Journal of Intelligent & Robotic Systems (2022) 104: 32
https://doi.org/10.1007/s10846-022-01577-5

REGULAR PAPER

Autonomous Learning in a Pseudo-Episodic Physical Environment

Kevin P. T. Haughn[1] · Daniel J. Inman[1]

Received: 26 May 2021 / Accepted: 13 January 2022 / Published online: 8 February 2022

Abstract
For practical considerations reinforcement learning has proven to be a difficult task outside of simulation when applied to a physical experiment. Here we derive an optional approach to model free reinforcement learning, achieved entirely online, through careful experimental design and algorithmic decision making. We design a reinforcement learning scheme to implement traditionally episodic algorithms for an unstable 1-dimensional mechanical environment. The training scheme is completely autonomous, requiring no human to be present throughout the learning process. We show that the pseudo-episodic technique allows for additional learning updates with off-policy actor-critic and experience replay methods. We show that including these additional updates between periods of traditional training episodes can improve speed and consistency of learning. Furthermore, we validate the procedure in experimental hardware. In the physical environment, several algorithm variants learned rapidly, each surpassing baseline maximum reward. The algorithms in this research are model free and use only information obtained by an onboard sensor during training.

Keywords Autonomous systems · Experience replay · Model-free · Reinforcement learning · Robot learning

1 Introduction

Reinforcement learning (RL) has gained significant attention in the field of artificial intelligence over the past several years. Although reward-based learning algorithms are not a new concept, recent advancements in hardware and increased data accessibility have made it possible to achieve accurate estimation of nonlinear models using large quantities of data [4, 26, 28]. Artificial neural networks (ANN), deep ANNs specifically, are frequently the preferred method of function approximation in supervised learning tasks and have gained popularity in the reinforcement learning community. There is now a subset of RL dedicated to the use of multilayered ANNs called deep reinforcement learning (DRL). This is not without merit, for they have led to superhuman performance in Atari game environments, Dota 2, and Go, as well as many simulated physics environments offered by the OpenAI libraries and MuJoCo to name a few [6, 27, 28, 35, 36]. While this is impressive, and certainly beneficial for the reinforcement learning community, it is necessary to build from what we've learned using simulated environments, where data accumulation is plentiful and instrumental wear and tear plays no role, and apply that knowledge to real-world environments and robotics.

While there has been substantial success in the simulated DRL environments, the data-hungry nature of ANNs has led to solutions geared towards artificially increasing the amount of data available to these nonlinear approximation functions. Many examples in the literature use transfer learning techniques to train an agent in a simulated environment and then load the pre-trained policy onto the physical agent of a real-world environment [2, 40, 41]. Other examples use imitation learning, manually guiding the agent through the motions desired for acceptable performance [21, 29]. Both methods are popular ways to limit the time spent training on hardware, and reduce wear and tear on equipment; however, not all environments lend themselves easily to accurate simulation. In many cases, manual examples may not be desirable, or even possible, to produce.

In this research we aim to show an option for model-free reinforcement learning that can be achieved autonomously and entirely online. In this work we consider training time and equipment safety through careful experimental design and algorithm decision making. Due to the limited availability of data and the computationally expensive

✉ Kevin P. T. Haughn
kevpatha@umich.edu

Daniel J. Inman
daninmana@umich.edu

[1] The University of Michigan Department of Aerospace Engineering, 1320 Beal Ave, Ann Arbor, MI 48109, USA

nature of ANNs, we chose to forgo the use of these nonlinear function approximators, opting instead for linear function approximation. Although ANNs were not used in this work, we leverage some techniques that found success in DRL. We chose to design our experiment to imitate the episodic style of many simulated reinforcement learning problems while maintaining strict autonomy of the agent. Additionally, we take advantage of our pseudo-episodic format to create additional opportunities for learning, where traditionally there are none, by using time and actions taken during the preparation of subsequent training episodes to perform policy updates. The rest of this paper is organized in the following manner. Section 2 provides background information and related work in reinforcement learning, the algorithms used for our research, and methods for overcoming obstacles faced in real-world RL. Section 3 presents the theory and methods we followed to achieve learning and preliminary simulation work used to determine algorithm viability and hyperparameter values. Section 4 describes the experimental setup used to demonstrate and validate our training scheme a real-world environment. In Section 5 we discuss the results of our experiments. The conclusions and future work are found in Section 6.

2 Background and Related Work

This section begins by introducing background information including basic concepts in reinforcement learning and the three baseline algorithms used in this research. The following two parts of this section present relevant research, including methods for reinforcement learning outside of simulation and their application to environments similar to our own, to provide context for where our contribution fits into the current state of the art.

2.1 Reinforcement Learning

Reinforcement learning is a subset of machine learning (ML) that, through trial and error and the use of a reward system, is capable of autonomously developing controllers for agents in a variety of environments. Agents and environments are the two fundamental pieces of any RL problem, where the agent is the subject of policy-determined actions, and the environment is the defined space within which the agent may act. The RL problem is traditionally further defined by a Markov decision process (MDP), consisting of every possible state-action combination, and their state outcomes, for the agent within the environment. Depending on the goal of the agent, certain state-action pairs may be preferable over others. The preferability of a state is defined by its *value*, and is determined by calculating the long-term reward obtained by the agent, from the environment, after residing in the specified state. In many algorithms the state *values* are used to determine the agent's policy, which is a map deciding which actions are to be taken given the current state. In the simplest cases, *values* can be recorded in a table; however, when environments become more complicated, requiring a continuous state space, they must be approximated. In recent years, ANNs have been the cutting-edge function approximation method of choice for many in the field of RL, and for good reason. Given the correct structure and enough data, ANNs are incredibly proficient at accurately approximating nonlinear functions. The success of DRL has brought a surge in popularity for the RL community, resulting in new algorithms and techniques to improve speed of learning and final overall performance. This is particularly true for traditional baseline environments, including Atari games and MuJoCo physics simulators for higher dimensional problems [16, 34, 43].

The reinforcement learning methods used in this research fall under a category called temporal difference (TD) learning. TD learning allows for a policy development from experiences gathered directly from agent-environment interaction. This presents an opportunity for training to be model free, meaning there is no need to approximate any dynamics of the environment to achieve control. Additionally, bootstrapping is used, which applies previously learned approximations to update current estimations so that learning takes place after every timestep [39]. This work focuses only on policy gradient methods, specifically actor-critic based methods, to achieve learning. Actor-critic methods consist of two function approximators, an actor and a critic. The actor is a parameterized policy function and uses a soft-max distribution to represent each action as a probability. The critic is a parameterized value function, akin to the Q-functions used in many action-value methods, and behaves as a judge for the actor's decisions by estimating the expected total reward, or *value*, of any given state in the available state space. Actor and critic weight updates, $\varDelta\boldsymbol{\theta}$ and $\varDelta\mathbf{w}$, are performed with gradient decent, using

$$\varDelta\mathbf{w} = \alpha\delta\nabla\hat{\mathbf{v}}(S, \mathbf{w}), \tag{1}$$

and

$$\varDelta\boldsymbol{\theta} = \alpha\delta\nabla\pi_\theta(A|S, \boldsymbol{\theta}), \tag{2}$$

where α is the learning rate. $\hat{\mathbf{v}}(S, \mathbf{w})$ is the *value* of the current state, S, given the critic weights, $\mathbf{w}$. For linear function approximation, the gradient of $\hat{\mathbf{v}}(S, \mathbf{w})$ is equal to the state features, $\mathbf{x}$(S, $\mathbf{w}$), for which we used 3rd order Fourier cosine basis functions [23]. For updating the actor's weights, $\boldsymbol{\theta}$, we use the natural log gradient of the trained policy, π_θ, calculated as

$$\nabla\mathbf{ln}\pi_\theta(A|S, \boldsymbol{\theta}) = \mathbf{x}(S, A) - \sum\nolimits_b \pi_\theta(b|S, \boldsymbol{\theta})\mathbf{x}(S, b), \tag{3}$$

