The Aeronautical Journal (2022), **126**, pp. 1101–1123
doi:10.1017/aer.2022.48

REGULAR PAPER

Improvements in learning to control perched landings

L. Fletcher[1,*], R. Clarke[1], T. Richardson[1] and M. Hansen[2]

[1]Department of Aerospace Engineering, University of Bristol, Bristol, UK and [2]Bristol Robotics Lab, Bristol, UK
*Corresponding author. Email: liam.fletcher@bristol.ac.uk

Received: 5 November 2021; **Revised:** 21 February 2022; **Accepted:** 11 April 2022

Keywords: UAV; Machine learning; Reinforcement learning; Perching; Agile manoeuvres

Abstract
Reinforcement learning has previously been applied to the problem of controlling a perched landing manoeuvre for a custom sweep-wing aircraft. Previous work showed that the use of domain randomisation to train with atmospheric disturbances improved the real-world performance of the controllers, leading to increased reward. This paper builds on the previous project, investigating enhancements and modifications to the learning process to further improve performance, and reduce final state error. These changes include modifying the observation by adding information about the airspeed to the standard aircraft state vector, employing further domain randomisation of the simulator, optimising the underlying RL algorithm and network structure, and changing to a continuous action space. Simulated investigations identified hyperparameter optimisation as achieving the most significant increase in reward performance. Several test cases were explored to identify the best combination of enhancements. Flight testing was performed, comparing a baseline model against some of the best performing test cases from simulation. Generally, test cases that performed better than the baseline in simulation also performed better in the real world. However, flight tests also identified limitations with the current numerical model. For some models, the chosen policy performs well in simulation yet stalls prematurely in reality, a problem known as the reality gap.

Nomenclature

a_t	an action
$\hat{A}_t$	advantage function estimate
DQN	Deep Q-Network
E	expectation
$\hat{E}$	expectation estimation
J	objective function
J^{CLIP}	PPO surrogate objective function
LSTM	Long-Short Term Memory
MLP	Multi-layer Perceptron
$\mathcal{N}$	normal distribution
PPO	Proximal Policy Optimisation
P	probability
q	pitch rate
R	return
RL	Reinforcement Learning
RNN	Recurrent Neural Network
r_t	reward
S	state space
s_0	initial state
s_t	state at time step t

This paper forms part of the 75th anniversary of aerospace engineering at the University of Bristol Collection

t	discrete time step
T	final time step
TRPO	Trust Region Policy Optimisation
u, w	velocity in aircraft body axis
$\mathcal{U}$	uniform distribution
u_r	airspeed component
UAV	Uncrewed Aerial Vehicle
VTOL	Vertical Takeoff and Landing
x	aircraft state vector
x_e, z_e	position in earth coordinates (z down)

Greek symbol

ϵ	PPO hyperparameter
η	elevator angle
θ	neural network parameters
θ_e	pitch angle
Λ	wing sweep
μ	dynamics parameters
π	policy parameterised by θ
τ	trajectory

1.0 Introduction

Fixed-wing uncrewed aerial vehicles (UAVs) typically have several advantages over multirotor vehicles, such as greater range and flight endurance. These attributes make them more suited to long-range surveillance or monitoring tasks. However, due to the nature of fixed-wing flight, these operations tend to be restricted to open environments. Runways or equipment such as catapults and arresting systems are often required to operate the UAV. In addition, conventional flight control systems for UAVs are designed for restricted flight envelopes, with limited ability to control highly agile manoeuvres. As such, they are not capable of sustained high-alpha flight or rapid avoidance manoeuvres. These constraints limit the ability for such fixed-wing aircraft to operate in complex environments, such as urban areas.

Vertical Take-Off and Landing (VTOL) aircraft, such as quad-plane and tailsitter configurations, are one method for landing in limited spaces. Another is to use a bio-inspired, perched landing approach, as described by Waldock et al. [1]. A perched landing uses a dynamic stall that decelerates the aircraft to safely land. This bio-inspired manoeuvre can be used to land small, fixed-wing aircraft in confined or challenging environments. Perching UAVs, both multirotor and fixed-wing, have been explored by others. For example, Meckstroth et al. [2] and Moore et al. [3] both demonstrated experimental perching manoeuvres with small fixed-wing UAVs. Meckstroth et al. used a motion capture system to measure pitch-up manoeuvres, generating a numerical model of perching based on component buildup methods. Moore et al. demonstrated perching at height, controlled using Linear Quadratic Regulator trees. More recently, Novati et al. [4] employ model-free reinforcement learning (RL) for controlled gliding and perching for a simulated elliptical body. Here, RL was able to generate robust controllers that could generalise to previously unseen initial conditions.

This paper builds upon previous work carried out by the Bristol Flight Lab, where a perched landing manoeuvre is performed by a custom variable-sweep aircraft using an RL controller. Greatwood et al. created the aircraft [5] and Waldock et al. demonstrated the efficacy of a DQN-based schedule optimiser when compared against a non-linear optimisation algorithm [1]. RL was used as a schedule optimiser in an open-loop manner, where the DQN algorithm was used to pre-generate a schedule of actions [1]. Clarke et al. trained an RL controller in simulation, then deployed it on the aircraft to form a closed-loop RL controller [6]. The aircraft state from the flight controller was sent to the deployed network to select an action during flight. Fletcher et al. investigated the addition of atmospheric disturbances to the simulation to reduce the gap between simulation and reality and improve real-world performance [7].

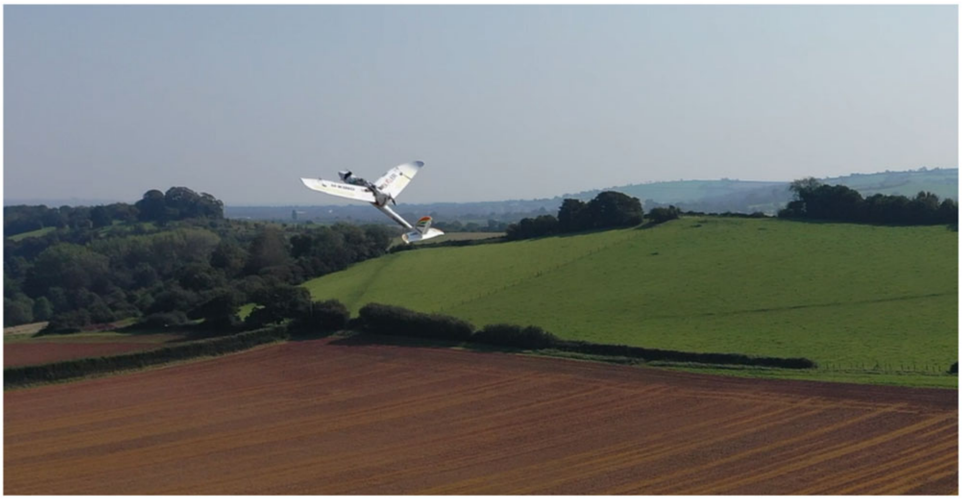

Figure 1. *Sweep wing Bixler 2 during flight testing.*

Modern reinforcement learning algorithms, using neural networks as non-linear function approximators, have been applied across several control tasks, though mainly in virtual environments, such as video games [8] and simulated robotics tasks [9]. RL control has been applied to several aerial robotics applications in recent years. For example, Koch et al. used RL to generate flight controllers for the attitude control of multirotors, demonstrating both simulated and real-world operation [10]. For fixed-wing aircraft, Bohn et al. used the Proximal Policy Optimisation (PPO) RL algorithm to make an attitude controller for an X8 UAV, demonstrating superior performance to that of a tuned PID [11].

Real-world applications, such as robotics, pose a greater challenge than simulated tasks. Current RL methods typically require a significant amount of experience to be collected during training, based on an agent's interactions with an environment. One approach to this is to learn in-situ, learning from experience gained by the robot in the real world. This can be unfeasible for an aerial robot due to the time required and the likelihood of damaging hardware. A more practical approach is to learn in simulation and then transfer the trained agent to the robot for real-world testing. However, it is unlikely that the simulator will precisely model reality. This is particularly true for highly dynamic environments such as agile fixed-wing flight. For example, a numerical model generated from wind tunnel testing will have several simplifications and sources of error. The cumulative discrepancies between the simulated environment and the real world are known as the reality gap.

This project investigates several enhancements and modifications to the reinforcement learning process to improve the performance of the perched landing controller, both in simulation and during flight testing. These enhancements span three key categories: input observation modifications, modelling refinements and changes to the reinforcement learning process and architecture. Results from simulation are presented, demonstrating the effect of each modification individually and the cumulative effect when combined. The objective is to find a combination of improvements that achieves greater performance than the baseline model. Results from flight testing are then shown, demonstrating the impact of these improvements on the real-world performance of the RL controller.

2.0 Aircraft model

The test platform for this work is a custom sweep-wing aircraft, based on a standard Bixler 2 model aircraft, seen in Fig. 1. The Bixler 2 is an off-the-shelf foam trainer model aircraft. The aircraft was modified with the installation of a servo and custom mechanism. The modifications allow the sweep angle of the wings to be rapidly changed during flight. Wingtip incidence control was also added. Rapidly varying sweep can generate pitching moments by changing the centre of lift relative to the centre of mass. These pitching moments are far in excess of what the small elevator alone can generate. Full details of the hardware modifications and airframe design are provided in Greatwood et al. [5].

As well as wing modifications, a suite of avionics hardware is installed, enabling flight testing. This hardware includes a Pixhawk flight controller, running a custom fork of the ArduPlane firmware [12], as well as sensors such as an airspeed sensor and GPS module. A Raspberry Pi is onboard to run the RL controller, and the Pi communicates with the Pixhawk over the MAVLink protocol.

The numerical model of the aircraft is based on wind tunnel data. A non-linear, longitudinal model was developed previously, with aerodynamic coefficients based on aircraft state. A full description of the wind tunnel testing and aerodynamic coefficients is given by Waldock et al. [1]. No propulsion is modelled as the perched landing manoeuvre does not require thrust. The ability to model atmospheric disturbances, in the form of steady-state winds and Dryden gusts, was later added to better resemble real-world flight conditions [7]. The numerical model is implemented in Python and forms the simulated environment for the reinforcement learning process. The model is wrapped as an OpenAI Gym [9] environment, making it compatible with popular RL libraries, such as Stable Baselines [13].

