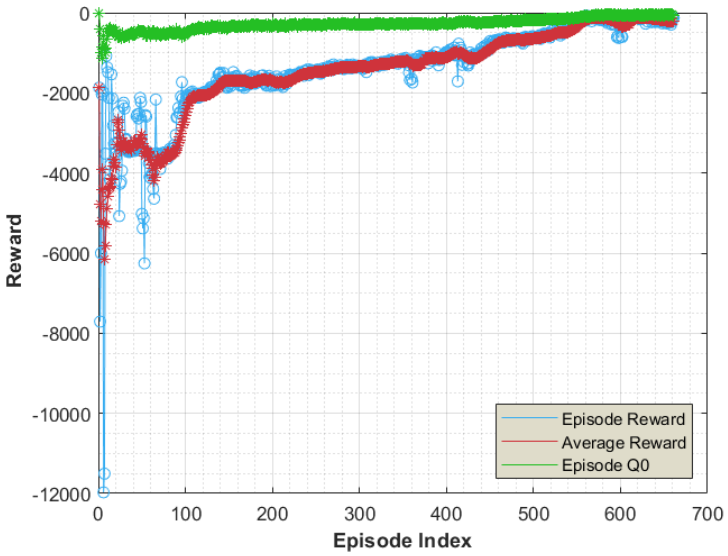


Figure 34:2nd Transition agent training progress

The exploration is good and the aircraft attempts flying at different pitch and aircraft modes:

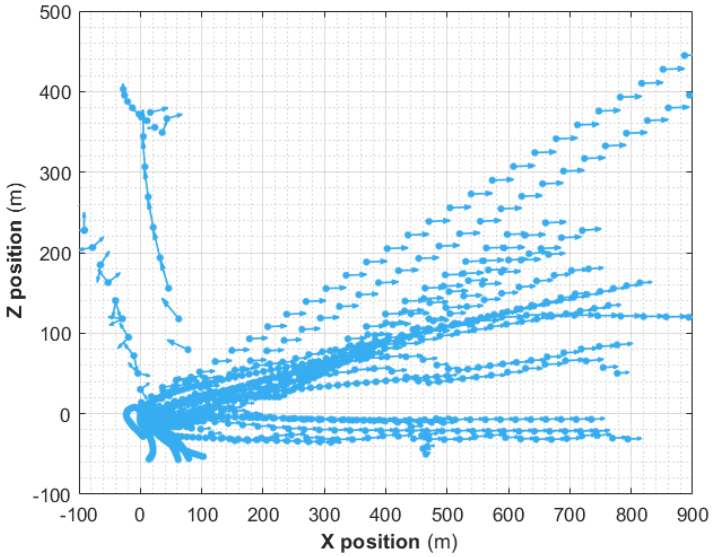


Figure 35: 2nd Transition agent exploration

The transition is very different from ArduPilots method, it is far more dynamic in its controller inputs. The desired trajectory requires much faster acceleration from the aircraft, to achieve this the agent uses gravity to speed up the transition. It flies up slightly in multirotor mode, then quickly changes into forward flight mode, using the motors to pitch the aircraft down and then accelerate with gravity. The aircraft accelerates quickly, rotating the motors to 65 degrees and correspondingly adjusts the controller weights to ensure stability.

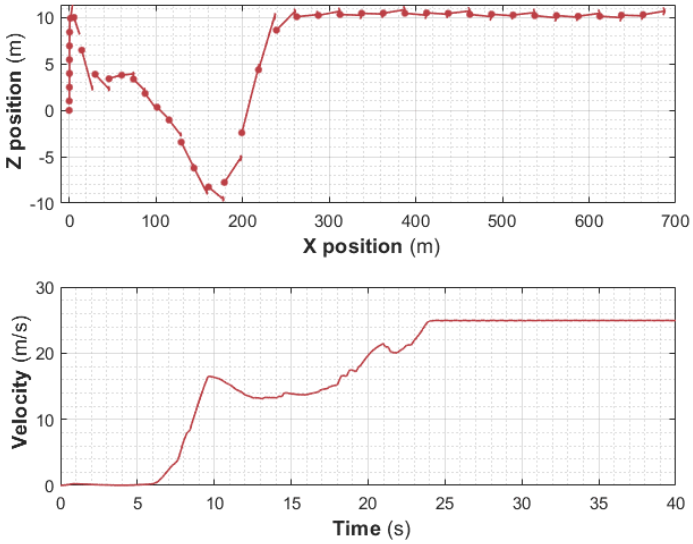


Figure 36: 2nd Transition agent trajectory

The outputs are stable, testament to the yaw term in the reward function. The motors are then quickly rotated back to 15 degrees, in multirotor mode the aircraft is accelerating and once again rotates the motors, quickly at first but more gradually as the aircraft becomes closer to forward flight mode. The motors are never tilted fully to 90 degrees, instead stopping at approximated 73 degrees. At this position they generate 96% of their maximum forward thrust and 30% of their maximum lift. This reduces the lift required from the aircraft and in turn the angle of attack and drag. This behaviour is exaggerated however, and the aircraft continues to gain altitude when it should remain level at 10m:

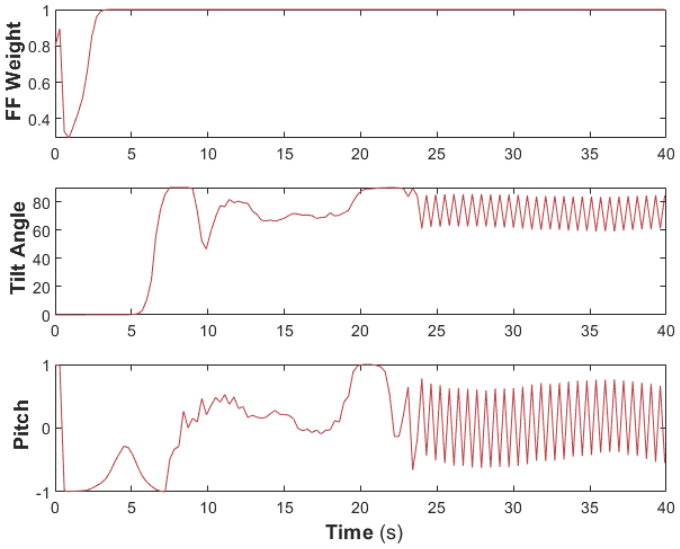


Figure 37: 2nd Transition agent outputs

8. Discussion

8.1 Relation to Original Workplan

Overall these results show that it is possible for the agent to complete a succesful transition, but it is not necessarily any better than one completed by a conventional controller. From the very beginning of this project there were challenges which meant that the intitial methodology and workplan could not be used [6]. It was thought that the simulation of the aircraft would be a modified flight dynamics model in the Gazebo simualtion environment. This would be controlled using software in the loop simulations of ArduPilot. The agent would be created in Python and would act as a control input for the ArduPIlot model which then controls the simulation. This was not possible as creating or modifying models in Gazebo is challenging and time consuming. In the time span of this project it was simply not possible. The Simulink model of the aircraft is an excellent alternative to this, and enables all aspects of the model to be modified and validated quickly, the only downside being that moving to a physical aircraft is much more challenging.

The workplan specified the need to develop an agent capable of flying between points, but did not envisage that this would be by far the most challenging part of the project. Once the aircraft was modelled the agent was created. Two weeks were allocated in the planning report for the first agent to fly the aircraft between points. After two weeks the agent was created, but it took a further three weeks to find settings which enabled it to converge to a solution. MATLAB 2019 was used throughout this project but was only released in late March, after the intelligent controller was supposed to have been created. Simply, the work plan under-estimated the problem of creating a flight dynamics model, identifying a suitable AI control algorithm, and then combining the two.

The most challenging part of the project was getting the agent to converge to a solution which reduces the cost function. The noise settings were usually the most challenging setting to get right, but towards the end of the project a setting was found which works well almost all of the time (Table 10). The cost function had a big influence on the way in which the aircraft transitioned. Weighting the energy usage too highly caused the aircraft to transition too quickly and use gravity to help with acceleration, requiring a considerable loss in altitude. If energy usage is not considered then the agent flies everywhere with the motors on full. Fundamentally this comes down to needing a better understanding of what is required from the transition.

8.2 Future Works

i. Model Accuracy

Wind was not modelled in these tests, and in the real world it is likely that wind would be a key factor at all times when the aircraft is not in forward flight mode. This is a key limitation of the project, and further investigation into the effects of wind would be very useful in future work. To accurately model the wind further CFD would be required and a more precise aerodynamic model would need to be used. It could be useful to use supervised learning to match a physical aircraft's dynamics to the model. This would give a more more accurate model which could be adapted to reinforcement learning. The project used a simulation that was never intended to be highly accurate, but this could be improved if the agent was ever going to be used to control a real aircraft. As it stands the aerodynamic model of the elevons needs to be improved with real world data, and the motors need their outputs based on real world thrust tests. Precise energy calculations for the reward function would be a worthwhile step.

ii. Agent Functionality

All these modifications would be useful if the agent was going to be modified for use on a real world aircraft, but if that were the case then much more functionality is required. The agent never successfully learned yaw control: in the real world this would be needed for almost every flight mode. Furthermore the agents were not trained to complete multiple tasks. The agent which completed the transition was not trained to fly between waypoints etc, selecting different flight modes and swapping between them 'on the fly' could ultimately replace the existing flight controller. As the technology develops these would have the potential for far more complex behaviour.

iii. Migration to a Physical Aircraft

If the agent was developed further and it was considered desirable as a replacement for a conventional controller then it would have to be recreated on a small scale computer which could fit inside an umanned aerial vehicle. By using a GPU or TPU accelerator to run neural networks quickly a Raspberry Pi sized computer would be capable of running the agent but might not be able to train 'on the fly'. Telemetry data from a real life flight controller is in the form of MAVlink messages, these can be modified into the observations which were used in the experiments in this report. The actor and critic networks can be copied with their weights to simplify the process of replicating the agent. Hardware in the loop simulations would serve as a good final stage of testing with a more accurate flight dynamics model before flight testing could take place. Migrating the system onto an aircraft is a project in itself and goes beyond what is likely possible in a final year project. Originally it was considered as a final stage to this project but difficulties in getting this far highlight how challenging this final step would be.

