Learning Quadrotor Control Tasks Through Deep Reinforcement Learning

Anonymous

***Abstract*— Learning quadrotor control tasks using deep reinforcement learning is an active area of research that is heavily dependent on the existence of quality learning environments. To date, research has been split on the use of ROS for learning control tasks, and custom-built simulations. In this paper, we present a set of challenging environments built in Python and extending the OpenAI Gym framework for learning interesting flight tasks that are not well treated by the literature. We provide an overview of the simulation and tasks, and present initial results applying current state-of-the-art algorithms towards learning goal-conditioned flight control policies. We show that current Monte-Carlo policy search methods are capable of learning continuous-action flight control without any additional learning aids.**

I. INTRODUCTION

Deep reinforcement learning has achieved recent successes in playing video games [1], [2] and learning robotics tasks from scratch [3], [4], as well as showing early promise in the guidance, navigation, and control of MAVs [5], [6], [7]. Though MAV control is a well-established field, many complex tasks still require human oversight, and techniques for reducing human involvement are still nascent. Deep reinforcement learning is a potential avenue for increasing the autonomy of such systems by directly training robotic systems to execute complex, human-like behaviors from scratch.

One of the major impediments to this goal is the need for environments that mirror the complexity of the real world. To date, computer games have provided the closest proxy, but equivalent frameworks for robotics have been lacking. In the domain of micro-air vehicles (MAVs), existing research has been split between use of the Robot Operating System (ROS), or writing custom simulations; in the case of ROS, the message-passing framework can be unwieldy, and is not as useful for reinforcement learning where researchers are often more interested in Monte-Carlo-based learning algorithms applied to a specific task. Custom simulations have the advantage of being minimalist, but are often difficult to adapt to new scenarios. There is no framework in either case that includes a variety of flight tasks that researchers might be interested in. As a consequence, most MAV RL applications have focused primarily on single, low-level control tasks, rather than higher level tasks for which RL is potentially better suited.

In this paper, we present ongoing work in building such a benchmark for MAV flight, that extends the OpenAI Gym framework. Accordingly, we present preliminary results in learning different flying tasks using current state-of-the-art policy search algorithms. One of our goals with this research is to provide a platform for deep reinforcement learning of complex aerial tasks without additional software dependencies beyond those required for Gym, and to enable code reuse for easy testing. We make the following contributions:

1) We introduce a quadrotor flight simulation and associated environments that interface with the OpenAI Gym framework, and require no additional dependencies;
2) We benchmark state-of-the-art methods on these tasks. To date, ours is the most comprehensive application of RL methods applied to quadrotor flight control that we know of, covering a wider range of tasks than previous works; and finally,
3) We show what is – to our knowledge – the only successful application of completely unaided, first-order methods to learning continuous-action quadrotor control tasks in the literature.

The structure of this paper is as follows: the following section outlines related work, and motivates the need for a collection of environments specific to aircraft. Section III outlines the underlying physics model and assumptions made, and details the process of policy optimization. We then outline the environments themselves in Section IV, before describing our experiments in V. Results and analysis are presented in Section VI, with a brief conclusion in Section VII.

II. RELATED WORK

[7], [9] used MPC-Guided Policy Search to learn to fly a quadrotor in Gazebo through unstructured environments using LIDAR. This was done using a heavily modified fork of RotorS [8], with a custom control node running the policy search algorithm. This research proved the efficacy of using RL to train continuous-action controllers for quadrotors.

[10] used a method similar to guided policy search to train a quadrotor controller to avoid collisions in obstacle-filled environments. In order to overcome the relative sparsity of such events, [10] used a novel sampling strategy to overexpose the agent to collisions, resulting in a more risk averse policy. The authors successfully tested the learned policy on the CrazyFlie nanoquad using a native C-library for the network weights.

[6] used a custom natural policy gradient method to recover a tumbling aircraft. To do this, the authors used a novel ”junction” exploration strategy, and wrote a very simple flight dynamics model – omitting the effects of drag – to demonstrate that they could still learn robust control policies using deep RL. The policies were successfully tested

on a real-world aircraft using a Vicon camera system for state estimation.

These previous studies are typically constrained to a single task, and – in the case of [6], [7], [9], [10] – make use of an expert to guide learning. In [7], [9], [10], a trajectory planner was used to turn the task of policy search into a supervised learning task, whereas in [6], a PD controller was used to help stabilize learning (though the final policy did not use the controller, and far exceeded its capabilities). Our work does not make use of such additional machinery, and learns the policy directly. Furthermore, we learn controllers for a greater number of tasks than these previous works.