3.0 Reinforcement learning

Reinforcement learning is a machine learning paradigm with the objective of training an agent from experience. An agent maps states or observations to actions. A state is a complete description of the state of the world, with no hidden information. In robotics, a complete state is not usually possible, and with an incomplete state, or observation, gathered from the robot's sensors. During RL training, an agent interacts with an environment. The environment can be simulated, such as a video game or robotics simulator, or can be the real world. In this work, the training environment is the flight dynamics model of the aircraft. At each time step of the learning process, the agent receives an observation from the environment. Based on the observation, the agent uses a policy to determine the action to take. The action space describes how the agent interacts with an environment. In this work, the action space relates to the movement of the aircraft's control surfaces. The agent's policy decides the mapping from observation to action. The policy is updated based on a reward signal emitted by the environment.

Reinforcement learning is formalised as a Markov Decision Process (MDP), described by a four-element tuple: (S, A, R, P), where S is the set of all valid states, A is the set of valid actions, R is the reward function, and P is the transition probability function. At each time step t, the agent receives a state s_t and selects an action a_t based on the policy, $\pi_\theta(a_t|s_t)$. Commonly, the policy will be deterministic and is parameterised by θ, the weights of a neural network. Based on the state, or state-action pair, the agent will receive a reward r_t from the reward function, $r_t = R(s_{t+1}, a_t)$. The transition function $P(s_{t+1}|s_t, a_t)$ defines the probability of transitioning to s_{t+1} when starting from s_t and taking action a_t. Generally, in the RL setting, the transition probabilities will be unknown. This scenario is known as model-free learning, where the agent does not have access to a model to predict the next state from the current. In an episodic scenario task, the goal of reinforcement learning is to find a policy that maximises the expected return, Equation (1).

$$R = \sum_t r_t \tag{1}$$

RL is an area of active research interest, with a wide array of algorithms and approaches. A key differentiator is model-free and model-based algorithms. In model-based RL, the agent has access to, or learns, a dynamics model of the environment, which can then be used in a planning algorithm. In model-free RL, the agent has no access to the underlying model and instead directly maps states to actions. Model-free learning has had much more research interest and is more widely applied in robotics literature. Model-based RL tends to be more sample efficient than model-free, however it is more complex and challenging to train [14]. Clarke et al. used a Deep Q-Network (DQN), a model-free, value-based algorithm [6]. DQN, developed by Mnih et al., was the first deep-RL algorithm to be successfully demonstrated. A neural network is used as a non-linear function approximator for the Q-function [15]. Whilst DQN has demonstrated efficacy for some tasks, it has several limitations, including being limited to only discrete action spaces. More modern algorithms are compatible with continuous action spaces and bring other improvements such as reduced complexity and higher performance. Proximal policy optimisation (PPO), developed by Schulman et al. [16] is a more modern algorithm and has been widely applied across a range of tasks and domains. For example, it has demonstrated high levels of performance for real-world dexterous manipulation [17] and quadrupedal walking projects [18]. As well as being compatible with discrete and continuous actions, PPO tends to be less sensitive to hyperparameter selection and is less complex than other algorithms.

PPO is an on-policy, model-free RL algorithm. Unlike DQN, which is a value-based method, PPO is a policy-gradient method that aims to optimise a surrogate objective function using gradient ascent. Being an on-policy algorithm, this optimisation can only update with data collected while acting according to the most recent version of the policy.

For a generic policy gradient algorithm, the aim is to maximise an objective function, Equation (2).

$$J(\pi_\theta) = \int_\tau P(\tau|\pi_\theta)R(\tau) = \mathop{\mathbb{E}}_{\tau\sim\pi_\theta}[R(\tau)] = \mathop{\mathbb{E}}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T} R(s_t, a_t)\right] \tag{2}$$

where π is a stochastic policy parameterised by θ. For deep reinforcement learning, this is a neural network with weights θ. $P(\tau|\pi_\theta)$ represents the probability of a trajectory τ following policy π_θ, given by Equation (3). The transition model $P(s_{t+1}|s_t, a_t)$ is determined by the dynamics of the environment, and $\pi_\theta(a_t|s_t)$ is the probability of policy π_θ suggesting a_t given s_t.

$$P(\tau|\pi_\theta) = P(s_0)\prod_{t=0}^{T-1} P(s_{t+1}|s_t, a_t)\pi_\theta(a_t|s_t) \tag{3}$$

The policy is optimised through gradient ascent for policy gradient algorithms to find the optimal policy π^*. To numerically compute the policy gradient, the objective function is commonly expressed in the form of Equation (4), with the policy gradient given by Equation (5).

$$J(\pi_\theta) = \hat{\mathbb{E}}\left[\log \pi_\theta(a_t|s_t)\hat{A}_t\right] \tag{4}$$

$$\nabla_\theta J(\pi_\theta) = \hat{\mathbb{E}}\left[\nabla_\theta \log \pi_\theta(a_t|s_t)\hat{A}_t\right] \tag{5}$$

Here, $\hat{A}_t$ is the estimate of the advantage function, a measure of how good an action is when compared with other actions on average, relative to the current policy. An actor-critic network architecture is commonly used to estimate $\hat{A}_t$, with a separate critic network trained to predict the value function. Employing the advantage function helps to reduce the variance of the gradient estimates.

This form of the policy gradient seen in Equations (4) and (5) is used in the vanilla policy gradient algorithm. However, performing optimisation using such a methodology can lead to large policy updates that cause instability during training. As such, Trust Region Policy Optimisation (TRPO) was created to limit the size of policy updates by enforcing Kullback–Leibler constraints [19]. PPO was created to achieve similar benefits and performance as TRPO whilst being less complex and easier to implement. PPO uses a clipped surrogate objective function to constrain the update's size. The surrogate objective function is given by Equation (6). Here, $r_t(\theta)$ is the probability ratio between the new and old policies, Equation (7), and ϵ is a hyperparameter that requires tuning.

$$J^{\text{CLIP}}(\theta) = \hat{\mathbb{E}}\left[\min\left(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t\right]\right. \tag{6}$$

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} \tag{7}$$

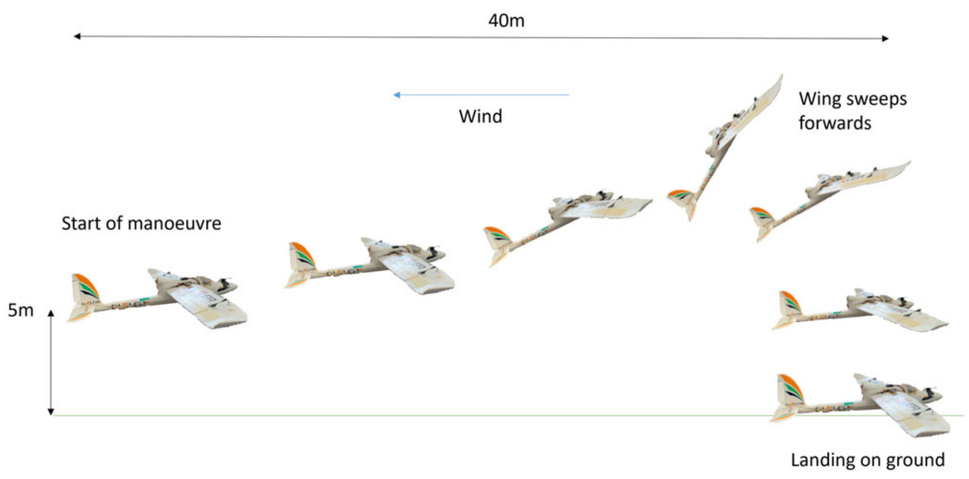

Figure 2. *Perched landing diagram.*

4.0 Perched landing

4.1 Manoeuvre description

Pure fixed-wing vehicles typically require a significant length of clear ground to land intact unless specialist equipment such as arresting gear can be used. This imposes significant restrictions on their use in congested environments or minimally prepared locations. However, many birds demonstrate the ability to land in unprepared locations and minimal distance when landing on a perch. If a similar manoeuvre can be carried out by a fixed-wing vehicle, it would allow for significantly increased flexibility in operations.

The ability of a bird to change the shape of its wings far outweighs that available to conventional fixed-wing aircraft, imparting a significant advantage in available control effort relative to their size. The many degrees of freedom that enable this are currently impractical to integrate into a fixed-wing platform. Therefore, the vehicle used in this work only has the additional ability to vary the sweep of its wings. This means the required trajectory is likely to differ significantly from that of a bird. Because of this, rather than specifying a desired trajectory for the perching manoeuvre, only the desired final state is specified.

The aircraft in this work can use the variable-sweep wings and elevator in unison to generate significant pitching moments, enabling a rapid increase in drag. This can be used to quickly slow the vehicle, helping to achieve the desired final state. Figure 2 shows a representation of the path and attitude of the vehicle throughout such a manoeuvre. The dynamics of the aircraft throughout this manoeuvre are highly non-linear, particularly the dynamic stall effects caused by the rapid change in main wing pitch [20]. This poses a significant challenge for a conventional flight controller. Waldock et al. approached the problem by using a set of pre-computed, open-loop schedules to drive the wing sweep and elevator. Closed-loop control was then used to correct for deviations in pitch rate from the pre-computed schedule, with control effected by additional elevator deflection. The schedule optimised through reinforcement learning was shown to perform better than that generated by a more conventional optimiser. More recently, as in this work, an RL agent has been directly used as a closed-loop controller.

Before performing a perched landing, it is assumed that the aircraft is 40m away from the desired landing site, heading towards it in level flight and at a constant airspeed. The manoeuvre is unpowered, so the throttle is set to zero at the start of the manoeuvre. In earlier work, the manoeuvre was started at a height of 2m above the target. However, when facing headwinds, this was too limiting, so was increased to 5m.

4.2 Reward function

The objective of the perched landing manoeuvre is to land flat on the ground, with velocities close to zero. As such, the reward is a function of the five relevant longitudinal aircraft states, $(x_e, z_e, \theta_e, u, w)$. The reward function generates a scalar output from these parameters. The reward is only returned at the end of an episode, when the terminal state is reached. This is known as a sparse reward.