where A is the current action. To perform both of these updates we must first calculate the temporal difference error, δ, found as

$$\delta = R + \gamma \hat{\mathbf{v}}(S', \mathbf{w}) - \hat{\mathbf{v}}(S, \mathbf{w}), \tag{4}$$

the difference between the current *value* and the combination of the expected *value* of the next state, S', and the newly obtained reward, R. In this work, the discount factor, γ, is equal to one. The algorithms of choice are Advantage Actor-Critic (A2C), Actor-Critic with Eligibility Traces (A2C(λ)), and Proximal Policy Optimization (PPO) [34, 39]. Eligibility traces are a means to use information gathered from each time step and propagate it through parameters updated in previous time steps, according to the contribution of the previous states. The update equations for the actor and critic including eligibility traces are

$$\mathbf{z}^{\mathbf{w}} \leftarrow \gamma \lambda^{\mathbf{w}} \mathbf{z}^{\mathbf{w}} + \nabla \hat{\mathbf{v}}(S, \mathbf{w}), \tag{5}$$

$$\mathbf{z}^{\theta} \leftarrow \gamma \lambda^{\theta} \mathbf{z}^{\theta} + \nabla \mathbf{ln} \pi_{\theta}(A|S, \boldsymbol{\theta}), \tag{6}$$

$$\Delta \mathbf{w} = \alpha \delta \mathbf{z}^{\mathbf{w}}, \tag{7}$$

and

$$\delta \boldsymbol{\theta} = \alpha \delta \mathbf{z}^{\theta}. \tag{8}$$

When calculating the eligibility trace vectors for the actor and critic, $\mathbf{z}^{\theta}$ and $\mathbf{z}^{\mathbf{w}}$ respectively, trace decay rates, λ^{θ} and $\lambda^{\mathbf{w}}$, are used to adjust an update's impact on previous states. PPO is one of the current leading on-policy actor-critic algorithms, showing impressive performance in several high dimensional MuJoCo physics environments. This performance improvement was achieved through the introduction of clipping. Clipping is a simple implementation for an idea akin to trust region policy optimization (TRPO), meant to limit the size of policy updates and mitigate overshooting [33, 34]. This is done by looking at the ratio, r, between an action's probability under the new and old policies, and limits that ratio to fall within $1 \pm \epsilon$, where ϵ is the clipping parameter. We used the suggested value $\epsilon = 0.2$. This relationship is described by the objective function,

$$\mathbb{L}^{CLIP}(\boldsymbol{\theta}) = \hat{\mathbb{E}}[\mathbf{min}(r(\boldsymbol{\theta})\delta, clip(r(\boldsymbol{\theta}), 1-\epsilon, 1+\epsilon)\delta]. \tag{9}$$

2.2 RL in the Physical World

Learning in real-world environments, such as robotics and vehicles, emphasizes different challenges in the reinforcement learning problem [11]. This research will focus primarily on safety constraints and learning from limited opportunities for data collection. Due to the innate cost of completing training in the physical environment, including wear and tear on equipment from extended use and time necessary to accumulate experience required for learning, some researchers choose to take their learning offline, using batch updates or simulated environments [13, 14, 24, 41]. When accurate simulations aren't easily attainable, and learning must be performed online, sample efficiency becomes very important. Improving sample efficiency is a common research problem in RL. Many have adopted creative methods to incorporate models during training to accelerate learning [3, 5, 8]. Others have delved into the field of meta RL, where the goal of an algorithm is to learn how to learn more quickly [12].

Reinforcement learning is a powerful tool to solve complex problems; however, these problems often take hundreds of thousands or millions of iterations to solve. Because of this, the push for autonomy in RL is highly preferable. Instead of human intervention, some RL problems in the literature are performed in environments with already well established means of control, and the goal of the RL algorithm is to discover a more optimal policy [38]. This can provide a safety net for the agent for cases where exploration approaches a known portion of the state space prone to damaging equipment, and allows for simple episode reset. However, this technique is limited to well understood environments. Haarnoja et al. use Soft Actor-Critic to train a four-legged robot to locomote over a variety of walking surfaces [15]. This was achieved model free and without simulation, but human intervention was necessary for resetting the environment when the robot fell over or wandered too far from the initial state. Zhu et al. pushed towards true autonomy in RL, abandoning the use of resetting mechanisms and relying solely on the robots' own sensors for state observation and reward signaling [44]. In this research we aim to continue this push toward autonomous RL, adhering to the constraints regarding sensing and reward systems, but do so in a pseudo-episodic manner for an environment that is not inherently stable.

2.3 RL in Aerial Environments

One such inherently unstable environment is UAV flight. UAV flight and maneuverability provide interesting control problems, particularly when operating in variable environments where adaptability offers a great benefit. Although our experiment doesn't truly fly and has relatively simple dynamics, it offers a step toward autonomous learning on an aerial system by achieving propeller driven control in an unstable environment. We are not the first to aspire toward this objective. The ability to autonomously adapt to aerial environments is an exciting goal that many in the reinforcement learning community are pursuing. The literature has many examples where control is achieved in simulated flight environments. Bohn and his colleagues achieve attitude control competitive with traditional proportional integral derivative (PID) methods using PPO for a simulated fixed wing UAV [7]. Koch et al. achieved similar results, outperforming PID methods when using PPO to train an open-source simulation for quadrotor flight [22]. In addition to attitude control, navigating a simulated environment is another challenge frequently accomplished with RL methods [18, 19]. Each of these cases shows the power of RL in simulation, but impressive work has been achieved in the real world as well. In 2003, Ng et al. used reinforcement learning to achieve autonomous control for a helicopter [30]. More recently, a trained quadrotor was capable of recovering from complex initial conditions, including an upside-down position [17]. However, in both cases a model or simulation was first developed for training, and no training was performed in real-time on the physical equipment. In 2018, navigation of an unknown environment was achieved in by a Parrot AR quadrotor drone; however, PID was used to aid in control of the UAV and, due to the constraint of battery life, human intervention was allowed for learning to continue after a UAV failure [31].

This previous work is impressive and has advanced the field of autonomous flight through reinforcement learning. The research presented here is meant to build from that foundation toward greater autonomy in training, requiring no human intervention, and doing so with only the experience gained by the agent while training in the physical environment. In other words, *we chose not to take advantage of policy optimization that took place in simulation nor a model of the environment's dynamics*. This is achieved through making novel decisions in experimental and algorithmic design. To the best of the authors' knowledge, this is the first example of combining two temporal difference learning algorithms to alternate on-policy and off-policy updates between episodes to accelerate learning.

3 Theory and Simulation

In this section we cover the theory behind the decisions made and methods used in order to achieve learning in our air-sled environment. We first describe our approach to achieve sufficient exploration while considering the safety of our equipment. Subsequently we present the additional updates we chose to implement during the "exploration episodes" that are at the heart of the pseudo-episodic training method. Although this method is designed to perform in physical environments, we investigated its viability in simulation to reduce unnecessary wear and tear on experimental hardware. The simulated training was not used to accelerate real-world learning, but is described at the end of this section for completeness.

3.1 Exploration and Safety

Exploration is essential for achieving optimal control through reinforcement learning; however, it often comes at the expense of compromised safety when operating in the real world. Because of this, methods are often implemented to limit the exploration necessary to achieve sufficient learning, including transfer learning and imitation learning, as mentioned earlier. For the purposes of this research, with the goal of fully autonomous learning, we chose to use only online and model-free temporal difference learning methods; therefore, any exploration of the environment must be achieved by the physical agent itself.