8.3 Future Technology

The learning agent never outperformed the conventional controllers but throughout the project it was apparent that it had the potential to do so. The problems which were tackled with reinforcement learning were easily solved with conventional controllers. Problems which require more planning and less predictable moves would be better suited to reinforcement learning. Navigation of complex airspace such as within buildings, prioritising waypoints, or controlling entire flights with battery constraints would all be good applications. The agent demonstrated the ability to look ahead, planning actions before it got to them such as slowing down on approach to a goal. Currently this is simple, and the temporal look ahead distance is a number of timesteps. Long term planning with deep LSTM networks has been demonstrated with reinforcement learning agents [49]. These are very recent, and the papers are yet to be published, but in the future the same technology could be applied to enable unmanned aircraft to make decisions and plan much further in advance than they do now. Applied to a similar projects this could enable the aircraft to make decisions well in advance and autonomous aircraft using this technology could outperform those that exist today.

9. Conclusion

This project has shown that reinforcement learning is a viable option for UAV control, but that its implementation is far more challenging than for conventional controllers. Noise parameters had by far the biggest effect on training agents and it took a very long time before they were understood properly. In later testing the key problems were to do with cost function but the agent converged to a solution either way. It was anticipated that that would be the case from a much earlier stage in the project. That said the tests showed good exploration, results as expected, and more forward planning than anticipated. The forward planning is perhaps key to unlocking the most potential from these techniques. Current literature is moving towards introducing deep LSTM networks for long term memory in reinforcement learning algorithms similar to this. Adding that to an aircraft would massively improve its capabilities as it would remove much of the need for careful pre-flight planning as it would be capable of updating 'on the fly'.

Key improvements for future work would be to improve the flight dynamics model and validate it with respect to a physical aircraft, and to add functionality to simulate entire flights. The biggest failing in the project was that yaw control was never successful: with further experimentation it should be possible in the future. It is extremely challenging however because it is a rate controller which requires very fine input, and thus may be negatively affected by noise which helps training in other areas.

Although the project did not manage to improve on the conventional controllers output it was still successful as it managed to complete a transition with no user input or prior experience and demonstrated examples of intelligent behaviour.

10. Bibliography

[1] Economist, “The Future of Agriculture,” *Technology Quarterly,* 2016.

[2] Economist, “Drone Technology Has Made Huge Strides,” *Technology Quarterly,* 2017.

[3] Economist, “TV Dinners,” *Science and Technology Quarterly,* 2016.

[4] J. Rupprecht, “Feds Make Major Moves To Relax Restrictions On Use Of Drones,” *Forbes,* 14 January 2019.

[5] Department for Transport, “Taking Flight: The Future of Drones in the UK Government Response,” 2019.

[6] M. Morris, “Project Planning and Background Review,” 2019.

[7] M. H. J. Amelink, v. P. Rene , M. Moulder and J. M. Flach, “Applying the Abstraction Hierarchy to the Aircraft Manual Control Task,” 2003.

[8] L. Zhong, Y. He, L. Yang and J. Han, “Control techniques of tilt rotor unmanned aerial vehicle systems: A review,” 2016.

[9] T. Dierks and S. Jagannathan, “Output Feedback Control of a Quadrotor UAV Using Neural Networks,” 2010.

[10] P. Casau, D. Cabecinhas and C. Silvestre, “Transition Control for a fixed-wing Vertical Take-Off and Landing Aircraft,” 2011.

[11] I. Guerrero, K. Londenberg, P. Gelhausen and A. Myklebust, “A Powered Lift Aerodynamic Analysis for the Design of Ducted Fan UAVs,” 2003.

[12] N. Johnson and M. A. Turbe, “Modeling, Control, and Flight Testing of a Small-Ducted Fan Aircraft,” 2006.

[13] R. Naldi, L. Marconi and A. Sala, “Modelling and control of a miniature ducted-fan in fast forward flight,” 2008.

[14] ArduPilot Dev Team, “Tilt Rotor Planes,” 2019. [Online]. Available: http://ardupilot.org/plane/docs/guide-tilt-rotor.html. [Accessed 02 08 2019].

[15] “NATOPS FLIGHT MANUAL NAVY MODEL AV--8B/TAV--8B 161573 AND UP 161573 AND UP,” 2007, p. 11:27.

[16] A. C. Miller and K. Sack, “Harrier Crash Renews Calls for an Inquiry,” Los Angeles Times, 2003.

[17] D. McNally, “Full-Envelope Aerodynamic Modeling of the Harrier Aircraft,” NASA Technical Memorandum 883676, 1986.

[18] W. Xinhua and C. Lilong, “Mathematical modeling and control of a tilt-rotor aircraf,” Aerospace Science and Technology, 2015.

[19] H. Baomar and P. J. Bentley, “An Intelligent Autopilot System that learns piloting skills from human pilots by imitation,” International Conference on Unmanned Aircraft Systems (ICUAS), 2016.

[20] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver and D. Wierstra, “Continuous control with deep reinforcement learning,” *CoRR,* vol. abs/1509.02971, 2016.

[21] T. Nageli, J. Alonso-Mora, A. Domahidi, D. Rus and O. Hilliges, “Real-time Motion Planning for Aerial Videography with Dynamic Obstacle Avoidance and Viewpoint Optimization,” IEEE ROBOTICS AND AUTOMATION LETTERS 1696–1703, 2017.

[22] J. Dentler, S. Kannan, M. Angel Olivares Mendez and H. Voos, “A tracking error control approach for model predictive position control of a quadrotor with time varying reference,” IEEE International Conference on Robotics and Biomimetics (ROBIO), 2016.

[23] A. Mahé, C. Pradalier and M. Geist, “Trajectory-control using deep system identification and model predictive control for drone control under uncertain load,” in *2018 22nd International Conference on System Theory, Control and Computing (ICSTCC)*, Sinaia, 2018.

[24] H. Baomar and P. J. Bentley, “Autonomous Navigation and Landing of Airliners Using Artificial Neural Networks and Learning by Imitation,” IEEE Symposium Series on Computational Intelligence (SSCI), 2017.

[25] C. Tomlin and M. Oishi, “Switched nonlinear control of a vstol aircraft,” in *Proceedings of the 38th IEEE Conference on Decision and Control (Cat. No.99CH36304)*, Phoenix, 1999.

[26] R. Naldi and L. Marconi, “Minimum time trajectories for a class of V/STOL aircrafts,” in *2010 Chinese Control and Decision Conference*, Xuzhou, 2010.

[27] H. Baomer and P. J. Bentley, “Autonomous landing and go-around of airliners under severe weather conditions using Artificial Neural Networks,” Workshop on Research, Education and Development of Unmanned Aerial Systems (RED-UAS), 2017.

[28] L. Duanzhang, C. Peng and C. Nong, “A generic trajectory prediction and smoothing algorithm for flight management system,” IEEE Chinese Guidance, Navigation and Control Conference (CGNCC), 2016.

[29] L. Buşoniu, de Bruin, Tim, D. Tolić, J. Kober and I. Palunko, “Reinforcement learning for control: Performance, stability, and deep approximators,” *Annual Reviews in Control,* vol. 46, pp. 8-28, 2018.

[30] M. Sheppard, A. Oswald, C. Valenzuela, G. Sullivan and R. Sotudeh, “Reinforcement Learning in Control,” in *9th Int. Conf. on Mathematical and Computer Modeling*, Berkeley, 1993.

[31] B. L. Stevens and F. L. Lewis, Aircraft Control and Simulation, New York: John Wiley & Sons, 1992.

[32] S. Naphade, “VTOL_Y4_Tiltrotor_UAV_MATLAB_Model,” 7 January 2018. [Online]. Available: https://github.com/SwapUNaph/VTOL_Y4_Tiltrotor_UAV_MATLAB_Model. [Accessed 2019].

[33] R. S. Sutton and A. G. Barto, Reinforcement Learning : An Introduction, Massachusetts: MIT Press, 1998.

[34] S. Gu, E. Holly, T. Lillicrap and S. Levine, “Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates,” in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, Singapore, 2017.

[35] P. Dayan and C. J. Watkins, “Technical Note: Q-Learning,” *machine Learning,* vol. 8, no. 3-4, pp. 279-292, `992.

[36] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra and M. A. Riedmiller, “Playing Atari with Deep Reinforcement Learning,” *CoRR,* vol. abs/1312.5602, 2013.

[37] Y. Hou, L. Liu, Q. Wei, X. Xu and C. Chen, “A novel DDPG method with prioritized experience replay,” in *2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, Banff, 2017.

[38] T. Hester, M. Vecerik, O. Pietquin, M. Lanctot, T. Schaul, B. Piot, D. Horgan, J. Quan, A. Sendonaris, I. Osband, G. Dulac-Arnold, J. Agapiou, J. Z. Leibo and A. Gruslys, “Deep Q-Learning from Demonstrations,” in *The Thirty-Second AAAI Conference on Artificial Intelligence (AAAI-18)*, New Orleans, 2018.