The terminal state is reached when the aircraft height relative to the target, z_e, becomes zero, simulating a landing on the floor. The reward is calculated based on the error between the final and target states. In this case, the target value for each state element is zero. The episode will end early if the aircraft breaches boundary conditions, such as maximum airspeed or pitch angle. In these cases, the reward for the episode is 0.

Waldock et al. used a reward function based on a weighted dot product of the mean square error for each parameter. However, with the introduction of atmospheric disturbances, this function resulted in undesirable multi-modal behaviour, with high rewards given to trajectories that would not reach the target position intact. Experimentation with modifications and alternative functions was performed in previous work, leading to the reward function that has been retained for this work [7]. The reward function is based on the product of a Gaussian applied to the error in each state element. This reward function is crafted such that a high reward will only be returned if all the target states are within acceptable bounds, otherwise the reward is near zero. Each state is normalised before being passed through the Gaussian function, with the normalisation values found from iterative experimentation until the desired behaviour was achieved. The reward function is defined in Equation (8).

$$
\begin{aligned}
\vec{s}_t &= (x_e, z_e, \theta_e, u, w)\\
\vec{b} &= (15, 5, 20, 10, 10)\\
c_i &= \frac{s_{t_i}}{b_i}\\
g(c_i) &= \frac{1}{\sigma\sqrt{2\pi}} e^{-(c_i-\mu)^2/2\sigma^2}\\
r_t &= \prod_i g(c_i)
\end{aligned}
\tag{8}
$$

where s_{t_i} represents each state element, μ is the target value for each element, in this case zero for all, and σ is 0.4.

5.0 Training improvements

This paper follows a similar methodology to previous work, where reinforcement learning models are trained in simulation and then deployed onto the real-world vehicle for flight testing. A series of modifications to the training process were investigated to improve real-world flight performance by minimising terminal state error.

5.1 Input observation

Previous flight testing of the perched landing manoeuvre demonstrated that atmospheric disturbances, steady winds and gusts, significantly impact real-world performance. Simulating wind and gusts showed an increase in mean reward achieved over the baseline models, with room for improvement. The baseline input observation, Equation (9), are the longitudinal states of the aircraft, including the longitudinal position of the aircraft in the world frame, x_e, z_e, the pitch angle θ_e, longitudinal body velocities u, w, pitch rate q, and wing sweep and elevator angles, Λ and η.

$$x = [x_e, z_e, \theta_e, u, w, q, \Lambda, \eta] \tag{9}$$

This observation, however, does not include any information about the state of the air. It was hypothesised that learning, and real-world performance, could be improved by augmenting the input state with air mass data. While the wind vector could be added to the observation in simulation, this information is challenging to measure in the real world. The test aircraft has an airspeed sensor, which the flight controller uses during automated flight. As such, this is a state that can be used in the RL controller. During training, the component of airspeed along the body x-axis is added to the input vector to give the augmented input vector, Equation (10).

$$x = [x_e, z_e, \theta_e, u, w, q, \Lambda, \eta, u_r] \tag{10}$$

5.2 Improving simulation

The reality gap is the difference between simulation and reality and poses a significant problem when performing experimental flight tests. Any uncertainty or error in the simulator can lead to poor real-world performance. Sources of discrepancy include unmodelled real-world dynamics, incorrect parameters, and stochasticity in the real environment. Accurately modelling fixed-wing flight is particularly challenging, with a typical numerical model requiring several assumptions and simplifications. For example, a numerical model constructed from wind tunnel data will have sources of error such as wall interference effect and loadcell measurement noise.

Domain randomisation is one technique used to improve the sim-to-real transfer of models trained in simulation. A policy trained using domain randomisation will be more robust to modelling uncertainty and error by experiencing variations of the domain. Whilst training the RL policy during simulation, various dynamics parameters are sampled. Peng et al. and Tan et al. have shown it to be effective for generating robust controllers for robotic arm manipulation tasks [21] and locomotion for quadrupedal robots [18]. Formally, the objective function in Equation (2) is modified to produce Equation (11), to maximise the expected return across a distribution of dynamics models ρ_μ, where μ is a set of dynamics parameters [21].

$$J(\pi_\theta) = \mathop{\mathbb{E}}_{\mu\sim\rho_\mu}\left[\mathbb{E}_{\tau\sim p(\tau|\pi_\theta,\mu)}\left[\sum_{t=0}^{T-1} R\left(s_t, a_t\right)\right]\right] \tag{11}$$

Domain randomisation was previously introduced for this perched landing scenario by randomising wind and gusts during training [7]. Steady-state winds were randomised at the start of each episode, sampled from a uniform distribution, and gusts were injected as Dryden noise. Further sources of domain randomisation are introduced in this work.

Sensor noise is simulated using a similar method as suggested by Peng et al. [21]. Gaussian noise is applied to the observation at each time step, with a mean of zero and a standard deviation of 5% of the running standard deviation for each observation feature. A small amount of acceleration noise, in the form of Gaussian noise, is applied to the acceleration term of the numerical model. This is similar to the noise applied by Walodck et al. where it was found to improve performance in the real world. This noise augments the Dryden gusts already added, helping to generate models that are more robust to numerical model inaccuracies and disturbances during flight. A noise level of 0.3m/s^2 is used in these experiments.

Whilst the flight controller aims to reach an initial target state before handing over control to the RL model, in reality, there is variation in these start conditions. Previous experiments demonstrated variation in initial airspeed and pitch angle in particular. As such, initial state noise was simulated between episodes, with initial pitch angle and airspeed selected from normal distributions. Table 1 details all of the domain randomisation parameters and values defined in this section.

Peng et al. identified action latency and observation noise as having a significant impact on RL training performance for controlling a robotic arm [21]. Ibarz et al. suggest that actuator dynamics and the lack of latency modelling are the leading causes of modelling error for learning quadrupedal walking [22]. A characterisation of the action latency on the test aircraft was performed. One source of latency for the real system is communication between the flight controller and RL model. At each time step,

Table 1. *Domain randomisation parameters*

Parameter	Modelling details	Units
Steady wind	$\mathcal{U}(-8, 0)$	m/s
Dryden gusts	Dryden($L_u, L_v = 2, \sigma_u, \sigma_v = 1.06$)	m/s^2
Action latency	$18 + lognormal(0, 1)$	ms
Sensor noise	$\mathcal{N}(0, 0.05S_{x_i})$	–
Acceleration noise	$\mathcal{N}(0, 0.3)$	m/s^2
Initial variance	Airspeed: $\mathcal{N}(13, 1)$ Pitch: $\mathcal{N}(0,3)$	m/s deg

the flight controller sends the input observation to the onboard computer over a serial connection, using the observation to select an action from the trained model. This action is then sent back to the flight controller over the serial connection to be performed by the servos. This communication and computation latency was characterised by timing the delta between the flight controller sending the state and it receiving the control action from the model. This latency model does not consider mechanical latency.

Table 1 summarises all of the domain randomisation parameters used in this work, including the steady-state wind gust parameters. Similar to previous work, steady-state winds are selected from a uniform distribution. Only headwinds are modelled in this work, up to 8m/s due to poor performance of tailwind scenarios during previous flight testing. The modelling of Dryden gusts remains unchanged, using the process and parameters for low altitude, light turbulence defined by Beard and Mclain [23] and Langelaan et al. [24].

5.3 Hyperparameter optimisation

Investigations into improving the performance of the underlying RL algorithm were performed. Current RL algorithms are highly sensitive to appropriate hyperparameter selection and require significant optimisation for maximum performance. Henderson et al., for example, demonstrated the effect of changing several hyperparameters for several algorithms, including PPO [25].

In previous research, the hyperparameters and network structure remained as the default values defined in the baseline implementation of PPO in Stable-Baselines [13]. These settings are defined to achieve good performance across a range of environments, particularly the OpenAI Gym and Atari environments [9]. With sufficient computational capacity, a full hyperparameter sweep would allow for the optimisation of each hyperparameter. Instead, with the resources available, a more limited investigation was performed to identify the parameters that would significantly impact training performance. An initial hyperparameter sweep was performed using the Weights and Biases sweep tool [26]. A Bayesian sweep was performed on ten parameters, based on the mean reward after 5 million time steps, for a total of 102 runs. The hyperparameter sweep showed that changing the `n_steps` parameter had a significant effect on training performance. This parameter is defined as the number of time steps to run for each environment before running the PPO update [13]. Figure 3 shows a training history plot, comparing the reward per time step for several `n_steps` parameters. The plot shows a general trend of the reward increasing as `n_steps` increases, with the highest reward obtained when the parameter is set at 2,048, the highest value used in the sweep. With a higher number of steps, the policy update will be more generalised, instead of fluctuating between extreme policies that may be present if `n_steps` is lower. For the models in this paper, this is the value used, yet future investigations with `n_steps` greater than 2,048 could lead to further improvement.

As well as the sweep, tuned hyperparameters from other environments were identified from literature, such as those provided by the Stable Baselines Zoo collection [27]. This research suggested that increasing the `nminibatches` hyperparameter from the default value should lead to improved performance. This parameter is defined as the number of training minibatches per update. From the research

Table 2. *Change of hyperparameters*

Parameter	Default value	Tuned value
`n_steps`	128	2,048
`nminibatches`	4	32

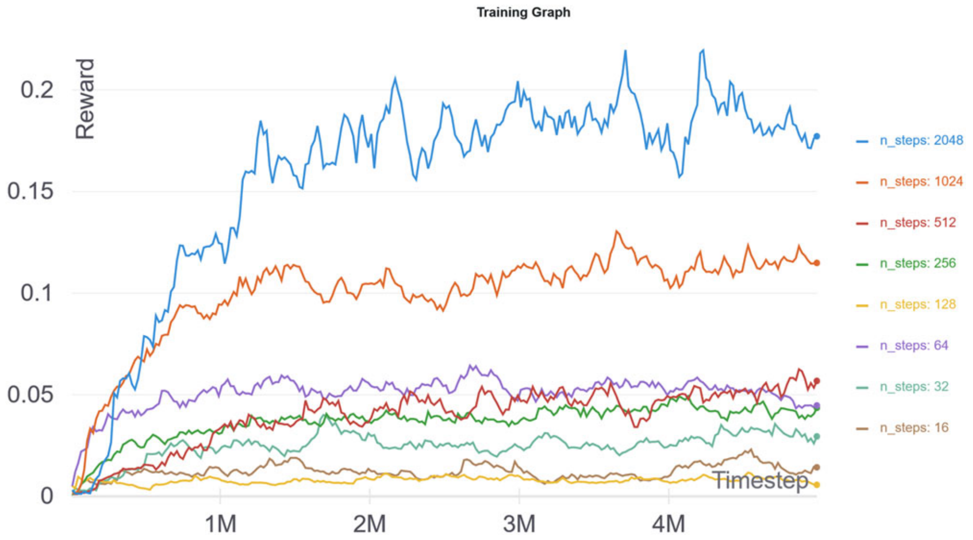

Figure 3. *Training graph, showing reward against time steps for several values of the* `n_steps` *hyperparameter.*

and investigations carried out, it was decided that these two parameters would be varied in a number of test cases. Table 2 summarises these values.