In this work, we achieved exploration by means that are often successful in simulated environments, including randomized initial positions, negative rewards, and epsilon-greedy. The latter two are easily implemented in the physical environment as well as in simulation. The use of negative rewards facilitates exploration when the adjustable weights of our approximation function are initialized to zero, presetting the values for each state-action combination to zero. When rewards are negative, a value of zero is only achieved when the goal is met perfectly, which is impossible to reach for practical purposes. At this point, any action taken in any state will result in a reward less than what is expected, resulting in different future action selections when following the learned policy. This method is known as *optimistic initial values*, and is particularly useful early in learning and for stationary tasks specifically [39]. To achieve this technique, we assign rewards as the negative squared distance between the current position and the target. Another common technique used to facilitate the exploration of the action space is epsilon-greedy [39]. At the time of action selection there is a set probability, ε, that a random action is selected. For our case the ε value of choice was 0.1, prompting for random action selection approximately 10% of the time. The other 90% of actions selected are considered "greedy" and follow the agent's trained policy, π_θ, for achieving highest expected reward. Performing RL in simulated environments lends itself easily to an episodic format that comes with built-in exploration. At the conclusion of each individual training episode, the environment may be reset with any initial conditions within the state space, allowing for the potential of a previously unseen, or rarely seen, state to be visited. By randomizing this initial position, the agent may explore the state space in a relatively uniform manner. Unfortunately, selecting an initial state for an agent that has yet learned to control itself can be difficult. Additionally, for the sake of autonomy, setting this initial state via human intervention is not an option. To overcome this, we chose to alternate training episodes with what we called "exploration episodes," in which the agent selects actions randomly from a modified action space for half the number of steps in a training episode. This allowed each training episode to begin in a pseudo-randomized state; however, we found that, due to a slight bias in the action space, the agent would end the exploration episode predominantly in a portion of the state space below the target position. To enable more complete exploration of the state space, we split the exploration action space into two modified spaces, one containing the six lowest motor outputs and another containing the six highest. By alternating between these two action spaces for each exploration episode we ensured initial states occurring both above and below the target position.

While this method allowed for ample exploration of the state space, choosing random actions for extended series of consecutive timesteps is potentially hazardous to the equipment, running the risk of relatively high-speed collision between the air-sled and the end of the air-track. To account for this, we implemented several safety precautions, one of which is achieved during action selection. If the air-sled comes within a specified distance of either end of the air-track, 5 centimeters, and does not have a velocity of at least 0.1 m/s toward the center of the air-track, then the maximum or minimum action is selected appropriately and automatically to accelerate the air-sled back toward the safe exploration zone.

3.2 Exploration Updates

Implementing this method of alternating learning episodes with exploration episodes as a means of randomized initial position has benefits for exploring the state space but is accomplished at the cost of time. The period of each exploration episode occurs without updating any weights for the actor or critic, yet the equipment still experiences the wear and tear of additional use and fuel or energy consumption. This time, although not wasted, could be spent more efficiently by including additional updates. The two methods studied in this work are using off-policy updates and applying an experience replay (ER) during the exploration episodes.

Off-Policy Actor-Critic (Off-PAC) is an algorithm introduced by Thomas Degris et al. for updating both the actor and critic from actions made using a decision process other than the trained policy [10]. For our case, these updates are performed using the randomized actions selected during exploration episodes. As with any off-policy algorithm, an importance sampling ratio is included to estimate the expectation of the trained policy when given a sample from the acting policy. Off-PAC traditionally uses an eligibility trace format to perform updates; however, for the purposes of this research, we chose to set the trace decay rates to zero so that the updates were equivalent to that of a one-step TD algorithm. We chose to do this because our goal is not to further complicate traditional learning algorithms with additional parameters in need of tuning, but to illustrate the advantage of exploiting time otherwise ignored during autonomous training in a physical experiment.

Experience replay is another tool commonly used in RL to increase the speed of learning and mitigate catastrophic forgetting in deep learning by keeping a memory of previous quadruples, (S, A, S', R), each containing a state and action with the following state and corresponding reward earned [20, 25, 32]. More recently, this technique has been applied to actor-critic methods and has become incredibly popular for deep reinforcement learning methods due to its ability to provide series of uncorrelated data for batch updates of neural networks [1, 42]. In our case we built a mini-batch that can hold 5 episodes' worth of the most recent quadruples, from which they are selected randomly in sets of 10 for each timestep of an exploration episode.

Algorithm 1: Exploration with Off-PAC or Experience Replay.

```
Input: Off-PAC ← $Boolean$
Input: REPLAY ← $Boolean$
initialization;
$NE$ ← Number of episodes;
$NS$ ← Number of steps per episode;
$NR$ ← Number of updates from replay buffer;
for iteration=0,1,...,NE do
    if iteration is even then
        for step=0,1,...,NS do
            Follow policy $\pi_\theta$;
            Update $\boldsymbol{\theta}$ and $\mathbf{w}$;
            if REPLAY is True then
                Save (S,A,S',R) to replay buffer;
            end
        end
    else
        for step=0,1,...,NS/2 do
            Follow random action policy if Off-PAC is True then
                Update $\boldsymbol{\theta}$ and $\mathbf{w}$ using Off-PAC;
            end
            else if REPLAY is True then
                for replays=1,2,...,NR do
                    Sample (S,A,S',R) from replay buffer;
                    Update $\boldsymbol{\theta}$;
                end
            end
        end
    end
end
```

The quadruples contain enough information to calculate a current gradient and perform an update similar to that which we performed when using Off-PAC, but with an importance sampling ratio of one since actions were selected following the trained policy. This allowed for the agent to continue to learn from past experiences when updates would otherwise not occur. The implementations of these methods are further illustrated in Algorithm 1.

3.3 Simulations

Due to wear and tear experienced by equipment from regular use, it is preferable to limit time spent tuning and training in the physical environment. Because of this, we decided to perform preliminary tests in a simulated environment, where many repetitions of training can be used for tuning hyperparameters and testing algorithm designs, without the consequence of equipment damage. The environment that we are simulating is depicted in Fig. 1. For this air-sled/air-track environment, the target position for the air-sled is 0.6 meters from the end of the air-track. The state space is continuous and two dimensional, capturing all position values within the range of 0 and 1.2 meters and velocities falling between -1 and 1 meters per second. Set at a 10-degree incline, the sled has an analog DC propeller motor that must produce a force output for position control. The incline renders the open loop system unstable. The physics of the air-track/air-sled environment are captured by these iterative kinematic equations:

$$x_{k+1} = x_k + \frac{T_k - G}{2}\Delta t^2 + v_k \Delta t \tag{10}$$

and

$$v_{k+1} = v_k + (T_k - G)\Delta t. \tag{11}$$

Here G is the acceleration due to gravity, which in this case is 0.52 m/s2 to account for the air-track incline,T_k is the acceleration from thrust, Δt is the simulation timestep size of 0.05 seconds, x_k is the position at timestep k, and v_k is the velocity at time step k. An air-sled mass of 0.170 kg is used to determine T_k. These equations are used to determine the air-sled's state (position and velocity) over the 400 timesteps of each training and testing episode as well as the 200 timesteps of each exploration episode.