Surprisingly, no work that we know of has attempted training current state-of-the-art algorithms without any additional aids. Other works have reported that DDPG struggles to learn continuous flight control tasks [6], but no works have attempted using GAE or PPO for the problem of continuous-action flight control (though TRPO was used as a comparison in [6]). Given that PPO and TRPO are state-of-the-art algorithms with notable success in continuous-action control tasks – and, more recently – DOTA2 [11], it stands to reason that we might want to test such algorithms on interesting flight tasks.

III. BACKGROUND

A. Quadrotor Dynamics

We model a plus-configuration aircraft in an East-North-Up axis system, and assume a flat Earth with constant air density. We provide a rough diagram of our aircraft in Fig. 1. Our equations of motion are:

$$\begin{bmatrix}0\\ \dot{\mathbf{v}}\end{bmatrix} = \frac{1}{m}\begin{bmatrix}0\\ \mathbf{F}_b\end{bmatrix} + \mathbf{q}\otimes\begin{bmatrix}0\\ \mathbf{G}_i\end{bmatrix}\otimes\mathbf{q}^{-1} - \begin{bmatrix}0\\ \omega\times\mathbf{v}\end{bmatrix} \tag{1}$$

$$\dot{\omega} = \mathbf{J}^{-1}(\mathbf{Q}_b - \omega\times\mathbf{J}\omega) \tag{2}$$

$$\begin{bmatrix}0\\ \dot{\mathbf{x}}\end{bmatrix} = \mathbf{q}^{-1}\otimes\begin{bmatrix}0\\ \mathbf{v}\end{bmatrix}\otimes\mathbf{q} \tag{3}$$

$$\dot{\mathbf{q}} = -\frac{1}{2}\begin{bmatrix}0\\ \omega\end{bmatrix}\otimes\mathbf{q} \tag{4}$$

Where $\dot{\mathbf{v}}$, $\dot{\omega}$, $\mathbf{v}$, and ω are the linear and angular accelerations and velocities expressed in the body frame, $\mathbf{F}_b$ an $\mathbf{Q}_b$ are the external (thrust and aerodynamic) forces and moments acting on the aircraft and expressed in the body frame, $\mathbf{G}_i$ is the gravity vector expressed in the inertial frame, and $\mathbf{J}$ is the aircraft's inertia tensor. The unit quaternion $\mathbf{q}$ encodes the aircraft's attitude, with $\mathbf{q}^{-1}$ being its inverse, and $\otimes$ denoting the Hamilton product.

Motor thrusts and torques are modeled as:

$$\mathbf{F}_T = k_t\begin{bmatrix}0\\ 0\\ \sum_{i=1}^{m}\Omega_i^2\end{bmatrix} \tag{5}$$

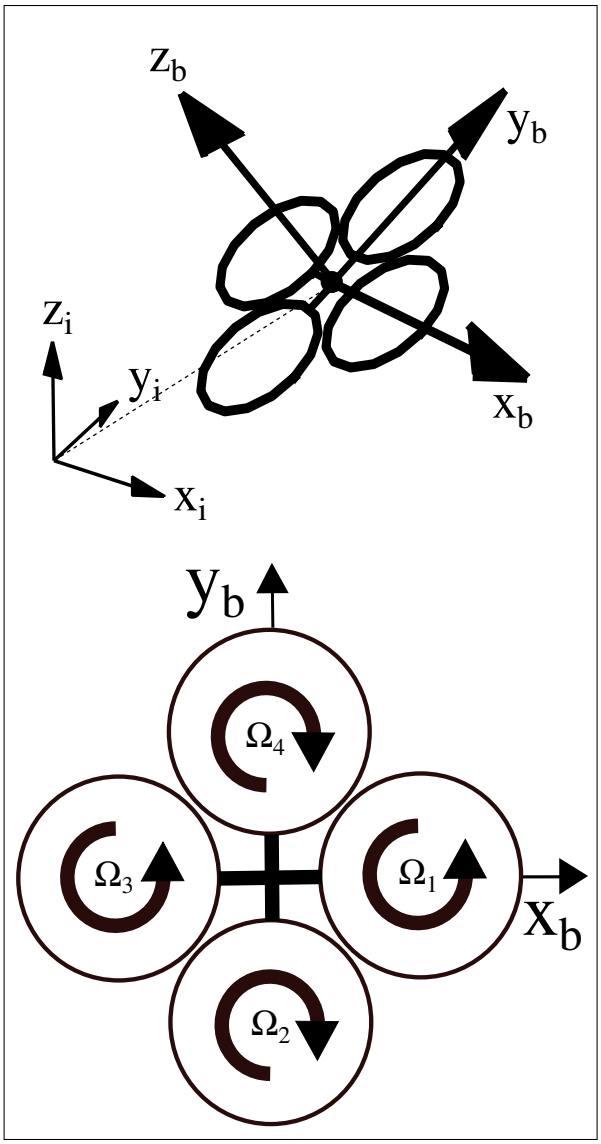

Fig. 1. A plus-configuration quadrotor is modeled in an East-North-Up axis system. Thrust forces act in the positive z-direction in the body frame, and we use a standard right-hand convention for rotations. Rotation arrows on the lower figure denote the direction of rotation of the motors, with torque acting in the opposite direction.