[39] S. Bradtke, “Reinforcement Learning Applied to Linear Quadratic Regulation,” in *In Advances in Neural Information Processing Systems 5*, Morgan Kaufmann, 1993, pp. 295-302.

[40] J. Peters, “Policy Gradient Methods,” *Scholarpedia,* vol. 5, p. 3698, 2010.

[41] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra and M. Riedmiller, “Deterministic Policy Gradient Algorithms,” in *31st International Conference on Machine Learning (ICML 2014)*, 2014.

[42] T. Degris, P. Pilarski and R. Sutton, “Model-free reinforcement learning with continuous action in practice,” in *American Control Conference*, Montreal, 2012.

[43] S. Bhatnagar, R. Sutton, M. Ghavamzadeh and M. Lee, “Incremental Natural Actor-Critic Algorithms,” in *Advances in Neural Information Processing Systems 20*, Vancouver, 2007.

[44] Ioffe, Sergey; Szegedy;, “Batch Normalization: Accelerating Deep Network Training b,” in *Proceedings of the 32nd International Conference on Machine Learning*, Lille, 2015.

[45] M. Plappert , R. Houthooft, P. Dhariwal, S. Sidor, R. Y. Chen, C. Xi, T. Asfour, P. Abbeel and M. Andrychowicz, “Parameter Space Noise for Exploration,” in *International Conference on Learning Representations*, 2018.

[46] J. Dy and A. Krause, “Decoupling Exploration and Exploitation in Deep Reinforcement Learning Algorithms,” in *Proceedings of the 35th International Conference on Machine Learning*, Stockholm, 2018.

[47] R. Murthy, H. Ruresh and C. Song, “Learning Control Policies for quadcopter Navigation with Battery Constraints”.

[48] MATLAB, “Train DDPG Agent to Control Flying Robot,” [Online]. Available: https://uk.mathworks.com/help/reinforcement-learning/ug/train-agent-to-control-flying-robot.html. [Accessed 2019 April 16].

[49] O. Vinyals et al, *AlphaStar: Mastering the Real-Time Strategy Game StarCraft II,* https://deepmind.com/blog/alphastar-mastering-real-time-strategy-game-starcraft-ii, 2019.

[50] ArduPilot Dev Team, “Copter Attitude Control,” [Online]. Available: http://ardupilot.org/dev/docs/apmcopter-programming-attitude-control-2.html. [Accessed 28 4 2019].

11. Appendices

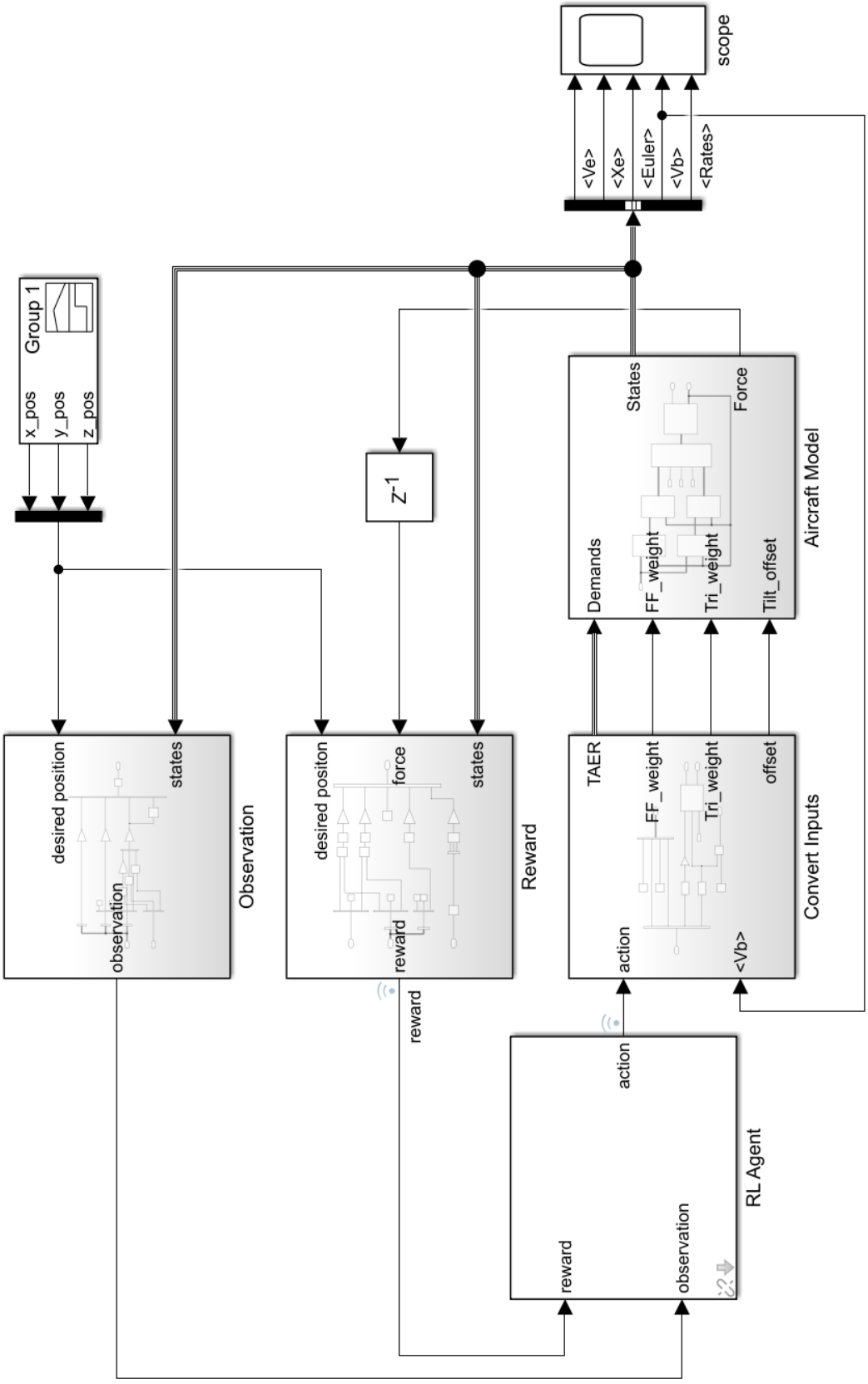
11.1 Training Code

```
mdl = 'Transition';
open_system(mdl)

% Specify observer information
numObs = 10;
obsInfo = rlNumericSpec([numObs 1],'LowerLimit',-1,'UpperLimit', 1);
obsInfo.Name = 'Observer';

% Specify actor information
numAct = 4;
actInfo = rlNumericSpec([numAct 1],'LowerLimit',-1,'UpperLimit', 1);
actInfo.Name = 'actor';

% Initialise environment - simulator, agent, observer information, actor
% information
env = rlSimulinkEnv('Transition','Transition/RL Agent',...
    obsInfo,actInfo);

% Set model params
Ts = 0.2;           % sample time
Tf = 40;            % finish time
rng('shuffle');     % Set rng seed

env.ResetFcn = @(in)setVariable(in,'Desired_Location',40*rand(3,1)-
20,'Workspace',mdl);

% Create some convolutional neural nets for the critic
% specify the number of outputs for the hidden layers.

% Create critic network
hiddenLayerSize = 200;
statePath = [
    imageInputLayer([numObs 1 1], 'Normalization', 'none', 'Name',
'observation')
    fullyConnectedLayer(hiddenLayerSize, 'Name', 'CriticStateFC1')
    reluLayer('Name', 'CriticRelu1')
    fullyConnectedLayer(hiddenLayerSize, 'Name', 'CriticStateFC2')
    reluLayer('Name', 'CriticRelu2')
    fullyConnectedLayer(hiddenLayerSize, 'Name', 'CriticStateFC3')];
actionPath = [
    imageInputLayer([numAct 1 1], 'Normalization', 'none', 'Name',
'action')
    fullyConnectedLayer(hiddenLayerSize, 'Name', 'CriticActionFC1',
'BiasLearnRateFactor', 0)];
commonPath = [
    additionLayer(2,'Name', 'add')
    reluLayer('Name','CriticCommonRelu1')
    fullyConnectedLayer(hiddenLayerSize, 'Name', 'CriticCommonFC1')
    reluLayer('Name', 'CriticCommonRelu2')
    fullyConnectedLayer(1, 'Name','CriticOutput')];

criticNetwork = layerGraph();
criticNetwork = addLayers(criticNetwork,statePath);
criticNetwork = addLayers(criticNetwork,actionPath);
criticNetwork = addLayers(criticNetwork,commonPath);
criticNetwork = connectLayers(criticNetwork,'CriticStateFC3','add/in1');
```

```
criticNetwork = connectLayers(criticNetwork,'CriticActionFC1','add/in2');
%specify critic options
criticOptions = rlRepresentationOptions('LearnRate',1e-
03,'GradientThreshold',2,'UseDevice','gpu');
critic =
rlRepresentation(criticNetwork,obsInfo,actInfo,'Observation',{'observation'
},'Action',{'action'},criticOptions);

% Create the actor network (another CNN)
actorNetwork = [
    imageInputLayer([numObs 1
1],'Normalization','none','Name','observation')
    fullyConnectedLayer(hiddenLayerSize,'Name','fc1')
    reluLayer('Name','relu1')
    fullyConnectedLayer(hiddenLayerSize,'Name','fc2')   %oscillates a bit
without these bois
    reluLayer('Name','relu2')                           %oscillates a bit
without these bois
    fullyConnectedLayer(numAct,'Name','fc3')
    tanhLayer('Name','tanh1')];
%specify actor options
actorOptions = rlRepresentationOptions('LearnRate',1e-
04,'GradientThreshold',2,'UseDevice','gpu');
actor =
rlRepresentation(actorNetwork,obsInfo,actInfo,'Observation',{'observation'}
,'Action',{'tanh1'},actorOptions);

% Create the DDPG agent
agentOpts = rlDDPGAgentOptions(...
    'SampleTime',Ts  ,...
    'TargetSmoothFactor',0.125,...
    'ExperienceBufferLength',1e6 ,...
    'DiscountFactor',0.99,...
    'MiniBatchSize',32);

agentOpts.NumStepsToLookAhead = 10;

%specify policy noise
agentOpts.NoiseOptions.InitialAction = 0;
agentOpts.NoiseOptions.Mean = 0;
agentOpts.NoiseOptions.Variance = 0.08;
agentOpts.NoiseOptions.VarianceDecayRate = 0.01;
agentOpts.NoiseOptions.MeanAttractionConstant = 0.05;

agent = rlDDPGAgent(actor,critic,agentOpts);

%specify training parameters
maxepisodes = 2000;
maxsteps = ceil(Tf/Ts);
trainOpts = rlTrainingOptions(...
    'MaxEpisodes',maxepisodes, ...
    'MaxStepsPerEpisode',maxsteps, ...
    'ScoreAveragingWindowLength',15, ...
    'Verbose', false, ...
    'Plots','training-progress',...
    'StopOnError', 'off',...
    'StopTrainingCriteria','AverageReward',...
    'SaveAgentCriteria',"EpisodeCount",...
    'SaveAgentValue',1,...
    'StopTrainingValue',8000);
```