5.4 Network architecture

Previous work has used a fully connected multi-layer perceptron (MLP) neural network architecture with two layers of 64 nodes. This is the default network architecture for an MLP as defined in the Stable-Baselines library [13]. One potential change is to increase the size of the network, such as changing the number of layers or the number of nodes of each layer. Raffin et al. and Henderson et al. both show performance increases when using a larger neural network [28,25]. As such, investigations with a more extensive network architecture, two layers of 256 nodes, were performed in this work.

With the use of further domain randomisation as described in Section 5.2, there is a significant change in the dynamics of the environment during training. Peng et al. suggest that a policy with a form of memory will perform better than a standard MLP in such a scenario [21]. One method of adding memory is the use of recurrent neural networks (RNNs), such as the Long-Short Term Memory (LSTM) architecture used by OpenAI [17]. However, RNNs are usually more challenging to train, requiring finer tuning of the underlying hyperparameters and increased computational time. An alternative approach, as used by Haaronja et al. [29] and Xiao et al. [30], is frame stacking. Instead of a single observation, the observation at each time step is augmented by a window of $n - 1$ previous observations, where n is the number of frames in the stack. Frame stacking can be used with a standard MLP structure, requires minimal modification to the existing architecture, and is easier to train than an LSTM. Initial investigations were conducted to identify the number of frames n to use, summarised in Fig. 4. The plot shows that using a stack of four frames results in a greater reward during evaluation than eight. As such, test cases using frame stacking in Section 7 use a n of 4.

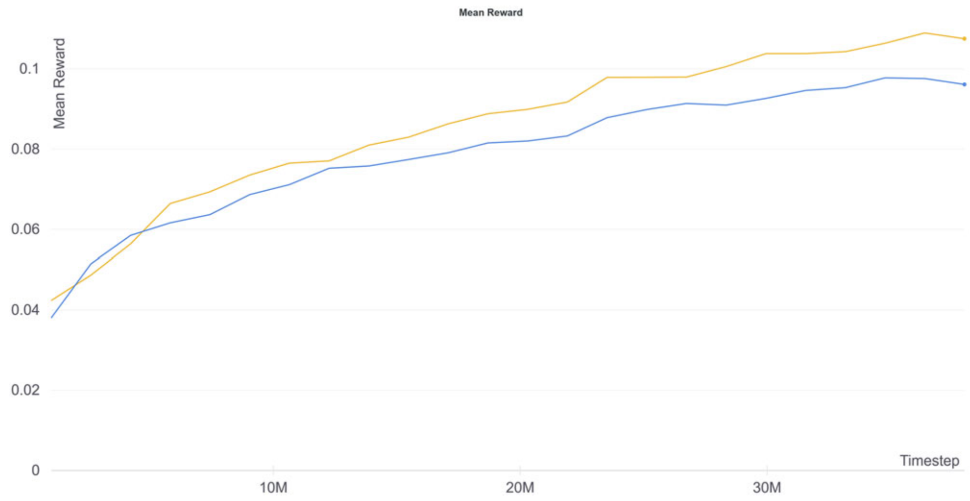

Figure 4. *Mean reward obtained during training by models using frame stacking, with n of four (yellow line) and n of eight (blue line).*

5.5 *Continuous actions*

Discrete actions spaces have been used previously with both Deep Q-Network [6] and PPO [7]. The actions are encoded as a pair of arrays of angular rates for the wing sweep and elevator, with the action effectively selecting a pair of indices into these arrays. Equation (12) shows the discrete action arrays.

$$\begin{aligned}\dot{\Lambda} &= [-60, -10, -5, 0, 5, 10, 60]^\circ/s \\ \dot{\eta} &= [-60, -10, -5, 0, 5, 10, 60]^\circ/s\end{aligned} \tag{12}$$

PPO is compatible with continuous action spaces, where ranges of actions are defined, and the model selects a continuous value in that range. In this case, the continuous action space was encoded as the minimum and maximum angular rates for both the sweep and elevator. It was hypothesised that, with a finer level of control over actions at each time step, the agent would be able to achieve greater performance levels. However, the use of continuous actions expands the size of the action space and can increase learning difficulty. Equation (13) shows a continuous representation of actions.

$$\begin{aligned}-60^\circ/s \le \dot{\Lambda} \le 60^\circ/s \\ -60^\circ/s \le \dot{\eta} \le 60^\circ/s\end{aligned} \tag{13}$$

6.0 Simulation methodology

For each test case, five instances are trained using different random seeds. Each instance is trained for 40 million time steps on eight CPU cores of the University of Bristol's BluePebble HPC, with each instance taking approximately 72 hours to train. This represents approximately 1 million simulated attempts at performing the manoeuvre. Multiple workers are employed, where experience from eight environments per time step is used to train the agent.

Once trained, each test case was evaluated in simulation. This was performed by running each trained model through the simulator to obtain time histories and simulated rewards. Two methods of evaluation are used in the simulated investigations. The baseline evaluation represents the evaluation scenario used in previous work, with randomisation of only steady winds and gusts. Enhanced evaluation employs further domain randomisation, such as variable initial conditions, latency and sensor noise, as described in Section 5.2. This presents a more challenging environment, with trained models expected to receive less reward in this scenario than when evaluated under baseline conditions. Table 1 provides details

Table 3. *Test Case configurations for the first set of simulation evaluations*

Test Case	Airspeed input	Domain randomisation	Network size	frame stacking	Hyperparameters	Actions
Baseline	No	No	[64, 64]	No	Default	Discrete
Test Case 1	Yes	No	[64, 64]	No	Default	Discrete
Test Case 2	Yes	Yes	[64, 64]	No	Default	Discrete
Test Case 3	Yes	Yes	[256, 256]	No	Default	Discrete
Test Case 4	Yes	Yes	[256, 256]	Yes	Default	Discrete
Test Case 5	Yes	Yes	[256, 256]	Yes	Tuned	Discrete

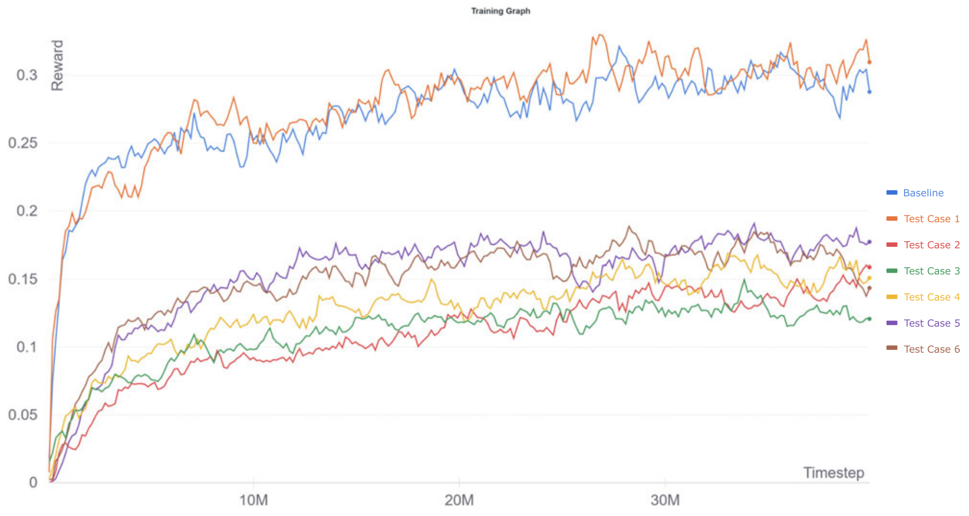

Figure 5. *Training history for test cases detailed in Table 3.*

of the domain randomisation parameters used in the two evaluation scenarios. During evaluation, each model is evaluated against a range of steady wind speeds. To account for the stochastic nature of the evaluation environments, the reward presented at each wind speed for each test case is the mean of 100 evaluations for the five instances.

7.0 Simulation results

The first set of experiments looked to compare the performance of adding each improvement in succession, with the baseline test case representing a model without any improvements. Table 3 shows the test cases for this set of investigations.

Figure 5 shows the training history of the test cases from Table 3. Each line represents the average training performance for the five instances of each test case. The plot shows the reward obtained per time step. An overall increase in reward with time step is expected to show that the models are training successfully. An exponential moving average with a smoothing factor of 0.7 is applied to the data. The baseline and Test Case 1, in general, obtain the highest reward per episode, due to the lack of additional variation of the domain during training. Using domain randomisation for the other test cases leads to a more challenging scenario with greater variance between episodes, hence lower reward. For the remaining test cases described, Test Cases 5 and 6 demonstrate generally higher rewards per episode, with Test Cases 2 and 3 obtaining the lowest reward per episode during training. This suggests that improvements such as frame stacking and tuning of hyperparameters lead to greater training performance when additional noise is added.