This research aims to achieve learning in a real-world environment; thus, it is important to create as accurate a simulation as possible. Incorporating the noise produced by the propeller thrust output is crucial to accurately model the uncertainty experienced by the controller. Modeling the propeller thrust noise is achieved by first measuring samples of the output force produced by the propeller at a series of motor values. We chose to make force measurements for ten different motor values ranging from 25 to 250. These measurements were taken at 1000 Hz for 5 seconds each, resulting in 5000 force output readings for each determined motor value. These samples created a series of non-Gaussian distributions best represented by the triangular distribution function in python's "scipy" statistics library. Due to the 0.05 second timestep, a force output for a given motor value was determined by averaging fifty samples from the respective triangular distribution. Following the central limit theorem, this leads to Gaussian distributions of force outputs for each motor value. These average acceleration values, calculated using the air-sled's mass, and their standard deviations are shown in Fig. 2. As illustrated, 90% of the average acceleration from the propeller is achieved by a motor value of 100, after which the increase in propeller force diminishes. Because of this we chose to limit the discrete action space to 11 evenly spaced motor values ranging from 0 to 100. The average motor values not covered by the original data set were estimated with the

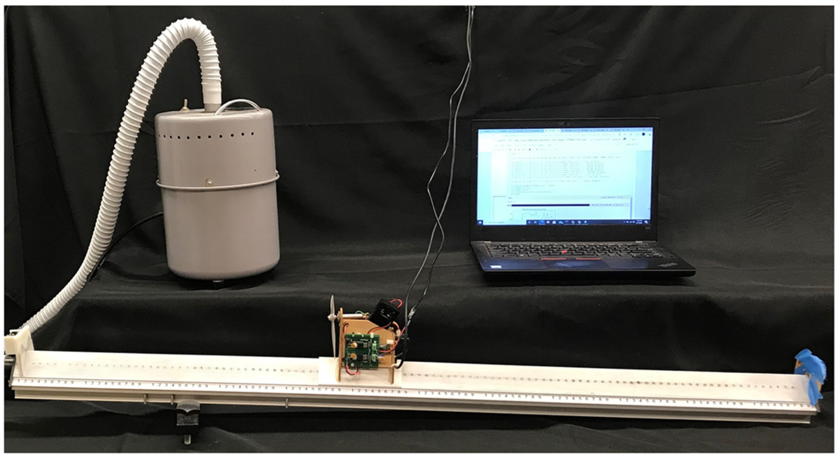

Fig. 1 Air track and air-sled environment: because the track is tilted at a 10-degree angle, the system is inherently unstable, requiring constant feedback control to maintain the air-sled's position

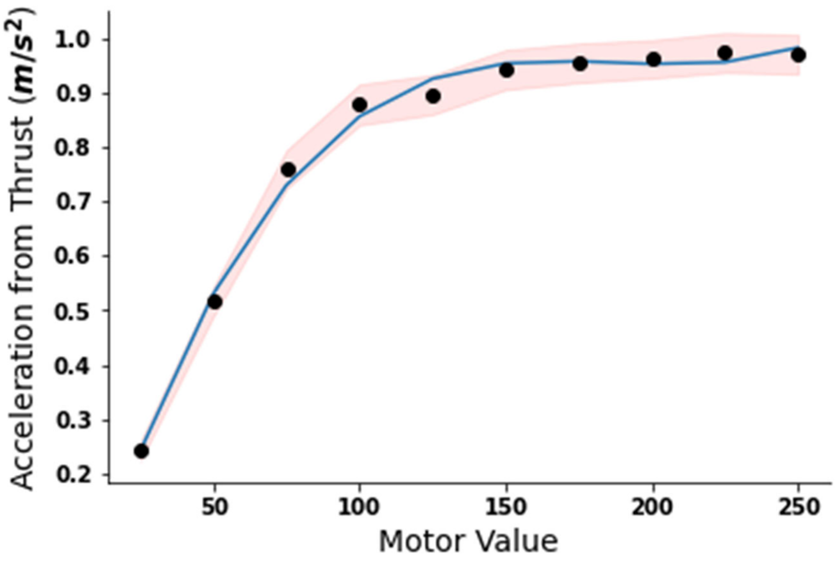

Fig. 2 Average acceleration due to propeller output for each measured motor value: a fitted line represents the sampling mean during simulation. The shaded area is 1 standard deviation from the averaged measurement values

fitted curve also depicted in Fig. 2. The average standard deviation of force output samples for motor values up to 100 was 0.005. We used this to determine the distributions from which the acceleration values were sampled in simulation.

To mimic the real-world environment as closely as possible, we used the unique exploration methods described in Section 3.2. This required forgoing the traditional random initialization of each episode to instead incorporate alternating exploration episodes, simulating the pseudo-episodic format of the physical experiment. Additionally, during exploration episodes, we implemented the safety zones meant for aided collision prevention.Throughout each training iteration, the total reward earned from every other training episode was stored for later comparison between algorithms. This was done to account for the alternation of action spaces between exploration episodes so that recorded rewards came from episodes with similar initial positions given our pseudo-episodic format. Although training occurred during these episodes in the same way as experienced in the other training episodes, we denote these episodes as "testing episodes" for clarification. It is important to keep in mind that although we performed a total of 101 episodes for each rendition of training, the graphs only consider the testing episodes, starting with the first, to set a baseline performance without training, and then occurring every fourth episode subsequently.

Figure 3 presents the learning curves for each base algorithm and their respective variants in the simulated environment. Because consistency is crucial when applying RL to real-world applications, several iterations of training occur. For our simulated case, we chose to average 10 iterations for each base algorithm and its variants, each iteration with random seeds ranging from 0 to 9. Since our final goal wasn't to achieve learning in simulation, but instead on the physical system, we will only briefly describe the simulation results. Off-PAC appears to accelerate learning for each of the baseline algorithms; however, the average reward earned appears to converge on values slightly less than that achieved by the baseline algorithms. The inclusion of ER during exploration shows the fastest

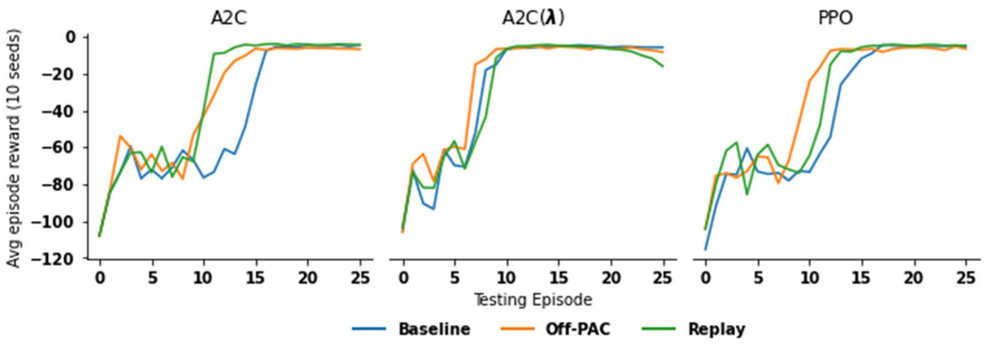

Fig. 3 Average reward earned per testing episode in simulated training for each baseline algorithm (A2C, A2C(λ), and PPO) and exploration episode update variants (Off-PAC and Experience Replay). It is important to note that a combined total of 101 episodes of traditional training and exploration were completed for each round of training. Testing episodes occurred every other training episode (every fourth episode when including exploration) to improve initial position consistency for testing and reward comparison

learning of all algorithm variations for A2C and PPO, and, in all cases, earned the highest average reward. These results are encouraging for when learning is conducted completely online in the experimental hardware.

As with any implementation of RL, hyperparameter tuning is a necessity. This intermediate step allowed us to pinpoint hyperparameter values for each algorithm that found success in this reinforcement learning problem. Several of these hyperparameters will remain unchanged when learning is moved to the experimental environment. Only the learning rates for each algorithm, α, α^{OP} and α^{ER} will be subject to change, but the relationships between the exploration update learning rates, α^{OP} and α^{ER}, and the common learning rate, α, will be maintained. These relationships and all other hyperparameters are given in Table 1. Determining these hyperparameters was one of two main purposes of this simulation work. The second of which being a justification to the merit of applying our training format, including policy updates during exploration episodes, to this RL problem before real-world implementation. So, although our experiment did not use any offline training, we did use simulation to determine a reasonable approach for hardware autonomy.

4 Experimental Demonstration and Validation

The objective of this research is to present an autonomous training scheme for developing learned controllers in a physical environment that would typically require human intervention or an additional controller for environment reset. In this section we introduce the experimental setup chosen to demonstrate such an environment, including the additional safety measures put in place to mitigate hardware damage. Additionally, we describe an adjustment made to the action selection strategy used in the traditional learning algorithm, and our reasoning for making this adjustment for hardware implementation.