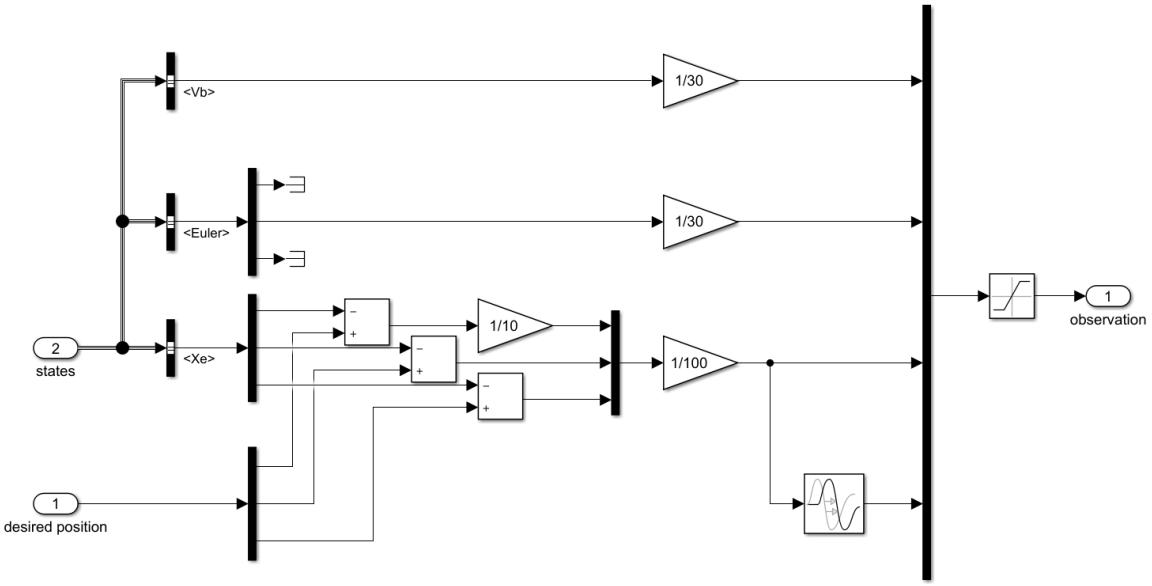
```
% do training or load agent
doTraining = True;

if doTraining
    % Train the agent.
    trainingStats = train(agent,env,trainOpts);
else
    % Load pretrained agent.
    agent = load('Agent2000.mat');
    agent = agent.saved_agent;

end
%run simulation without training noise
experience = sim(env,agent);
```

11.2 Simulink Model



i. Observations

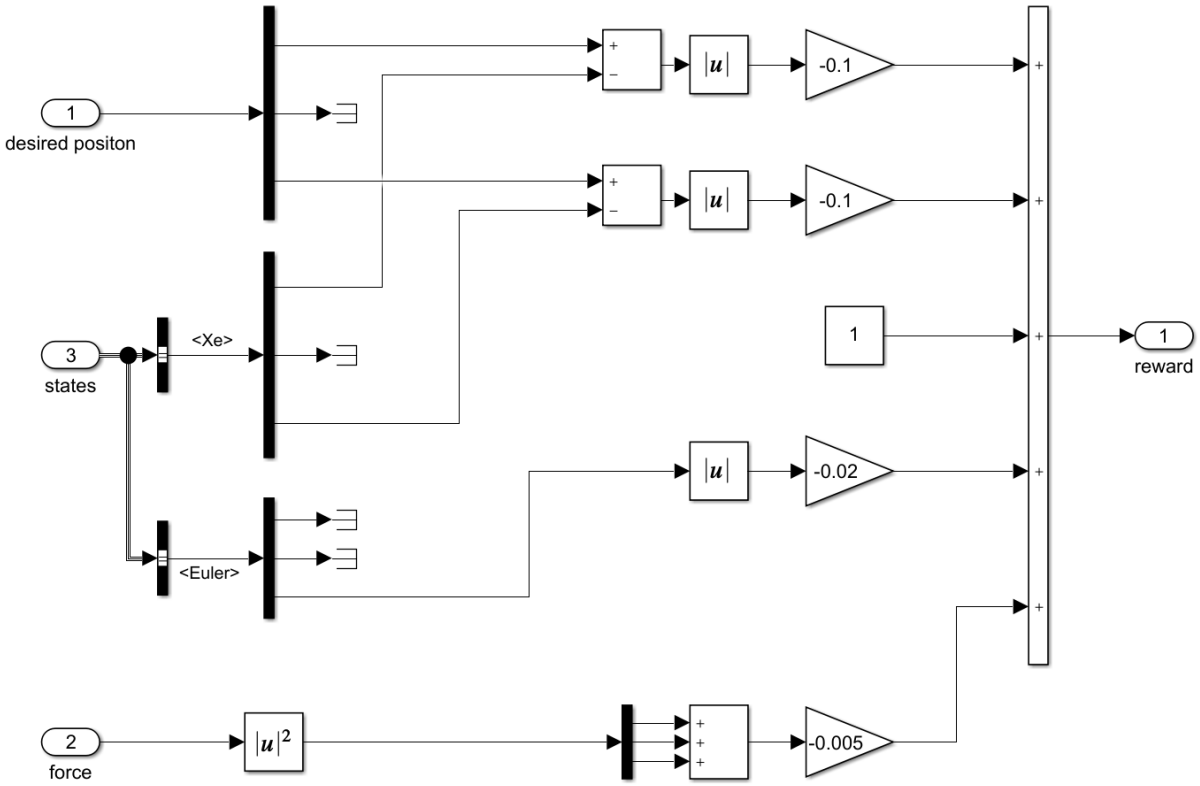
This takes the states and scales them so that they are in the range $\in [-1,1]$. The delay is equal to the time step and the saturation is applies limits on the outouts



ii. Reward

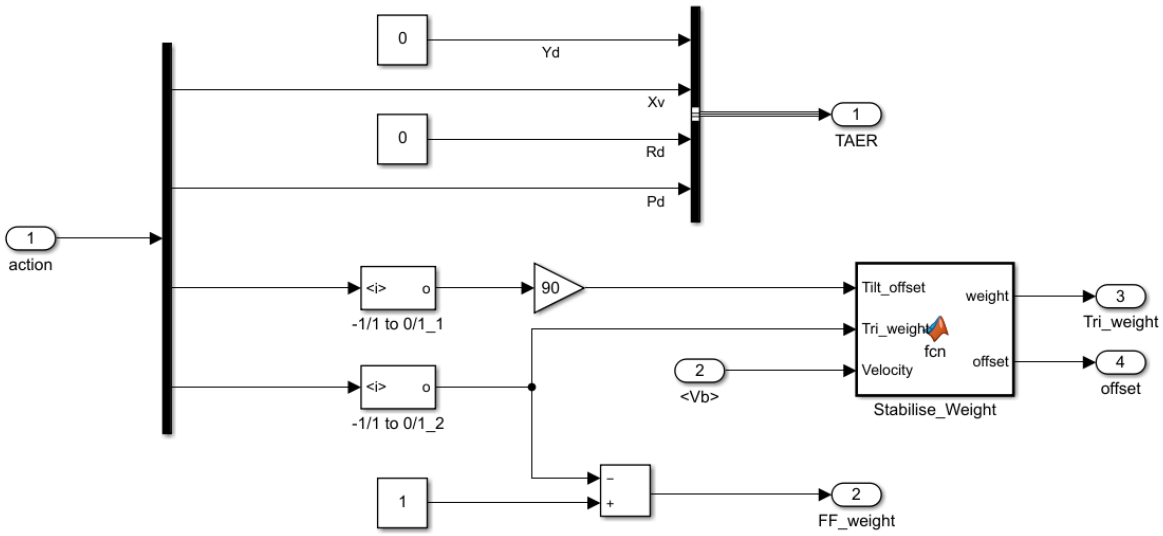
Reward function for the transitioning agent. Reward is:

$$Error_X + Error_y + Survival + Yaw + Energy$$



iii. Convert Inputs

This block converts the agents multiplexed output into one useful for the flight dynamics model. The yaw and roll are fixed for the transition.