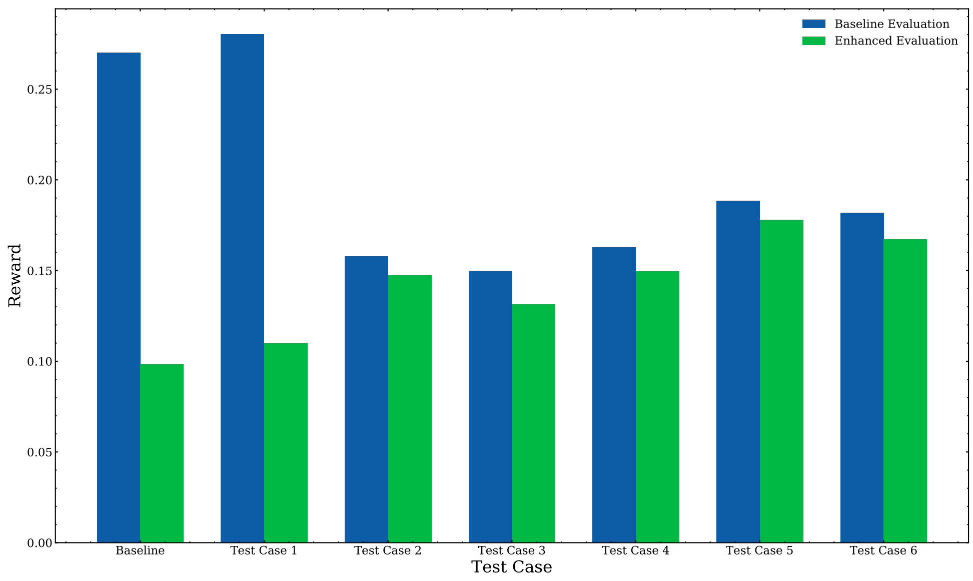

Figure 6. *Mean reward for 100 runs of each test case under the baseline evaluation (steady state wind with Dryden gusts) and under the enhanced evaluation with domain randomisation enabled.*

Figure 6 shows a plot of the mean rewards for the two evaluation scenarios, with higher reward showing superior performance. Figure 6 shows that, when evaluated against the baseline scenario, the baseline model and Test Case 1 obtain significantly higher rewards than the other test cases. Test Cases 2–6 add additional noise to the environment, and there is a general decrease in mean reward. The baseline and Test Case 1, trained without noise, have more experience during training of this simpler environment and obtain higher rewards when evaluated against the baseline scenario. However, these models perform poorly when evaluated against the enhanced scenario. These models cannot generalise to the noisier environment, as they have insufficient experience of these states during training. In general, across all test cases, there is a decrease in reward from the baseline evaluation to the enhanced evaluation, due to the more challenging environment.

Test Case 1 shows that the addition of airspeed into the observation leads to a slight increase in mean reward across both evaluations. This suggests that this additional information is useful to the model when learning. The addition of noise during training in Test Case 2 results in an increase in reward, as these models have experience of a more comprehensive array of states and can generalise more successfully. Test Case 5 obtains the highest mean reward of 0.178 under enhanced evaluation. This test case employs all previous improvements yet still uses discrete actions. Continuous actions in Test Case 6 lead to a decrease in mean reward compared to Test Case 5. This is most likely a product of the training constraints. With sufficient training time and a comprehensive parameter tuning process, the models with continuous actions would likely show performance above that of discrete. Test Case 3 also shows worse performance than Test Case 2, which is identical except uses a larger network. Whilst this result suggests that the smaller default network size should be used, further investigation showed that performance is sensitive to the combination of improvements. As such, the best performing model may not be that with all improvements enabled.

7.1 Combinations of modifications

Based on the first set of simulation results, a series of tests were conducted with various combinations of improvements to identify the best performing model. This section provides results from a selection

Table 4. *Test Case configurations for the second set of simulation investigations*

Test Case	Airspeed input	Domain randomisation	Network size	frame stacking	Hyperparameters	Actions
Baseline	No	No	[64, 64]	No	Default	Discrete
Test Case 3	Yes	Yes	[256, 256]	No	Default	Discrete
Test Case 5	Yes	Yes	[256, 256]	Yes	Tuned	Discrete
Test Case 6	Yes	Yes	[256, 256]	Yes	Tuned	Continuous
Test Case 7	No	Yes	[64, 64]	No	Default	Discrete
Test Case 8	No	Yes	[64, 64]	Yes	Tuned	Discrete
Test Case 9	No	Yes	[256, 256]	Yes	Tuned	Discrete
Test Case 10	Yes	Yes	[64, 64]	Yes	Tuned	Discrete

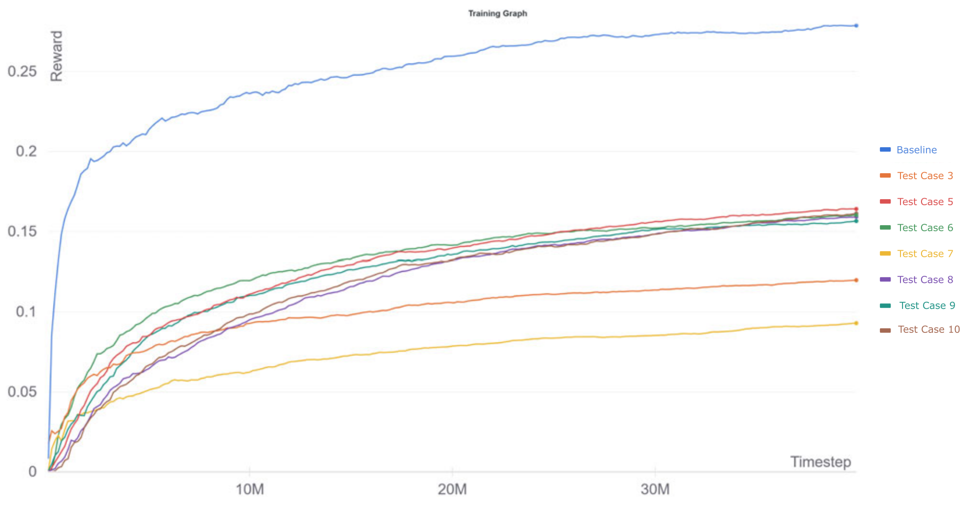

Figure 7. *Training history for test cases detailed in Table 4.*

of the experiments conducted. Table 4 describes each of the test cases, with the baseline included for comparison. Figure 7 shows the learning curves for these test cases. Similar to Fig. 5, the baseline models show the highest reward during training, due to a simpler training environment. With additional domain randomisation applied to the other test cases, there is a general decrease in reward per time step of training. Test Case 7 adds additional domain randomisation during training, yet without any of the learning improvements, resulting in the worst learning performance. The remaining test cases from Table 4 have learning improvements such as frame stacking and tuned parameters for improved learning performance. Some test cases are identical to the previous section, are given the same name, and are included for a complete comparison.

In Table 5, the baseline case obtains the lowest overall mean reward of 0.099, as well as the lowest reward at each evaluation headwind. With the lack of domain randomisation and with default training parameters and architecture, these models are least able to generalise to the evaluation environment detailed in Section 7. Using a larger neural network led to a decrease in mean evaluation reward in the previous set of results. However, in Table 5, Test Case 5 obtains a slightly higher mean reward of 0.178 compared to 0.176 for Test Case 10. Similarly, Test Case 9 obtains a reward of 0.175, whilst Test Case 8 receives a mean reward of 0.170. Test Cases 5 and 10 are identical except for network size. This relationship is also true for Test Cases 9 and 8. This suggests that having a larger neural network is advantageous to training performance for these more complex models. Table 5 shows that Test Case 5 still receives the highest overall reward of 0.178. Similar to Fig. 6, the inclusion of airspeed

Table 5. *Simulated mean reward for several test cases*

	Wind speed (m/s)					
Test Case	8	6	4	2	0	Mean
Baseline	0.000	0.009	0.077	0.202	0.205	**0.099**
Test Case 3	0.002	0.069	0.186	0.190	0.210	**0.131**
Test Case 5	0.003	0.094	0.243	0.307	0.244	**0.178**
Test Case 6	0.004	0.090	0.227	0.281	0.234	**0.167**
Test Case 7	0.001	0.053	0.135	0.162	0.212	**0.113**
Test Case 8	0.004	0.091	0.219	0.311	0.224	**0.170**
Test Case 9	0.003	0.090	0.222	0.316	0.244	**0.175**
Test Case 10	0.004	0.096	0.231	0.322	0.227	**0.176**

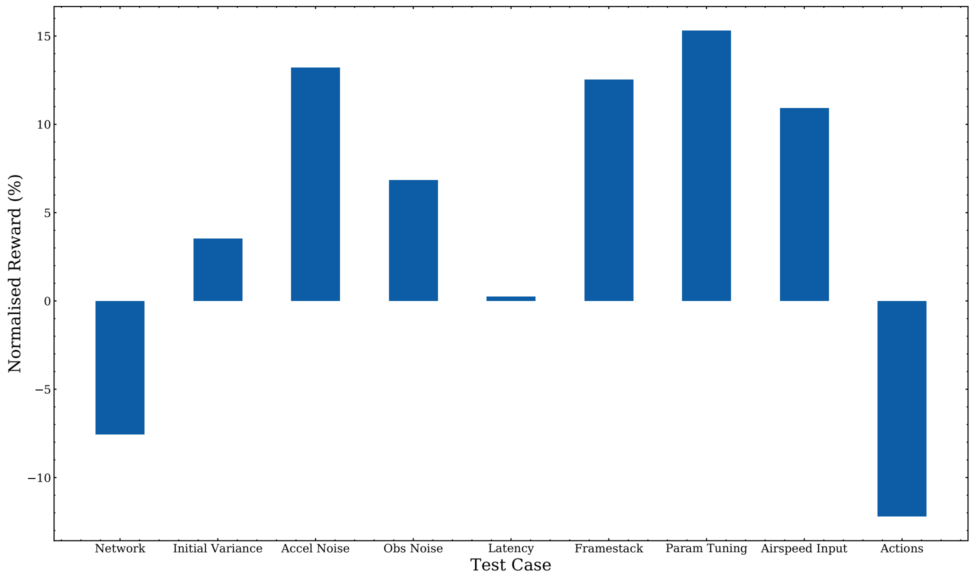

Figure 8. *Rewards for each test case, normalised against the baseline reward.*

in the observation results in a slightly higher mean reward. The difference in mean reward between Test Cases 3 and 5 emphasises the importance of the learning improvements, such as frame stacking and hyperparameter tuning.