Table 1 Algorithm hyperparameters

Hyperparameter	A2C	A2C(λ)	PPO
α	0.01	0.01	0.01
α^{OP}	3α	3α	3α
α^{ER}	0.5α	0.2α	0.1α
α^{zw}	0.01	0.01	0.01
λ	–	0.4	–
ϵ	–	–	0.2

[a]The variant learning rates, α^{OP} and α^{ER}, are listed as relationships to common learning rate, α

[b]α is adjusted to $10e-4$ when training on experimental equipment

4.1 Experimental Environment

Stability is the cost of maneuverability for aerobatic aircraft, requiring autonomous feedback control to maintain trim without explicit piloting. Using sensor input to accomplish equilibrium in an unstable environment provides an appropriate analog for such maneuverable flight vehicles. Therefore, to demonstrate the capability of our pseudo-episodic training scheme with exploration updates we chose to develop a controller for an air-sled such that it maintains a desired position on an inclined one-dimensional air-track, as seen in Fig. 1. A pump is used to force air through a series of holes set along the length of the track to allow for a near frictionless surface for the sled to slide along, controlled by a single propeller, simulating one dimensional trimmed flight. The propeller output is connected through a motor control board to an Arduino Nano microcontroller. In addition to providing a motor value signal between 0 and 100 to the motor control board, the Arduino Nano receives position information from an infrared distance sensor and communicates with external devices through USB. Distance gauged by the infrared sensor is used to calculate the velocity of the air-sled after each 0.05 second timestep. The USB connection allows information to be relayed between the air-sled and a laptop. The laptop runs a custom Python-based RL script that takes state data (position and velocity) as inputs and outputs the selected action for each timestep. Distance values are continually measured and stored as averaged sets of two in the Arduino Nano's serial buffer. At each timestep, when the learning algorithm asks for the air-sled position, the two most recent averaged values are read into the Python script to be averaged again and the serial buffer is emptied. This is done to limit the amount of data stored in the serial buffer and to smooth the noisy distance data obtained from the infrared sensor.

After moving to the hardware environment, two physical safety measures were put in place, in addition that implemented by the exploration's action selection, to mitigate the severity of any impact that may occur. The first physical safety measure occurred naturally through the placement of the power supply wiring and communication wires between the air-sled's Arduino Nano and the computer. By using the correct length and placement of this wiring, mounted above and centered over the air-track, the air-sled may move freely about most of the state space; however, when the air-sled moves closer to either end of the air-track, an additional force is applied to air-sled toward the center of the air-track due to the weight of the wiring. This reduced the velocity of the air-sled and often prevented a collision all together. This safety measure does come at a cost. Any movement experienced by the wire caused momentum and placed additional external force on the air-sled. This creates a greater variance in the accelerations experienced by the air-sled that must be overcome during training. In case this safety measure failed to prevent the air-sled from bumping into either end of the air-track, we included padding at either end of the air-track to mitigate any damage that may occur.

4.2 Deterministic Policy Gradient

Each of the algorithms chosen for this experiment are actor-critic methods which are typically stochastic in their decision-making processes. While stochasticity has many benefits for certain environments and allows for exploration to be more implicit in an algorithm's action selection, a stochastic controller will continue to take suboptimal actions after training. Although this undesirable decision-making is infrequent, it can lead to less stability in the final control of the air-sled, particularly when training time is limited. One option to get around this issue is to use a stochastic policy during training and then only select the action of highest probability during testing; however, this too failed to guarantee sufficient control after a short period of training in our experiment. As mentioned, one point of concern for training in a hardware-based environment is the amount of time necessary to achieve sufficient control of our air-sled. While time is always a constraint in RL, it becomes more influential in the physical setting due to wear and tear on equipment, potential accidents, and fuel consumption.

We found that by switching to a deterministic action selection during training, not only was the air-sled capable of smoother control, but it learned to do so in a much shorter training period, completing training in under 30 minutes. Figure 4 highlights this improved learning speed. To account for the loss of intrinsic exploration, we chose to implement an epsilon greedy action selection, as described in Section 3.2. Although deterministic policy gradient methods have been used before, as in [37], we took an approach similar to a simple off-policy update, using an importance sampling ratio between the stochastic probability of an action's occurrence, defined by trained policy, and the probability defined by the action selection policy. When following the deterministic action selection policy this probability is equal to one; however, when the action is selected following the random policy, occurring 10% of the time due to epsilon-greedy, the action selection probability is the inverse number of available actions. This leads to the full update equation,

$$\Delta\boldsymbol{\theta} = \alpha\delta\nabla\mathbf{ln}\pi_{\theta}(A|S,\boldsymbol{\theta}) * \frac{\pi_{\theta}(A|S,\boldsymbol{\theta})}{\pi_b}(A|S) \tag{12}$$

5 Results

In this section we apply our autonomous learning concepts to hardware, taking that step toward self-adaptive flight. After tuning the common learning rate, α, we found that a value of 10e-4 allowed for learning to be fast enough to complete training in 101 episodes (25 minutes) for all but one case, but slow enough to be stable and allow for recognizable differences between algorithm variants. All other hyperparameter values are available in Table 1. Although the values of the exploration update learning rates were adjusted for real-world training, their relation to the common learning rate was maintained as described in Table 1. All other hyperparameters were consistent with those used in simulation.

Episode reward is a common metric used to gauge an algorithm's ability to learn. Additionally, due to noise and randomness that occurs naturally in physical environments,

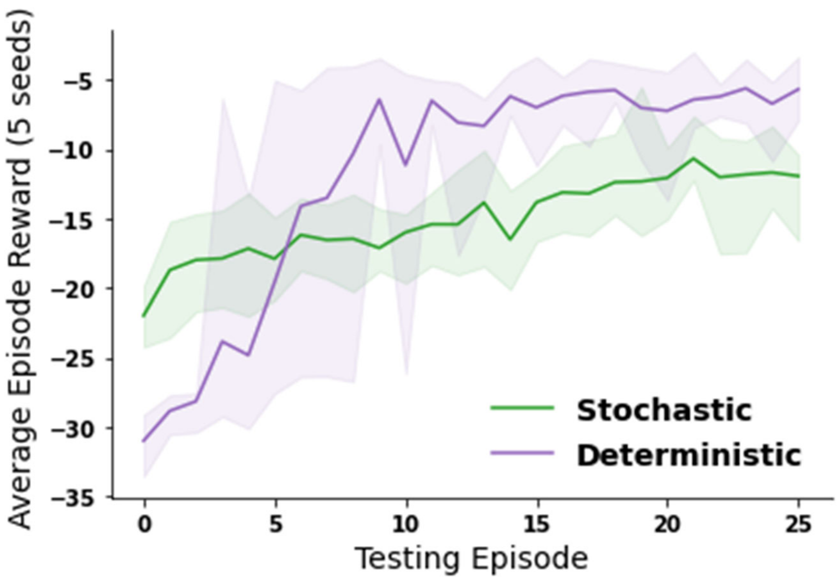

Fig. 4 Average reward earned per testing episode for stochastic and deterministic PPO throughout training on the physical experiment. The shaded area includes all reward values earned by each random seed in respective testing episodes

consistency becomes increasingly important for measuring the viability of an algorithm. To account for consistency, each algorithm of consideration, and its two additional variants, were repeated 5 times. The rewards from the 5 renditions, each with a different random seed ranging from 0 to 4, are averaged to represent the reward for that algorithm or variant thereof. These values, along with the standard deviation of reward earned per episode, are available for each algorithm and the additional variants in Fig. 5. While standard deviation may not be specifically meaningful for these small distributions, it can still be a useful means to quantify spread, and therefore consistency, of our algorithm's operation. In addition to consistency, another metric we consider for an algorithm's performance is speed of learning. This is crucial for learning in a real-world environment. In addition to the increased wear and tear placed on the equipment during lengthy training sessions, this extends the period in which poor actions may be taken, putting the agent in potentially dangerous positions. A black dashed line is included to represent the highest reward earned for each of the base algorithms, allowing us to quantify the speed of learning as the number of episodes required for each algorithm variant to surpass the base algorithm's maximum reward.