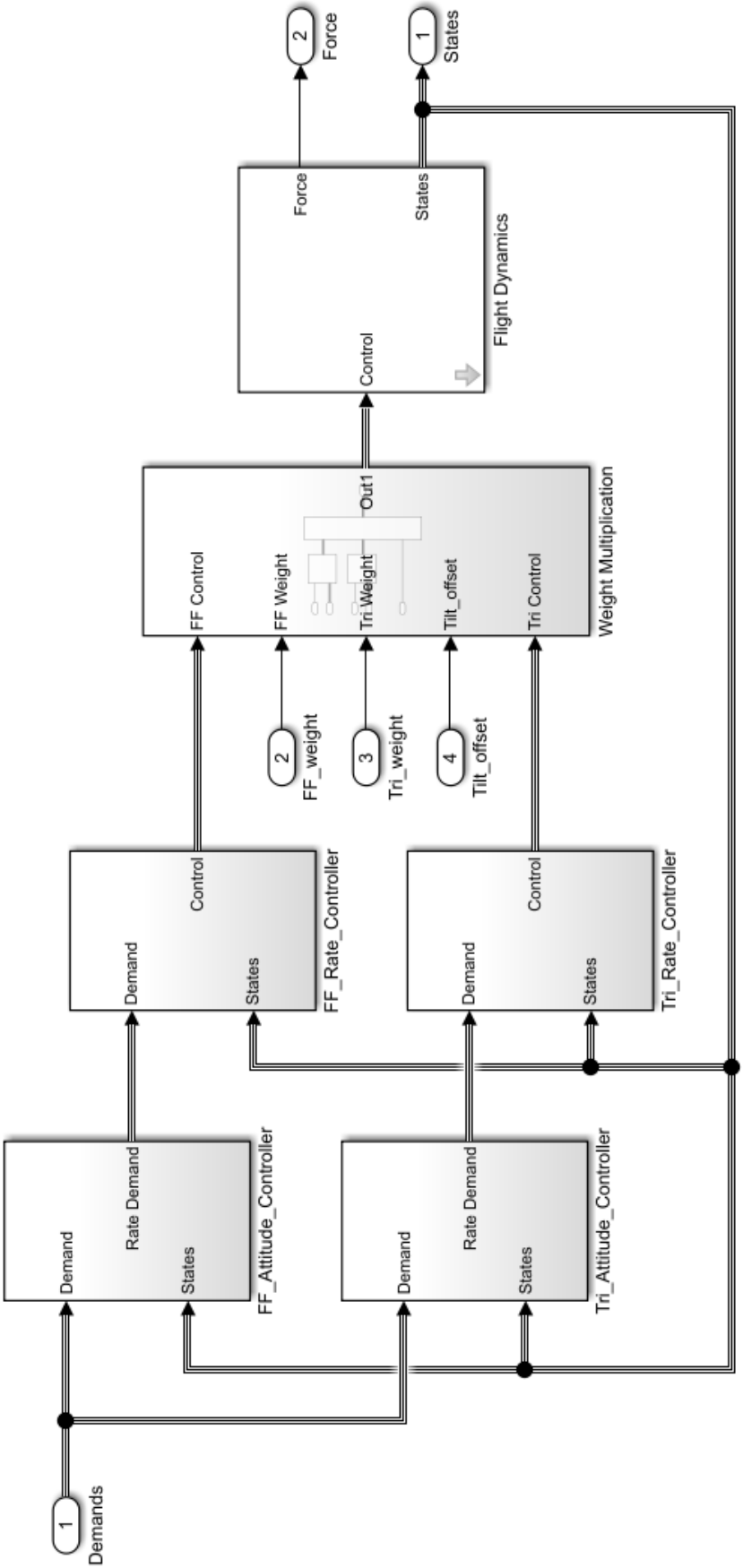


Where the function 'Stabilise_Weight' is:

```
function [weight,offset] = fcn( Tilt_offset, Tri_weight,Velocity)
    %sets the tricopter weight and offset to improve stability
    if Velocity(1) <8
        weight = 1;
        if Tilt_offset > 30
            offset = 30;
        else
            offset = Tilt_offset;
        end
    else
        weight = Tri_weight;
        offset = Tilt_offset;
    end
end
```

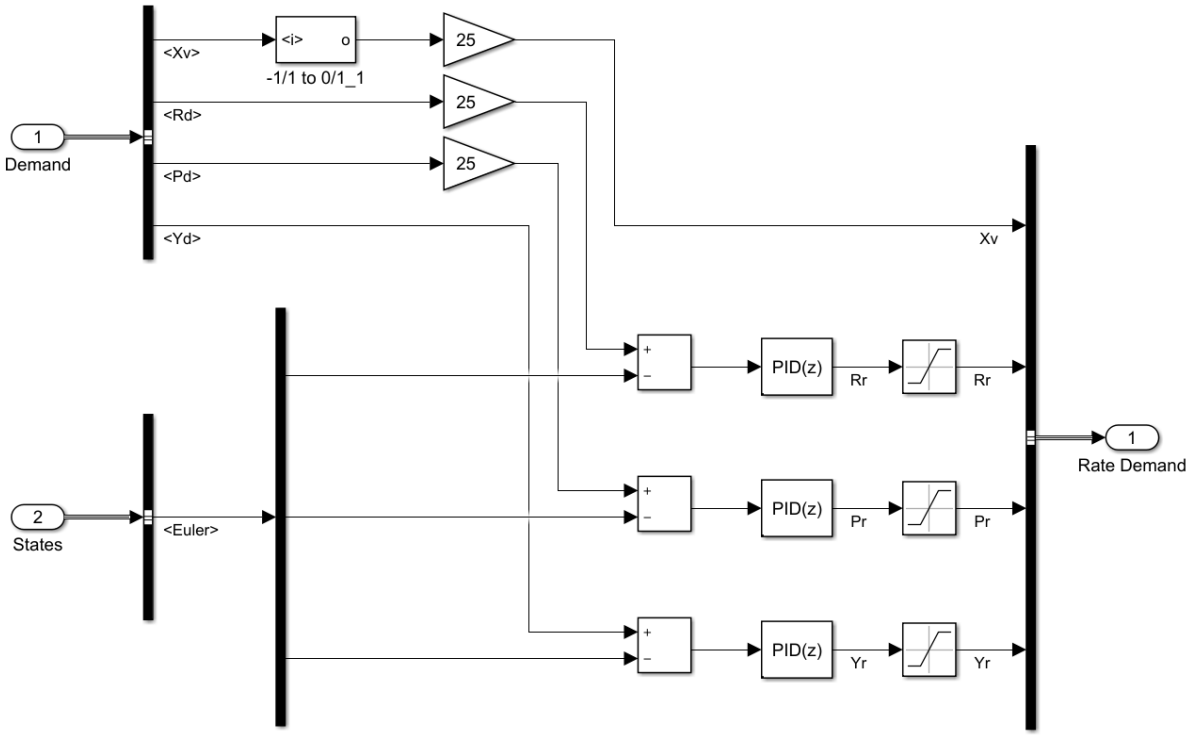
This stops the agent crashing when the weights, speed, and offset are very unbalanced.