7.2 Impact of individual improvements

The results presented so far suggest that the implemented improvements are of varying levels of importance. In the previous set of results, there is a small increase in reward when airspeed is added to the observation, and a more significant increase in reward is seen with the use of frame stacking and change of hyperparameters. Figure 8 shows the effect of adding each improvement individually to the baseline model. Five models are trained for each test case using the same methodology as previously and then evaluated against the enhanced evaluation scenario described previously. The plots show the percentage difference in mean reward for each test case compared to the baseline model. The data show that the most significant increase in mean reward comes from changing the hyperparameters. Even though only two hyperparameters have been adjusted, a significant increase in reward above the baseline is observed. It is likely that with a complete hyperparameter optimisation, an even greater increase in

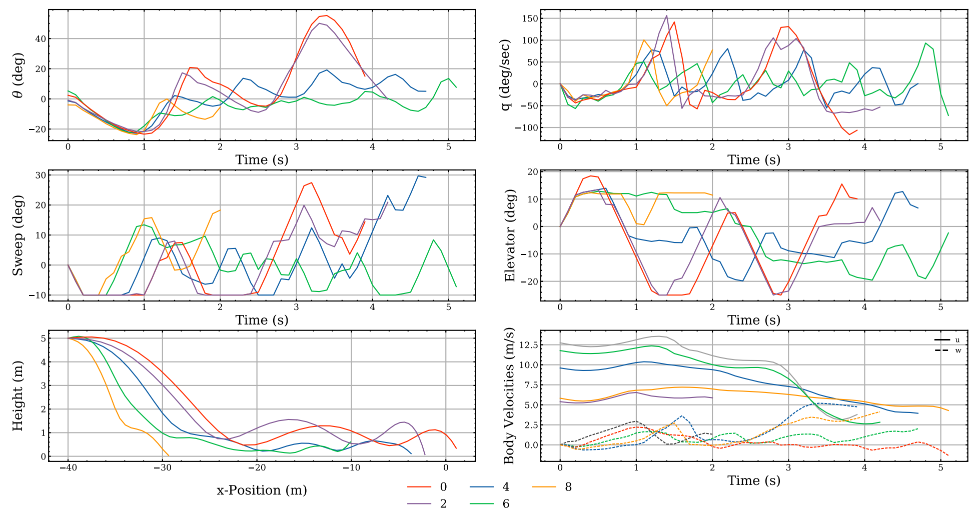

Figure 9. *Simulation time history results for a baseline model at different headwinds.*

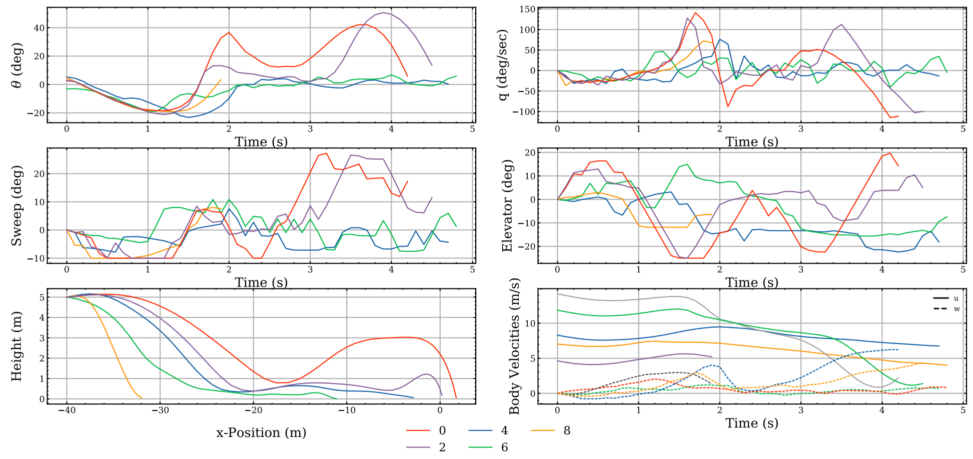

Figure 10. *Simulation time history results for a Test Case 5 model at different headwinds.*

training performance will be seen. The impact of adding acceleration noise in training is also significant, as models trained with only the acceleration noise modification show a considerable increase in reward when compared to the baseline. Using a larger neural network and using continuous actions both result in a decrease in reward. The model of latency used in this work has negligible impact, with only a slight increase in reward when included during training. It is likely that training with a more comprehensive characterisation of latency, including mechanical effects, would be required before significantly impacting performance.

Figures 9 and 10 show example time history plots during simulated evaluation. They represent the best performing model of the baseline and Test Case 5, respectively. Each model is evaluated against a series of steady-state winds, with all noise sources enabled. The plots are from a single evaluation with random seeds and so only provide example trajectories. There will be variance between evaluations with domain randomisation, so the plots do not accurately represent performance. Present in the plots are the key longitudinal states, such as pitch angle and body velocities, and the actions. The plots show the variance present and the effect of wind speed on performance. For example, when facing a headwind

Table 6. *Reward achieved by 2 different test cases during the first session of experimental flight testing*

	Baseline			Test Case 7		
Attempt	Flight 1	Flight 2	Flight 3	Flight 1	Flight 2	Flight 3
1	0.000	0.000	0.001	0.248	Fail	Fail
2	0.000	0.000	0.000	0.007	0.000	0.000
3	0.005	0.001	0.000	0.000	0.000	0.000
4	0.003	0.001	0.023	0.000	0.000	0.000
5	0.000	0.041	0.012	Fail	Fail	0.001
6	0.017	0.006	0.000	0.010	0.000	0.000
7	0.001	0.002	0.001	0.000	–	–
8	–	0.015	–	0.000	–	–

	Baseline	Test Case 7
Mean	0.006	0.011
SD	0.010	0.054

of 8m/s, both models fall well short of the target position. As the headwind decreases, the aircraft lands closer to its target position.

8.0 Experimental flight testing

Over two separate sessions, experimental flight testing was conducted to validate the simulation results. These flight tests use an automated testing process, as used and described in previous experimentation by the authors [7]. The autopilot flies pre-generated waypoints based on the prevailing wind direction. The waypoints are generated using a script to create a racetrack pattern, with parameters to adjust the size and location of the pattern. A heading parameter is used to specify a wind direction, with the pattern rotated such that the straight legs of the circuit are parallel to the wind direction.

At a designated waypoint, the autopilot hands over control of the elevator and wing sweep to the RL model running on the onboard computer. The ArduPilot firmware was modified to include a new waypoint type to specify the point at which the perching manoeuvre begins and the aircraft switches to RL control. An interface script transforms the state reported by the autopilot, such that the initial position and direction match those used for training; in this case, $x_e, z_e = (-40, -5)$.

When the aircraft passes through ground level in this transformed state, i.e. 5m below the RL start altitude, the RL controller disengages, and the autopilot attempts to recover the aircraft, fly to the next waypoint and repeat the pattern. The test pilot only has to launch and land the aircraft during optimal operation, with the autopilot performing the rest. However, the pilot retains manual override to take over control of the aircraft if necessary. The autopilot aims for an airspeed of 13m/s for the manoeuvre entry waypoint.

For each test case being evaluated during flight testing, the model that obtained the highest reward in simulation was deployed on the vehicle. The first set of tests compared the baseline models with Test Case 7 from Table 4, a model trained with domain randomisation, as described in Section 7. These tests were conducted on a day with inconsistent winds from variable directions, gusting up to approximately 5m/s. Table 6 shows the results from these first tests. During a flight, the aircraft is launched, performs several attempts at the manoeuvre, and then lands when the battery level is low. Due to inconsistent power usage, there is variance in the number of manoeuvre attempts per flight.

In Table 6, the Test Case 7 model obtains the higher overall mean reward of 0.011. compared to 0.006 for the baseline. However, the individual rewards and greater standard deviation suggest that this

Table 7. *Mean final states for a baseline model and a Test Case 7 model during the first flight testing session*

Test Case	x Position error (m)		Pitch error (deg)		Velocity error (m/s)	
	Mean	SD	Mean	SD	Mean	SD
Baseline	15.71	5.55	5.49	11.32	8.37	2.29
Test Case 7	18.95	5.40	35.80	33.23	6.27	3.47

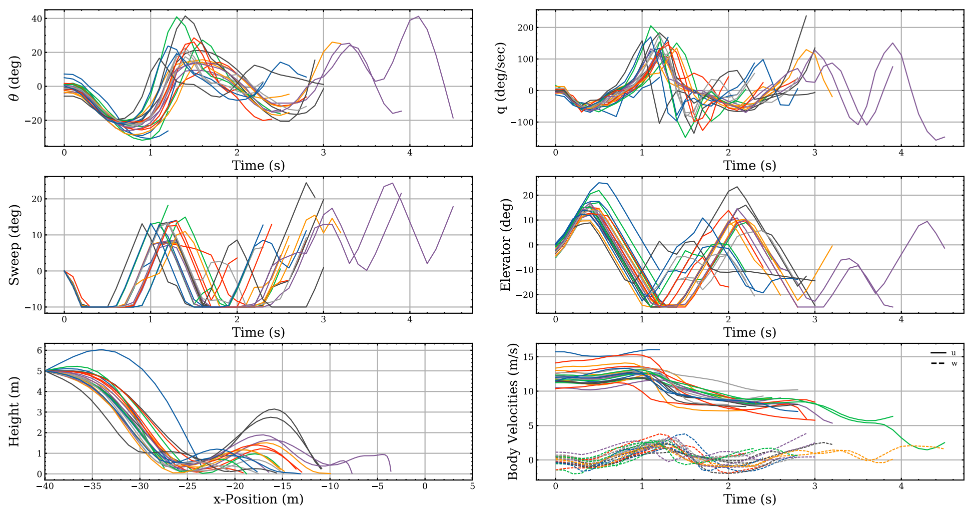

Figure 11. *Plot of states and actions for several attempts for the baseline model from the first experimental flight testing session.*

overall mean is skewed by a single attempt, with a reward of 0.248 obtained on the first attempt of flight 1. If this result is ignored, the mean for Test Case 7 drops to 0.001. This suggests that the baseline model performed better than Test Case 7 during these tests when this outlying result is ignored. This is supported by Table 7, which shows the mean final states across all attempts for both models. Table 7 shows that the baseline model has lower final state error for both position and pitch angle. However, the Test Case 7 model attains a lower final mean velocity.