Figure 5 shows that with the basic A2C algorithm, the separate additions of Off-PAC and ER during the exploration episodes dramatically accelerate learning in the first several episodes, approaching and surpassing the maximum after 11 and 7 testing episodes for the addition of Off-PAC and ER respectively. The baseline A2C did not achieve this until the 22nd testing episode. The standard deviation plots show that incorporating Off-PAC and ER during exploration episodes both have improved the consistency of the algorithm's learning. One thing to note when looking at the standard deviation plots is that, across all three base algorithms, the standard deviation is always highest when the algorithm is doing most of its learning and the reward is still increasing. We can gauge when similar performance is achieved between iterations as the episode in which the standard deviation drops below and maintains a value beneath a defined low point. This speaks to the consistency of an algorithm's learning. After 9 test episodes, A2C has relatively high standard deviation. This means that although one of the algorithm's iterations may have approached its highest reward, and learned a suitable policy, several iterations have yet to do so. We see that this is true even after training has concluded; however, this changes with the addition of Off-PAC or ER. Adding Off-PAC to the algorithm allows for consistency in learning after 17 test episodes. For the case of ER, consistency is almost achieved as early as test 7, but there are two high peaks in standard deviation that occur around tests 10 and 15, and then again a smaller spike at 19. Because of this, we define the metric of "achieving consistency" ($\sigma < 4$) as the first test episode in which the standard deviation drops below 4, represented by the red dotted line in the Fig. 5, and then remains below 4 until training is complete. Additionally, we chose to record the total number of test episodes with standard deviations below 4. With this metric we can say A2C with ER achieved consistency after 21 episodes and had a total number of 13 test episodes where learning was consistent. These comparisons, in addition to the number of test episodes needed to reach the baseline algorithm's

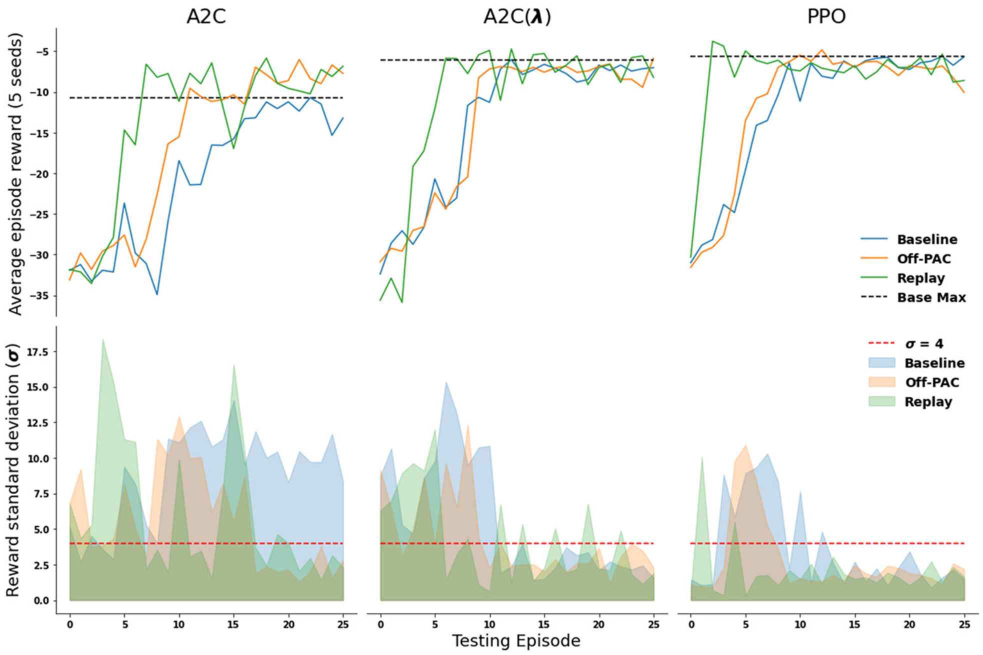

Fig. 5 Performance comparison between baseline algorithms (A2C, A2C(λ), and PPO) and their exploration episode update variants (Off-PAC and Experience Replay) in real-world training. The top 3 plots show the average reward earned per testing episode. The dashed black line represents the maximum average reward achieved by the respective baseline algorithm. Consistency in learning performance is gauged by the bottom three plots illustrating the standard deviation of rewards earned per training episode for each algorithm and its variants. We consider a standard deviation held below 4 as an indication of consistent learning performance

Table 2 Hardware learning performance metrics

Algorithm	Max avg. reward	TTBM	σ < 4(total)
A2C	–10.7	22	(4)
A2C Off-PAC	–6.02	11	17 (12)
A2C ER	–5.85	7	21 (13)
A2C(λ)	–6.05	12	11 (15)
A2C(λ) Off-PAC	–5.88	25	10 (18)
A2C(λ) ER	–4.71	6	23 (14)
PPO	–5.62	23	13 (18)
PPO Off-PAC	–4.83	10	8 (22)
PPO ER	–3.78	2	5 (24)

[a]Metrics include the maximum average reward achieved, the number of test episodes needed to reeach the baseline maximum rewared (TTBM), and the earliest test episode in which teh reward standard deviation dropped below and maintained a value less than 4 through the remainder of training (σ < 4)

[b]The total number of test episodes achieving a standard deviation less than 4 during training is also listed under σ < 4 in parenthesis

maximum reward (TTBM) as well as the overall maximum reward of each variant, are made for each of the three base algorithms and presented in Table 2.

Through these metrics we can see several patterns that begin to develop. In each case, the addition of Off-PAC or ER improved the overall learning achieved by the baseline algorithm when comparing highest average rewards achieved during training, with the ER variant consistently earning the highest average rewards overall. With regards to learning speed, measured by TTBM, the implementation of ER surpassed both the baseline and Off-PAC variant for each of the three algorithms; however, the addition of ER didn't frequently aid in learning consistency. This is well illustrated in both the A2C and A2C(λ) cases. Although implementing ER would typically achieve standard deviation values below 4 early in training, as the training continued the standard deviation would frequently spike, briefly increasing to a value above 4. These spikes would frequently only last for one episode, but at times the spike would require 3 testing episodes before reducing back to a value representative of consistent learning. On the other hand, introducing Off-PAC updates during the exploration phase improved consistency in every case. Off-PAC was not the fastest algorithm variant when comparing TTBM. When added to A2C(λ), Off-PAC was slower than the baseline. However, adding Off-PAC updates during exploration achieved learning consistency after the fewest number of testing episodes for 2 of the 3 algorithms. Interestingly, this was not the case for PPO, where ER proved to give an advantage in speed and also showed capability in consistent learning. This could be due to PPO's clipping mechanism that is designed specifically to prevent updates from becoming too large and overshooting, potentially improving learning consistency. Although PPO's baseline algorithm does not appear to be as fast at achieving consistency as A2C(λ) according to σ < 4, this could be due to PPO's slower learning speed, not reaching its maximum average reward until episode 23. Therefore, when another algorithm is used to accelerate learning, such as ER, the result is a fast learning algorithm where the overall standard deviation of reward is able to remain low.