iv. Aircraft Model

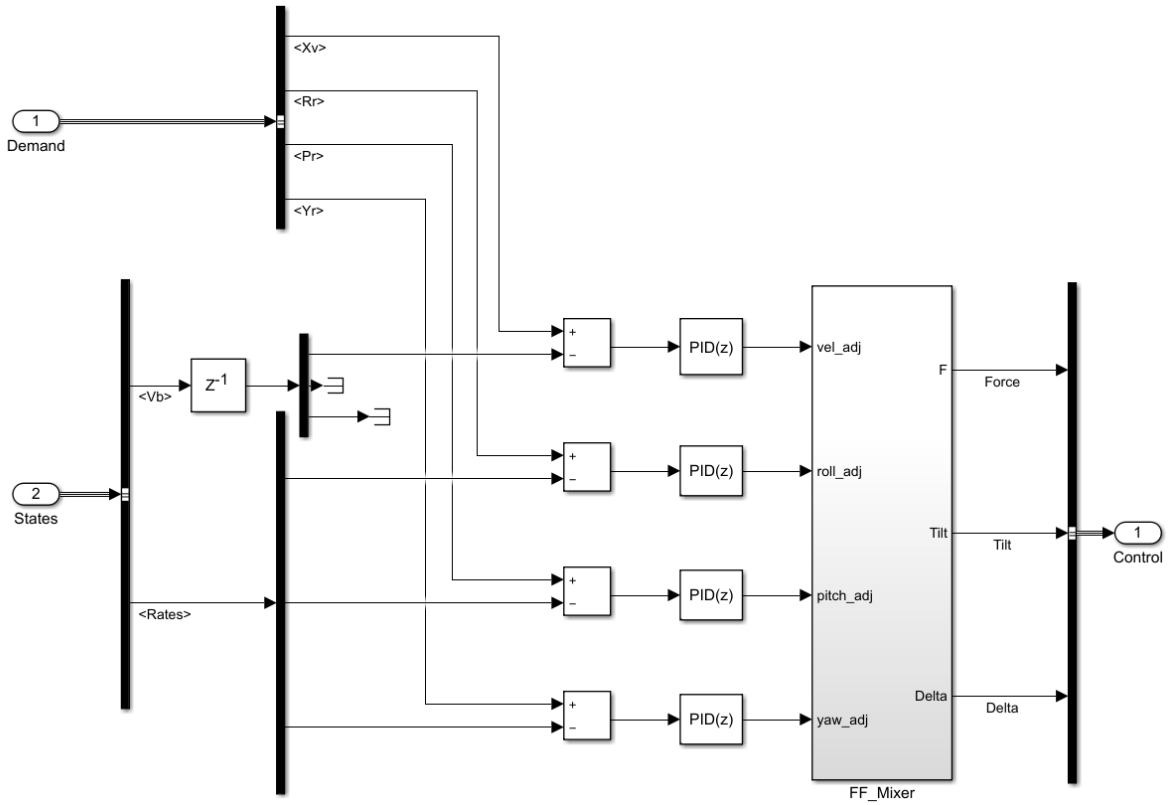


11.3 Flight Controllers

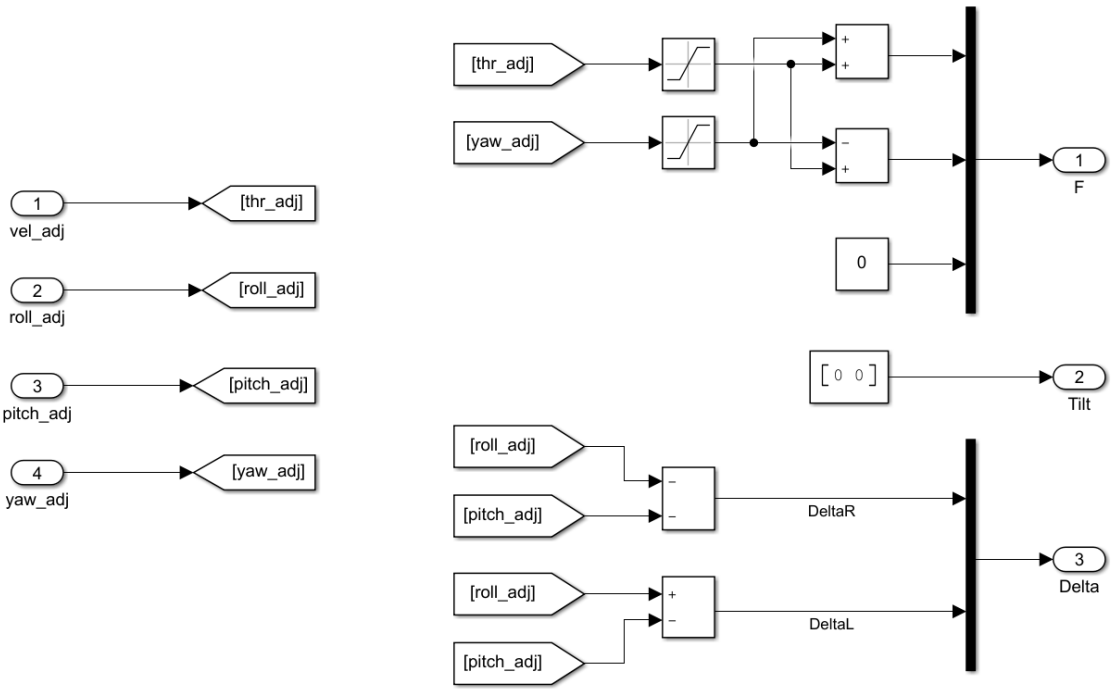
i. FF_Attitude_Controller



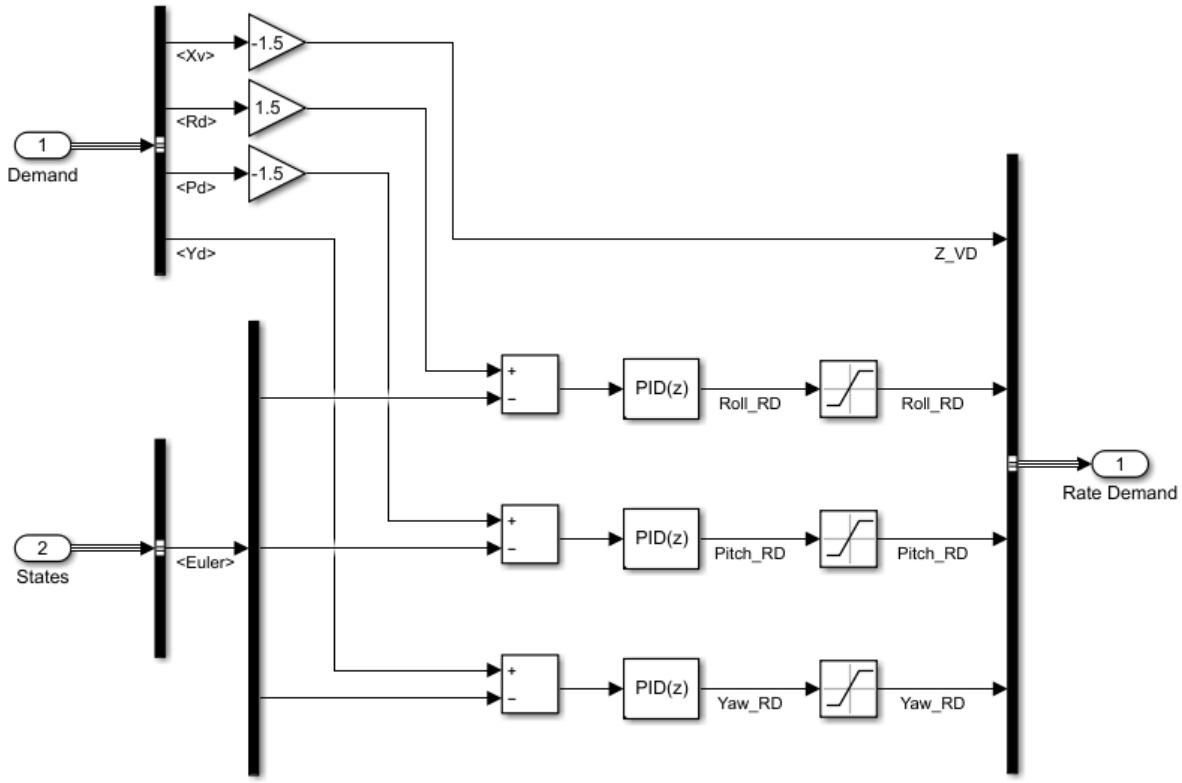
ii. FF_Rate_Controller



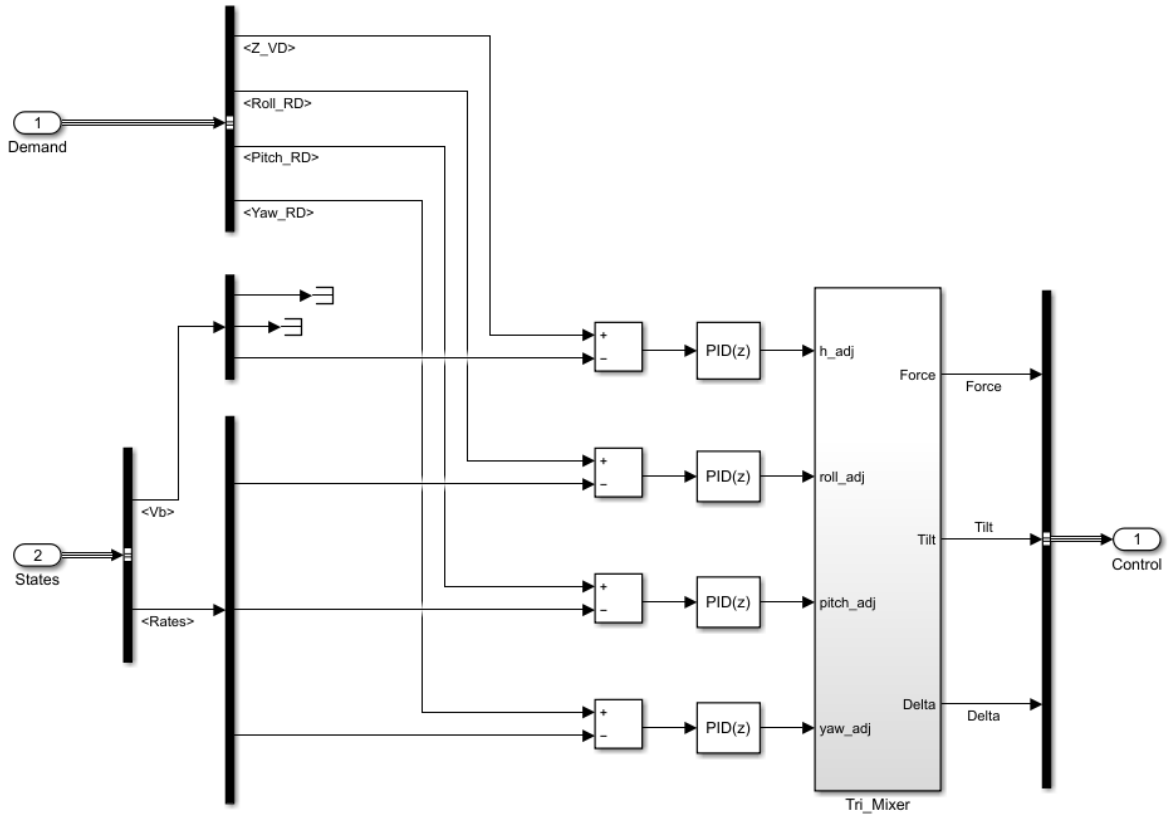
This has the following mixer



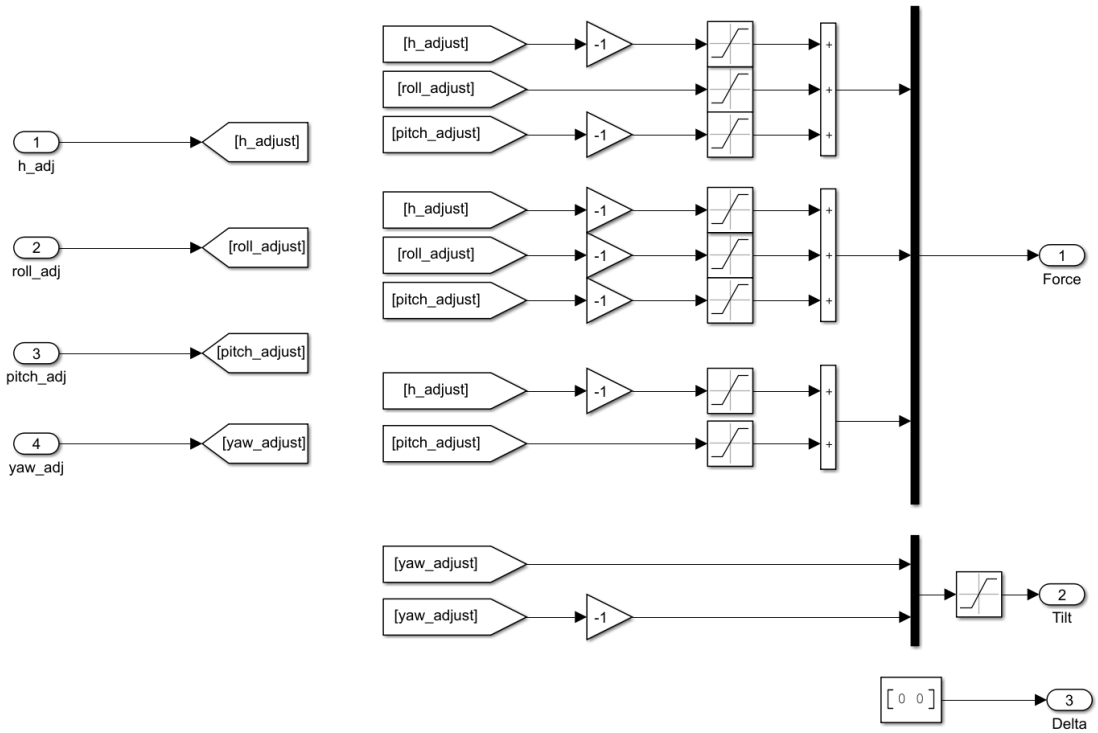
iii. Tri_Attitude_Controller



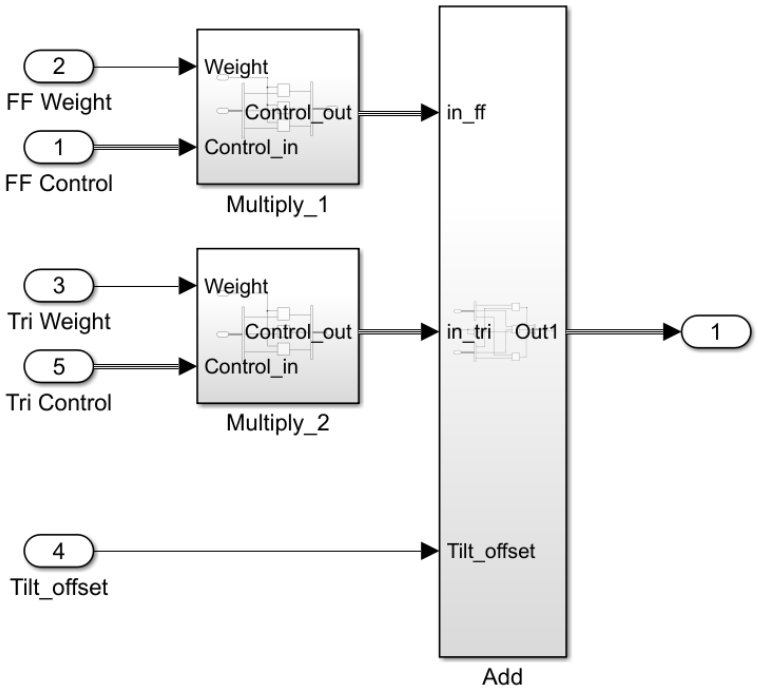