Figures 11 and 12 show time history plots from the experiments, with each line corresponding to a manoeuvre attempt. The plots show how the key longitudinal states of the aircraft, as well as the surface deflections. Of particular interest is the difference between the chosen actions for the two models and the effect on the corresponding trajectories. Figure 11 shows that the policy for the baseline model opts to initially sweep the wings rearwards as it pitches the aircraft downwards into a dive, and then shifts them forward as it pitches up into a flare. For the remainder of each manoeuvre, both actuators are then used to attempt to maintain pitch close to zero degrees. Figure 12 demonstrates a different policy. The wing sweeps forward early in the manoeuvre, and the vehicle pitches up. For many of the attempts, the aircraft maintains maximum forward sweep for the manoeuvre duration. Many of the attempts fall short of the target position, falling vertically after losing lift, suggesting the aircraft has stalled. Therefore, it is likely that the underperformance of the Test Case 7 model in the real world is due to the reality gap. The early stalling behaviour and subsequent lower reward is not seen during simulation, suggesting there is underlying real-world aerodynamics not present in the numerical model. The behaviour shown by the baseline model may be more stable. This discrepancy is likely exacerbated by the aircraft's velocity, with the Test Case 7 model achieving a lower mean final velocity than the baseline.

Table 8. *Reward achieved by 3 different test cases during the second session of experimental flight testing*

	Baseline			Test Case 5	Test Case 9		
Attempt	Flight 1	Flight 2	Flight 3	Flight 1	Flight 1	Flight 2	Flight 3
1	0.001	0.000	0.000	0.000	0.221	0.000	0.000
2	0.000	0.000	0.006	0.000	0.023	0.001	0.000
3	0.000	0.000	Fail	0.000	0.000	0.000	0.000
4	0.000	0.005	0.000	0.000	0.000	0.000	0.013
5	0.000	0.000	0.000	0.017	0.197	0.002	0.269
6	0.174	0.000	0.000	0.036	0.005	0.199	0.000
7	0.000	0.006	0.000	0.234	0.017	0.000	0.001
8	0.000	0.005	0.000	0.000	0.082	–	0.001
	–	0.001		–	–	–	–

	Baseline	Test Case 5	Test Case 9
Mean	0.008	0.036	0.044
SD	0.034	0.076	0.084

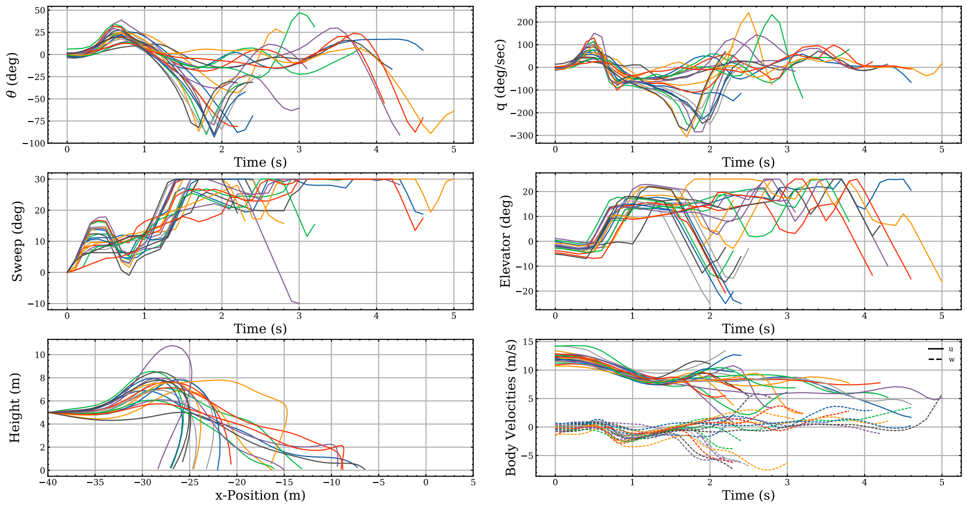

Figure 12. *Plot of states and actions for several attempts for the Test Case 7 model from the first experimental flight testing session.*

The second session of flight tests compared the performance of the baseline model against Test Cases 5 and 9. On this day, there were steady winds of approximately 5m/s gusting up to 8m/s, with consistent wind direction. Both Test Cases were trained with domain randomisation, increased network size, frame stacking and tuned hyperparameters. Both also used discrete actions, while only Test Case 5 has airspeed as part of its observation. These experiments used the same baseline model as the first testing session. Table 8 provides a summary of the rewards obtained for each manoeuvre attempt for each test case across a number of flights. Due to time constraints, only one set of flights for Test Case 5 could be obtained. The baseline model shows a slight increase from the first flight tests, with an increase in reward from 0.006 to 0.008. The Test Case 9 model achieves a greater mean reward of 0.044. Test Case 5 achieves a mean reward of 0.036; however, this is across only eight attempts.

Table 9 shows the average final states of the aircraft. Compared with Table 7, the baseline attempts show a greater position error of 20.78m, compared to 15.71m in the first flight tests. However, there is a

Table 9. *Mean final states for a baseline model, a Test Case 5 model and a Test Case 9 model during the second flight testing session*

	x Position error (m)		Pitch error (deg)		Velocity error (m/s)	
Model	Mean	SD	Mean	SD	Mean	SD
Baseline	20.78	8.58	1.82	14.62	7.17	2.83
Test Case 5	20.84	10.76	7.74	11.63	9.05	2.88
Test Case 9	14.61	6.09	5.72	22.78	6.02	2.53

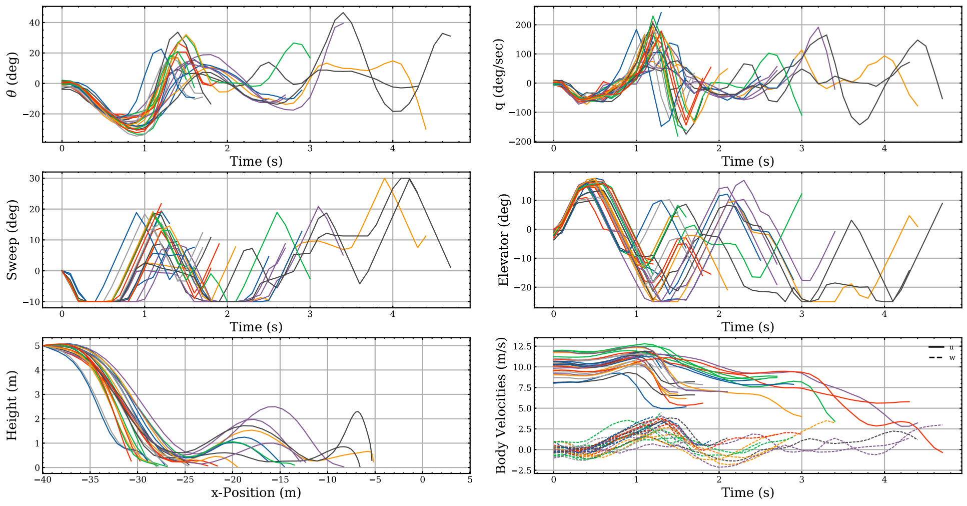

Figure 13. *Plot of states and actions for several attempts for the baseline model from the second experimental flight testing session.*

decrease in final pitch and velocity error, leading to the overall higher reward. Test Case 9 has reduced mean position and velocity errors of 14.61m and 6.02m/s respectively, yet does have a greater pitch error of 5.72 degrees. Test Case 5 generally showed the greatest mean error level across the three states, however this was with only eight attempts.

Figure 13 shows the state history for the baseline model. The plots show similar characteristics to Fig. 11, with the policy of the model selecting similar sequences of actions with similar resultant trajectories. Compared with Fig. 13, Fig. 14, showing the state histories for Test Case 9, shows that general trend of landing closer to the target position of 0m, as well as reduced body velocities. Like the baseline model, this policy opts to pitch downwards at the start of the manoeuvre and then flare later on for a perched landing. Figures 11 through 14 demonstrate the level of variance between attempts from the same model. Even for the best performing Test Case 9, shown in Fig. 14, there is significant variance in the selected actions and the resultant trajectories. This is likely due to the nature of the perched landing manoeuvre – the aircraft is highly sensitive to gusts once it has committed to a flare trajectory, especially when lacking throttle.

9.0 Conclusions

Using the PPO algorithm, reinforcement learning was used to generate a series of models to perform a perched landing manoeuvre. A series of training enhancements were identified to improve real-world

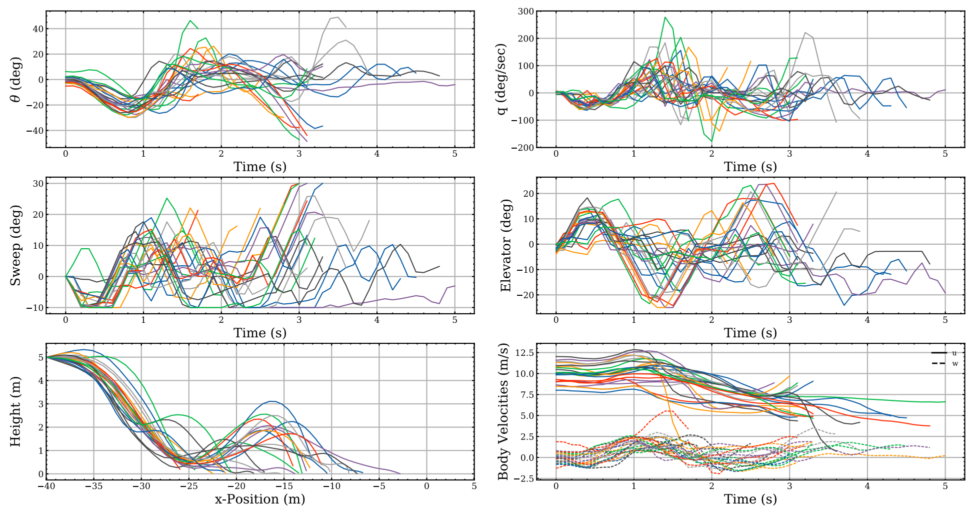

Figure 14. *Plot of states and actions for several attempts for the Test Case 9 model from the second experimental flight testing session.*

performance, measured primarily through the final reward achieved. These enhancements include passing additional information into the input observation, introducing further noise into the training through domain randomisation, and modifying the network architecture and underlying RL algorithm. A series of investigations were conducted, assessing the impact of each enhancement on the final reward, both individually and in combination. Simulations demonstrated that the most significant individual increase in reward came from tuning two of the key hyperparameters of the PPO algorithm. Other modifications, such as using a continuous action space instead of discrete, decreased the final reward. After testing several combinations, test cases with higher performance than the baseline model were identified for flight testing.