In addition to reward earned throughout training, it is important that the policy learned by the algorithms can

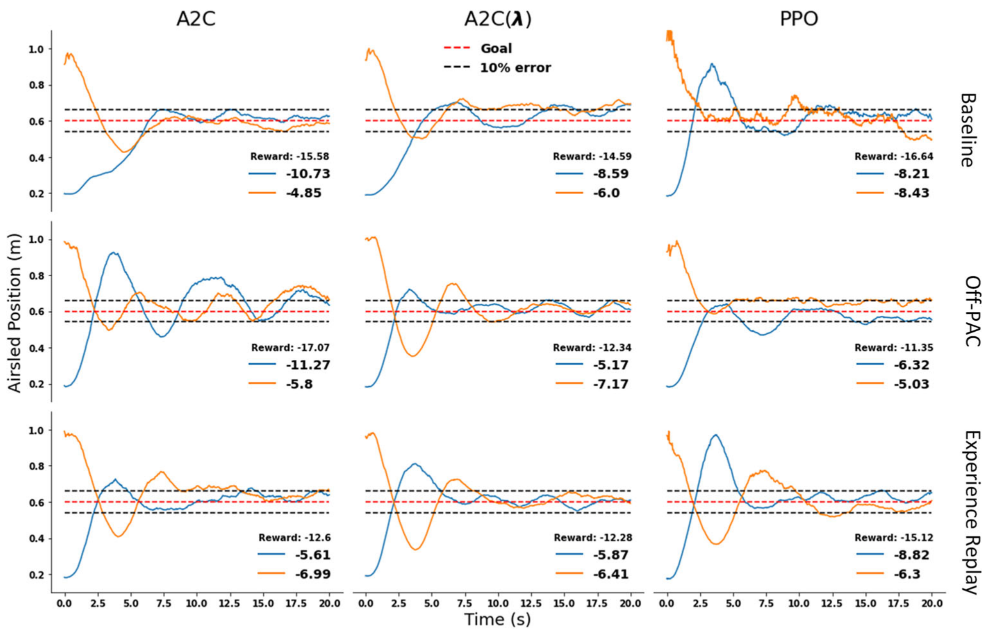

Fig. 6 Position over time of the air-sled on the air track for the best performing policy from each baseline algorithm (A2C, A2C(λ), and PPO) and their exploration episode update variants (Off-PAC and Experience Replay). Two initial positions are included, each located ±0.4 meters (±1.31 ft) from the target. The target position of 0.6 meters (1.97 ft) is represented as a red dashed line and positions of 10% error from the target are black. Reward earned for each time series and the combined total reward are given in the bottom right corner of each plot

control the air-sled to a satisfactory degree after training is completed. In Fig. 6 we see the position values over a period of 20 seconds for the random seed of best performance for each of the base algorithms and their variants. In each plot we see the behavior for two initial positions, one near each end of the air-track. Due to the noise of the infrared sensor, we applied an averaging technique for smoothing the data to improve recognition of the air-track's position for each timestep. The black dotted lines represent locations of 10% error from the target position of 0.6 meters from the end of the air-track. In many cases of control, metrics such as rise time and settling time are used to gauge performance. However, our controllers were trained based on another metric, earned reward, which we can use again to compare overall performance between learned policies for the two initial conditions. When considering traditional metrics, A2C(λ) with Off-PAC achieves the best control from an initial position of 0.2 meters and the baseline A2C algorithm trained the best policy for control from an initial position of 1 meter, needing only 4.2 and 6.25 seconds to settle between the 10% error margins respectively. However, when we compare the rewards earned over the duration of the control test, although these algorithms did generate the best control for their respective initial positions, the overall best performance was achieved by the PPO algorithm with Off-PAC exploration updates, earning a combined reward of -11.35. The next best performances in order came from A2C(λ) with ER, A2C(λ) with Off-PAC, and A2C with ER, each earning a combined reward greater than -13. The top 4 performances came from algorithms with the addition of Off-PAC or ER. This suggests that the inclusion of some form of off-policy update during the exploration episodes can benefit training for control with a variety of actor-critic algorithms, assuming performance is measured by the system used to direct learning.

6 Conclusion and Future Work

The implementation of reinforcement learning in physical experiments has proven to be a difficult task. In this work we developed a pseudo-episodic approach for the autonomous training of an RL agent in a one-dimensional, unstable environment. Our method is model-free and uses only information gathered by an on-board sensor. Although training is achieved entirely online, its structure allows for additional policy updates to occur between training episodes. Additionally, we validated this autonomous training method in experimental hardware. The addition of ER and Off-PAC updates during the exploration episodes showed training benefits such as improved speed and consistency in learning respectively. When paired with PPO, ER performed particularly well, overcoming its weakness in maintaining learning consistency. Further improvement was displayed in controller performance when using reward as the compared metric.

This work focuses on using novel techniques to improve speed and consistency in learning, while maintaining safety and autonomy in a real-world environment; however, the resulting controllers lack accuracy. Learning quickly and safely may be prioritized over accuracy in many environments, such as a UAV with precious cargo adapting to a new environment where it is more important to quickly learn safe flight than to achieve optimum performance. With that being said, there are several environments where final performance is the priority; therefore, future work should be dedicated to achieving optimal control, at the sacrifice of speed if necessary. Another potential avenue for this work is to implement neuromorphic chips to allow fast and continuous hardware based learning in this unstable system, taking another step toward in flight adaptation [9].

Acknowledgements We are grateful to Professor Peter Washabaugh, at the University of Michigan's Department of Aerospace Engineering, for providing the air-sled and air-track experiment and to Connor Stadler, an undergraduate in the same department for working to determine an efficient method of communicating state and action information between the Arduino and Python script.

Author Contributions Both authors contributed to the concept and design of the experiment presented in this article. Algorithm design, experimental setup, data collection and analysis where completed by Kevin Haughn. The first draft of this manuscript was written by Kevin Haughn. Both authors were active in revising the first draft, and all subsequent versions, and approve of the final manuscript.

Funding This work is supported in part by the US Air Force Office of Scientific Research under a grant number AFOSR-FA9550-19-0213, titled "Brain Inspired Networks for Multifunctional Intelligent Systems in Aerial Vehicles" and in part by grant number FA9550-16-1-0087, titled "Avian-Inspired Multifunctional Morphing Vehicles" both monitored by Dr BL Lee.

Availability of data and material Not applicable

Code Availability Code used to exemplify methods implemented in this paper may be available upon request.

Declarations

Ethics approval This article has the approval of both authors.

Consent to participate Both authors gave consent to participate in this article.

Consent for Publication Both authors authorized the publishing of this article.

Conflict of Interests The authors have no conflict of interest.