iv. Tri_Rate_Controller



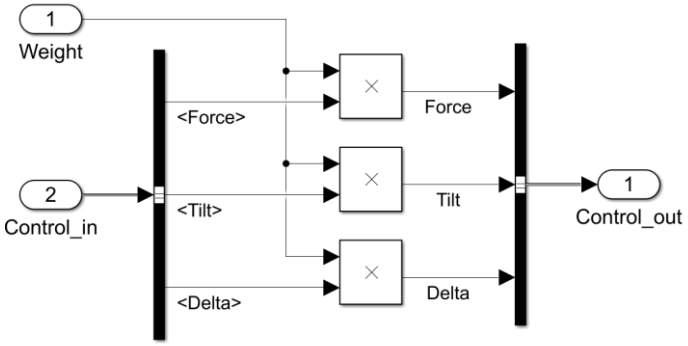
This has the following mixer



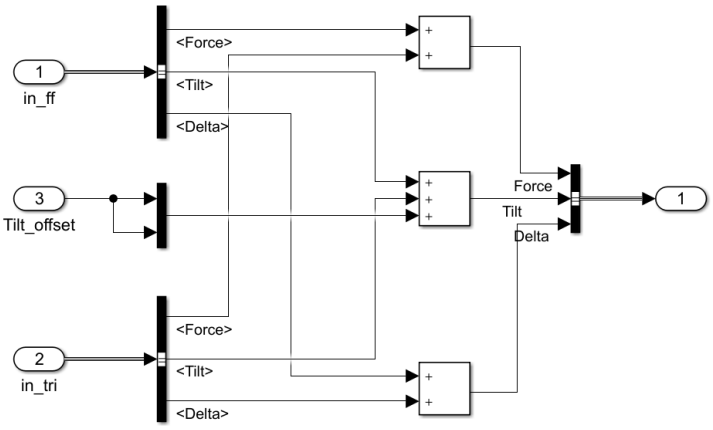
v. Weight Multiplication



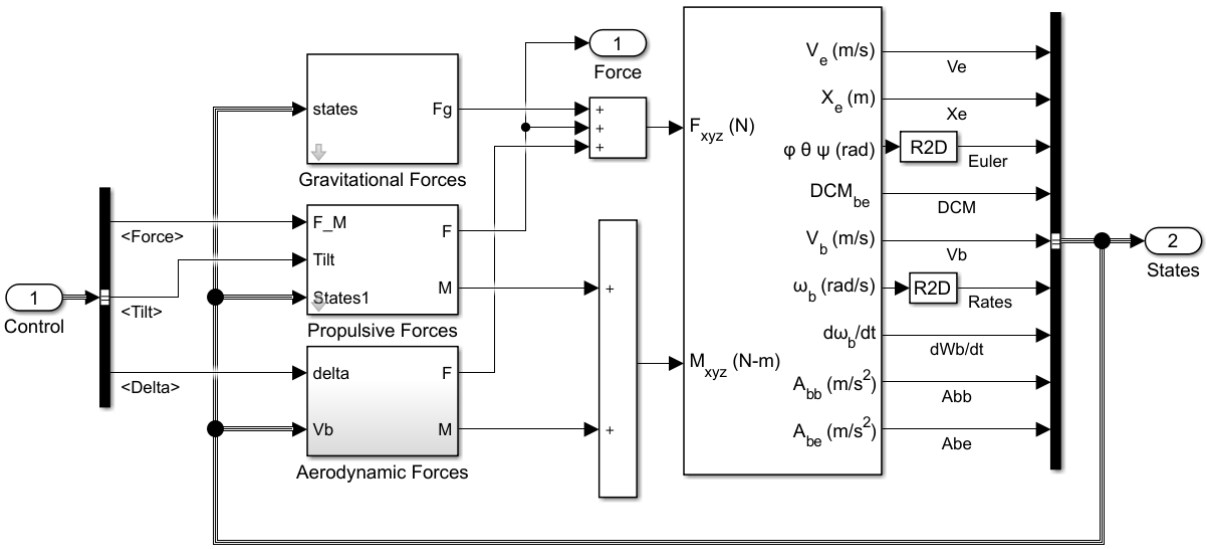
Buses are difficult to multiply, so they have blocks to handle this as follows:



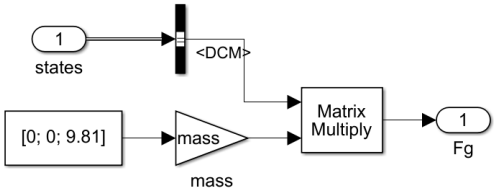
They are then added together in a separate block:



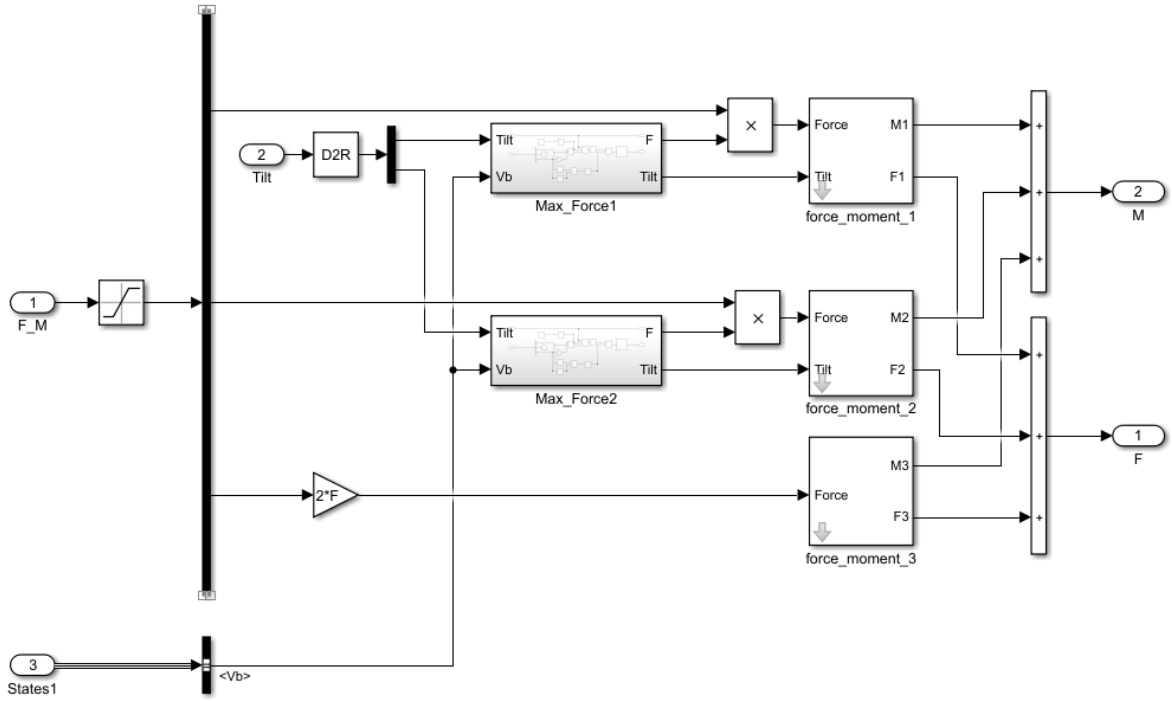
11.4 Flight Dynamics



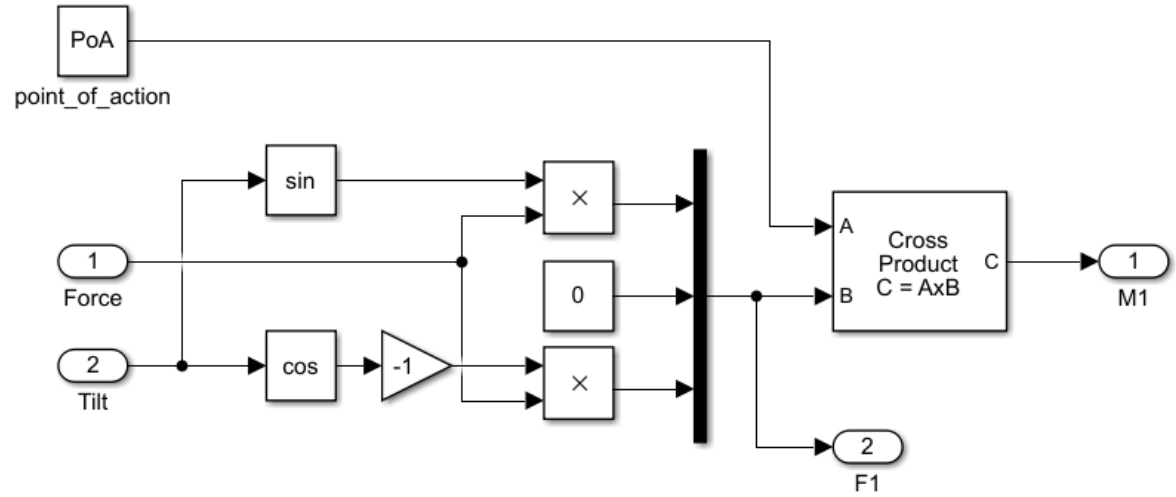
i. Gravitational_Forces



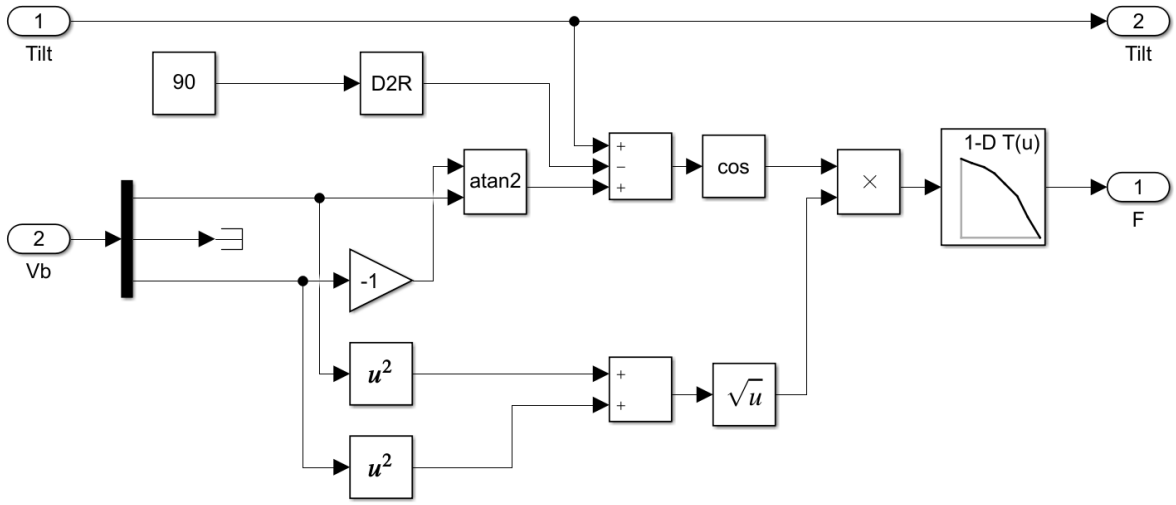
ii. Propulsive_Forces



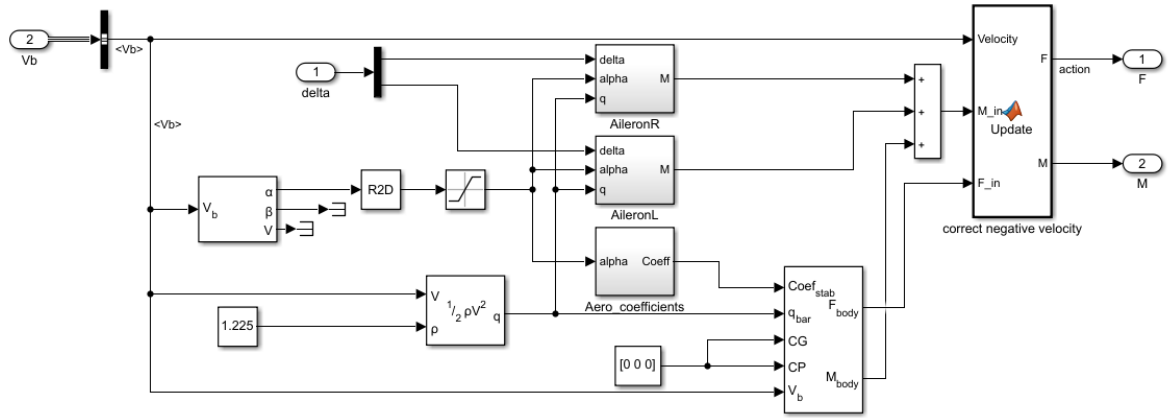
Where the Force_Moment blocks calculate the forces and moments based on the tilt angle and point of action.



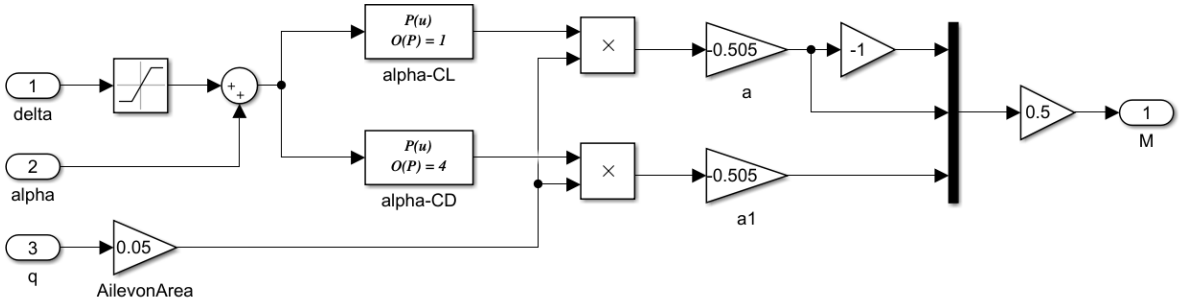
The maximum force is calculated from the air speed over the props



iii. Aerodynamic Forces and Moments



This has elevons as follow:



There are no forces when the velocity is negative:

```
function [F,M] = Update(Velocity,M_in,F_in)
    if Velocity(1) < 0
        F = [0;0;0];
        M = [0;0;0];
    else
        F = F_in;
        M = M_in;
    end
end
```