These trained models were deployed on the flight test aircraft. The results from several repeat attempts at the manoeuvre for each model were collected using an automated flight testing process. The first flight testing session compared a baseline model to a model trained with domain randomisation, Test Case 7. These flights demonstrated the limitations of the current approach, as the test case which performed better in simulation, performed worse in reality. This is likely due to the reality gap. The Test Case 7 model chooses a policy that works in simulation and receives a high reward yet leads to premature stall in reality. This suggests that either improvement of the numerical model through further data collection is required, or further domain randomisation of the aerodynamic parameters could be attempted.

The second session of flight testing compared a baseline model with models from Test Cases 5 and 9. These test cases were two of the best performing test cases from simulation, with their properties detailed in Table 4. Test Case 5 adds airspeed to the observation, whereas Test Case 9 uses the standard observation. Test Case 9 achieves higher mean rewards than the baseline model and generally achieves a lower final state error. Due to time constraints, only a limited number of Test Case 5 manoeuvre attempts could be collected. From the limited data set, Test Case 5 achieves a mean reward greater than the baseline yet lower than Test Case 9. However, the average final state errors are higher than the other two test cases. This suggests that further reward shaping may be required to achieve the desired perching behaviour across a range of real-world conditions.

Overall, enhanced models demonstrate improved performance compared to the baseline in both simulation and the real world. However, even the best performing model still lands somewhat short of the target position. It is likely that any further improvement in performance cannot come from changes to the RL process alone. One potential future pathway is to integrate throttle control into the RL controller. This would allow the aircraft the perform the manoeuvre when facing stronger headwinds instead of having insufficient energy and falling short, as is currently the case. Refinement and expansion of the underlying numerical model is another area of potential future work. The results suggest that, even with the domain randomisation performed in this work, there is still a significant reality gap. Further wind tunnel testing, real-world system identification methods, and model-based RL methods are future research paths that could improve simulation to reality transfer.

Acknowledgements. This work was part-funded by the Defence Science and Technology Laboratory (DSTL), Ministry of Defence. This work was partially supported by the EPSRC Centre for Doctoral Training in Future Autonomous and Robotic Systems (FARSCOPE) at the Bristol Robotics Laboratory. The authors would like to thank all contributors to ArduPilot [12], and the `pymavlink` and `stable-baselines` [13] libraries, all of which have enabled this work. This work was carried out using the computational facilities of the Advanced Computing Research Centre, University of Bristol – http://www.bris.ac.uk/acrc/

References

[1] Waldock, A., Greatwood, C., Salama, F. and Richardson, T. Learning to perform a perched landing on the ground using deep reinforcement learning, J. Intell. Rob. Syst. Theory Appl., 2018, **92**, pp 685–704.
[2] Meckstroth, C.M. and Reich, G.W. Aerodynamic modeling of small UAV for perching experiments, 31st AIAA Applied Aerodynamics Conference, 2013.
[3] Moore, J., Cory, R. and Tedrake, R. Robust post-stall perching with a simple fixed-wing glider using LQR-Trees, *Bioinspiration Biomimetics*, 2014, **9**, (2), p 025013.
[4] Novati, G., Mahadevan, L. and Koumoutsakos, P. Controlled gliding and perching through deep-reinforcement-learning, *Phys. Rev. Fluids*, 2019, **4**, (9), p 093902.
[5] Greatwood, C., Waldock, A. and Richardson, T. Perched landing manoeuvres with a variable sweep wing UAV, *Aerospace Sci. Technol.*, 2017, **71**, pp 510–520.
[6] Clarke, R.J., Fletcher, L., Greatwood, C., Waldock, A. and Richardson, T.S. Closed-loop Q-learning control of a small unmanned aircraft, AIAA Scitech 2020 *Forum* (Reston, Virginia), American Institute of Aeronautics and Astronautics, 2020.
[7] Fletcher, L., Clarke, R., Richardson, T. and Hansen, M. Reinforcement learning for a perched landing in the presence of wind, AIAA Scitech 2021 Forum, 2021.
[8] OpenAI, Berner, C., Brockman, G., Chan, B., Cheung, V., Dębiak, P., Dennison, C., Farhi, D., Fischer, Q., Hashme, S., Hesse, C., JÓzefowicz, R., Gray, S., Olsson, C., Pachocki, J., Petrov, M., Pinto, H.P.d.O., Raiman, J., Salimans, T., Schlatter, J., Schneider, J., Sidor, S., Sutskever, I., Tang, J., Wolski, F. and Zhang, S. Dota 2 with Large Scale Deep Reinforcement Learning, Tech. rep., OpenAI, 2019.
[9] Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J. and Zaremba, W. OpenAI Gym, ArXiv, 2016.
[10] Koch, W., Mancuso, R., West, R. and Bestavros, A. Reinforcement learning for UAV attitude control, *ACM Trans. Cyber-Phys. Syst.*, 2019, **3**, (2), pp 1–21.
[11] Bohn, E., Coates, E.M., Moe, S. and Johansen, T.A. Deep reinforcement learning attitude control of fixed-wing UAVs using proximal policy optimization, 2019 International Conference on Unmanned Aircraft Systems, ICUAS 2019, Institute of Electrical and Electronics Engineers Inc., 2019, pp 523–533.
[12] Tridgell, A., Ferreira, F., Morphett, G., Walser, J., De Marchi, L., du Breuil, M., Barker, P., Mackay, R., Pittenger, T., Geyer, B., Olson, C., Castelnuovo, E., Shamaev, E., Staroselskii, G., de Sousa, G., Beraud, J., Hall, L., Lawrence, M., Badaire, M., Denecke, M., Riseborough, P., Kancir, P., Mayoral Vilches, V. and Lucas, A. ArduPilot, https://github.com/ArduPilot/ardupilot, 2020.
[13] Hill, A., Raffin, A., Ernestus, M., Gleave, A., Traore, R., Dhariwal, P., Hesse, C., Klimov, O., Nichol, A., Plappert, M., Radford, A., Schulman, J., Sidor, S. and Wu, Y. Stable Baselines, https://github.com/hill-a/stable-baselines, 2018.
[14] Moerland, T.M., Broekens, J., Plaat, A. and Jonker, C.M. Model-based Reinforcement Learning: A Survey, arXiv, 2020.
[15] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S. and Hassabis, D. Human-level control through deep reinforcement learning, *Nature*, 2015, **518**, (7540), pp 529–533.
[16] Schulman, J., Wolski, F., Dhariwal, P., Radford, A. and Klimov, O. Proximal policy optimization algorithms, arXiv, 2017.
[17] Andrychowicz, O.M., Baker, B., Chociej, M., JÓzefowicz, R., McGrew, B., Pachocki, J., Petron, A., Plappert, M., Powell, G., Ray, A., Schneider, J., Sidor, S., Tobin, J., Welinder, P., Weng, L. and Zaremba, W. Learning dexterous in-hand manipulation, *Int. J. Rob. Res.*, 2020, **39**, (1), pp 3–20.
[18] Tan, J., Zhang, T., Coumans, E., Iscen, A., Bai, Y., Hafner, D., Bohez, S. and Vanhoucke, V. Sim-to-Real: learning agile locomotion for quadruped robots, Robotics: Science and Systems XIV, Robotics: Science and Systems Foundation, 2018.
[19] Schulman, J., Levine, S., Moritz, P., Jordan, M. and Abbeel, P. Trust region policy optimization, 32nd International Conference on Machine Learning, ICML 2015, vol. 3, 2015, pp 1889–1897.
[20] Corke, T.C. and Thomas, F.O. Dynamic stall in pitching airfoils: aerodynamic damping and compressibility effects, *Annu. Rev. Fluid Mech*, **47**, 2015, pp 479–505.
[21] Peng, X.B., Andrychowicz, M., Zaremba, W. and Abbeel, P. Sim-to-real transfer of robotic control with dynamics randomization, Proceedings - IEEE International Conference on Robotics and Automation, 2018, pp 3803–3810.

[22] Ibarz, J., Tan, J., Finn, C., Kalakrishnan, M., Pastor, P. and Levine, S. How to train your robot with deep reinforcement learning: lessons we have learned, *Int. J. Rob. Res.*, 2021, **40**, (4–5), pp 698–721.
[23] Beard, R. and McLain, T.W. *Small Unmanned Aircraft: Theory and Practice*, vol. 1, 2012, Princeton University Press.
[24] Langelaan, J.W., Alley, N. and Neidhoefer, J. Wind field estimation for small unmanned aerial vehicles, *J. Guidance Control Dyn.*, 2011, **34**, (4), pp 1016–1030.
[25] Henderson, P., Islam, R., Bachman, P., Pineau, J., Precup, D. and Meger, D. Deep reinforcement learning that matters, *32nd AAAI Conference on Artificial Intelligence*, AAAI 2018, **34**, (6), pp 3207–3214.
[26] Biewald, L. Experiment Tracking with Weights and Biases, 2020.
[27] Raffin, A. RL Baselines Zoo, https://github.com/araffin/rl-baselines-zoo, 2018.
[28] Raffin, A., Kober, J. and Stulp, F. Smooth exploration for robotic reinforcement learning, Proceedings of the 5th Conference on Robot Learning, PMLR, 2020.
[29] Haarnoja, T., Zhou, A., Hartikainen, K., Tucker, G., Ha, S., Tan, J., Kumar, V., Zhu, H., Gupta, A., Abbeel, P. and Levine, S. Soft Actor-Critic Algorithms and Applications, arXiv preprint arXiv:1812.05905, 2018.
[30] Xiao, T., Jang, E., Kalashnikov, D., Levine, S., Ibarz, J., Hausman, K. and Herzog, A. Thinking while moving: deep reinforcement learning with concurrent control, International Conference on Learning Representations, 2020.

Cite this article: Fletcher L., Clarke R., Richardson T. and Hansen M. (2022). Improvements in learning to control perched landings. *The Aeronautical Journal*, **126**, 1101–1123. https://doi.org/10.1017/aer.2022.48