References

1. Ahang, S., Sutton, R.S.: A deeper look at experience replay. arXiv:1712.01275 (2017)
2. Andrychowicz, O.M., Baker, B., Chociej, M., Jozefowicz, R., McGrew, B., Pachocki, J., Petron, A., Plappert, M., Powell, G., Ray, A., et al: Learning dexterous in-hand manipulation. Int. J. Robot. Res. **39**(1), 3–20 (2020)
3. Atkeson, C.G., Santamaria, J.C.: A comparison of direct and model-based reinforcement learning. In: Proceedings of International Conference on Robotics and Automation, vol. 4, pp. 3557–3564. IEEE (1997)
4. Babaeizadeh, M., Frosio, I., Tyree, S., Clemons, J., Kautz, J.: Reinforcement learning through asynchronous advantage actor-critic on a gpu. arXiv:1611.06256 (2016)
5. Berkenkamp, F., Turchetta, M., Schoellig, A.P., Krause, A.: Safe model-based reinforcement learning with stability guarantees. arXiv:1705.08551 (2017)
6. Berner, C., Brockman, G., Chan, B., Cheung, V., Debiak, P., Dennison, C., Farhi, D., Fischer, Q., Hashme, S., Hesse, C., et al: Dota 2 with large scale deep reinforcement learning. arXiv:1912.06680 (2019)
7. Bøhn, E., Coates, E.M., Moe, S., Johansen, T.A.: Deep reinforcement learning attitude control of fixed-wing uavs using proximal policy optimization. In: 2019 International Conference on Unmanned Aircraft Systems (ICUAS), pp. 523–533. IEEE (2019)
8. Caarls, W., Schuitema, E.: Parallel online temporal difference learning for motor control. IEEE Trans. Neural Netw. Learn. Syst. **27**(7), 1457–1468 (2015)
9. Chen, Y.: Brain-inspired synaptic resistor circuits for self-programming intelligent systems. Advanced Intelligent Systems, p 2000219 (2021)
10. Degris, T., White, M., Sutton, R.S.: Off-policy actor-critic. arXiv:1205.4839 (2012)
11. Dulac-Arnold, G., Mankowitz, D., Hester, T.: Challenges of real-world reinforcement learning. arXiv:1904.12901 (2019)
12. Finn, C., Abbeel, P., Levine, S.: Model-agnostic meta-learning for fast adaptation of deep networks. In: International Conference on Machine Learning, pp. 1126–1135. PMLR (2017)
13. Fujimoto, S., Conti, E., Ghavamzadeh, M., Pineau, J.: Benchmarking batch deep reinforcement learning algorithms. arXiv:1910.01708 (2019)
14. Fujimoto, S., Meger, D., Precup, D.: Off-policy deep reinforcement learning without exploration. In: International Conference on Machine Learning, pp. 2052–2062. PMLR (2019)
15. Haarnoja, T., Zhou, A., Abbeel, P., Levine, S.: Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In: International Conference on Machine Learning, pp. 1861–1870. PMLR (2018)
16. Hessel, M., Modayil, J., Van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., Horgan, D., Piot, B., Azar, M., Silver, D.: Rainbow: Combining improvements in deep reinforcement learning. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 32 (2018)
17. Hwangbo, J., Sa, I., Siegwart, R., Hutter, M.: Control of a quadrotor with reinforcement learning. IEEE Robot. Autom. Lett. **2**(4), 2096–2103 (2017)
18. Imanberdiyev, N., Fu, C., Kayacan, E., Chen, I.M.: Autonomous navigation of uav by using real-time model-based reinforcement learning. In: 2016 14th International Conference on Control, Automation, Robotics and Vision (ICARCV), pp. 1–6. IEEE (2016)
19. Kersandt, K.: Deep Reinforcement Learning as Control Method for Autonomous uavs. Master's thesis, Universitat Politècnica de Catalunya (2018)
20. Kirkpatrick, J., Pascanu, R., Rabinowitz, N., Veness, J., Desjardins, G., Rusu, A.A., Milan, K., Quan, J., Ramalho, T., Grabska-Barwinska, A., et al.: Overcoming catastrophic forgetting in neural networks. Proceedings of the national academy of sciences **114**(13), 3521–3526 (2017)
21. Kober, J., Peters, J.: Imitation and reinforcement learning. IEEE Robot. Autom. Mag. **17**(2), 55–62 (2010)
22. Koch, W., Mancuso, R., West, R., Bestavros, A.: Reinforcement learning for uav attitude control. ACM Trans. Cyber-Phys. Syst. **3**(2), 1–21 (2019)
23. Konidaris, G., Osentoski, S., Thomas, P.: Value function approximation in reinforcement learning using the fourier basis. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 25 (2011)
24. Lange, S., Gabel, T., Riedmiller, M.: Batch reinforcement learning. In: Reinforcement Learning, pp. 45–73. Springer (2012)
25. Lin, L.J.: Self-improving reactive agents based on reinforcement learning, planning and teaching. Mach Learn **8**(3-4), 293–321 (1992)
26. Mnih, V., Badia, A.P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., Kavukcuoglu, K.: Asynchronous methods for deep reinforcement learning. In: International Conference on Machine Learning, pp. 1928–1937. PMLR (2016)
27. Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., Riedmiller, M.: Playing atari with deep reinforcement learning. arXiv:1312.5602 (2013)
28. Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., et al: Human-level control through deep reinforcement learning. Nature **518**(7540), 529–533 (2015)
29. Mülling, K., Kober, J., Kroemer, O., Peters, J.: Learning to select and generalize striking movements in robot table tennis. Int J Robot Res **32**(3), 263–279 (2013)
30. Ng, A.Y., Kim, H.J., Jordan, M.I., Sastry, S., Ballianda, S.: Autonomous helicopter flight via reinforcement learning. In: NIPS, vol. 16. Citeseer (2003)
31. Pham, H.X., La, H.M., Feil-Seifer, D., Nguyen, L.V.: Autonomous uav navigation using reinforcement learning. arXiv:1801.05086 (2018)
32. Rolnick, D., Ahuja, A., Schwarz, J., Lillicrap, T., Wayne, G.: Experience replay for continual learning. Advances in Neural Information Processing Systems **32**, 350–360 (2019)
33. Schulman, J., Levine, S., Abbeel, P., Jordan, M., Moritz, P.: Trust region policy optimization. In: International Conference on Machine Learning, pp. 1889–1897. PMLR (2015)
34. Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O.: Proximal policy optimization algorithms. arXiv:1707.06347 (2017)

35. Silver, D., Huang, A., Maddison, C.J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al: Mastering the game of go with deep neural networks and tree search. Nature **529**(7587), 484–489 (2016)
36. Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., et al: A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. Science **362**(6419), 1140–1144 (2018)
37. Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., Riedmiller, M.: Deterministic policy gradient algorithms. In: International Conference on Machine Learning, pp. 387–395. PMLR (2014)
38. Smart, W.D., Kaelbling, L.P.: Effective reinforcement learning for mobile robots. In: Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No. 02CH37292), vol. 4, pp. 3404–3410. IEEE (2002)
39. Sutton, R.S., Barto, A.G.: Reinforcement learning: An introduction. MIT Press (2018)
40. Tai, L., Paolo, G., Liu, M.: Virtual-to-real deep reinforcement learning: Continuous control of mobile robots for mapless navigation. In: 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 31–36. IEEE (2017)
41. Tobin, J., Fong, R., Ray, A., Schneider, J., Zaremba, W., Abbeel, P.: Domain randomization for transferring deep neural networks from simulation to the real world. In: 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 23–30. IEEE (2017)
42. Wang, Z., Bapst, V., Heess, N., Mnih, V., Munos, R., Kavukcuoglu, K., de Freitas, N.: Sample efficient actor-critic with experience replay. arXiv:1611.01224 (2016)
43. Wu, Y., Mansimov, E., Liao, S., Grosse, R., Ba, J.: Scalable trust-region method for deep reinforcement learning using Kronecker-factored approximation. arXiv:1708.05144 (2017)
44. Zhu, H., Yu, J., Gupta, A., Shah, D., Hartikainen, K., Singh, A., Kumar, V., Levine, S.: The ingredients of real-world robotic reinforcement learning. arXiv:2004.12570 (2020)

Kevin PT. Haughn received the B.S. degree in aerospace engineering from the University of Michigan, Ann Arbor, MI, in 2018. He is currently pursuing a Ph.D. in aerospace engineering, also at the University of Michigan. He began his research career as a research assistant during his final two semesters of undergraduate study in 2017 and 2018. After graduation he continued his research as a graduate student research assistant and in the spring of 2019 earned promotion to Ph.D. candidacy. Mr. Haughn's current research interests include combining machine learning methods with smart material driven, multifunctional morphing aerospace structures to develop intelligent, adaptive uncrewed aerial vehicles.

Daniel J. Inman received his Ph.D. from Michigan State University in Mechanical Engineering in 1980 and is the Harm Buning Collegiate Professor and former Chair of the Department of Aerospace Engineering at the University of Michigan. Since 1980, he has published eight books (on vibration, energy harvesting, control, statics, and dynamics), eight software manuals, 20 book chapters, over 400 journal papers and 671 proceedings papers, given 72 keynote or plenary lectures, graduated 68 Ph.D. students, and supervised more than 75 MS degrees. He works in the areas of applying smart materials and structures to solve aerospace engineering problems including energy harvesting, structural health monitoring, vibration suppression and morphing aircraft. He is a Fellow of the American Institute of Aeronautics and Astronautics, American Society of Mechanical Engineers, International Institute for Acoustics and Vibrations, Society of Experimental Mechanics and American Academy of Mechanics. He won the ASME Adaptive Structures Award in April 2000, SPIE Smart Structures and Materials Lifetime Achievement Award in March of 2003, he received the ASME Den Hartog Award for lifetime achievement in teaching and research in vibration, the 2009 Lifetime Achievement award in Structural Health Monitoring, and the AIAA Structures, Structural Dynamics, and Materials Award, in 2014. He is currently Technical Editor of the Journal of Intelligent Material Systems and Structures (1999-present).