$$\mathbf{Q}_T = \begin{bmatrix}lk_T(\Omega_2^2-\Omega_4^2)\\ lk_T(\Omega_1^2-\Omega_3^2)\\ k_Q(-\Omega_1^2+\Omega_2^2-\Omega_3^2+\Omega_4^2)\end{bmatrix} \tag{6}$$

Where Ω is the angular velocity of the motor, l is the arm length between the centre-of-mass and the centre-of-thrust, and k_T and k_Q are the thrust and torque coefficients of the motor, respectively. Aerodynamic effects are modeled using two positive scalar coefficients – a linear drag coefficient k_D, and an aerodynamic moment coefficient k_M:

$$\mathbf{F}_A = -k_D(\mathbf{v}^T\mathbf{v})\hat{\mathbf{v}} \tag{7}$$

$$\mathbf{Q}_A = -k_M(\omega^T\omega)\hat{\omega} \tag{8}$$

With $\hat{\mathbf{v}}$ and $\hat{\omega}$ being the normalized linear and angular velocity vectors. Body forces and moments are found using $\mathbf{F}_b = \mathbf{F}_T + \mathbf{F}_A$, and $\mathbf{Q}_b = \mathbf{Q}_T + \mathbf{Q}_A$. Motor response is modeled as a first order linear differential equation of the form:

$$\dot{\Omega} = -k_\Omega \left(\Omega - \Omega_c\right) \tag{9}$$

Where Ω is the current angular velocity of the motors, Ω_c is the commanded angular velocity, and k_Ω is a damping coefficient. We integrate the equations of motion using a standard RK4 scheme. Our simulation assumes deterministic dynamics, and doesn't model more advanced phenomena such as ground effect, indoor (cave) aerodynamics, blade flapping, or vortex ring state (VRS).

We constrain hover thrust to a given percentage of the maximum thrust, and then set limits accordingly using the mass of the aircraft. The current parameters are estimated placeholders until more accurate coefficients can be determined by means of flight testing. Our primary goal at this stage is investigating how well current state-of-the-art methods are able to translate to tasks in a 6DOF dynamical system with no inherent stability, rather than modeling and validating a specific aircraft.

B. Policy Search

We assume a Markov Decision Process with continuous states $s \in \mathcal{S}$ and continuous actions $a \in \mathcal{A}$. We denote a trajectory τ as an ordered set of events $\{s_0, a_0, r_0, ..., s_{H-1}, a_{H-1}, r_{H-1}, s_H\}$ from time $t = 0$ to $t = H$, for states and actions distributed according to a policy π, and rewards $r \in \mathcal{R}$ from the environment [12], [13].

The goal of our agent is to maximize a gamma-discounted sum of rewards over time:

$$G_t = r_0 + \gamma r_1 + \gamma^2 r_2 + ... + \gamma^H r_H = \sum_{i=0}^{H} \gamma^i r_{t+i+1} \tag{10}$$

Where H can be either a finite or infinite horizon, and $\gamma \in [0, 1]$ is a discount factor that ensures that G_t converges for $H = \infty$.

The state and state-action value functions of policy π are defined as:

$$V^\pi(s_t) = \mathbb{E}_\pi \left[\sum_{i=0}^{H} \gamma^i r_{t+i+1} | s_t = s \right] \tag{11}$$

$$Q^\pi(s_t, a_t) = \mathbb{E}_\pi \left[\sum_{i=0}^{H} \gamma^i r_{t+i+1} | s_t = s, a_t = a \right] \tag{12}$$

With the advantage function of π being:

$$A^\pi(s_t, a_t) = Q^\pi(s_t, a_t) - V^\pi(s_t) \tag{13}$$

Our policy is a stochastic function $\theta : \mathcal{S} \rightarrow \mathcal{A}$, and our goal is to find a set of weights θ that maximizes the expected return over trajectories sampled under π.

We focus on the class of methods known as Monte-Carlo Policy Gradients, for which the policy selects actions, and a learned value function of the form $\phi : \mathcal{S} \rightarrow \mathbb{R}$ evaluates the resulting state. The value function is typically trained by minimizing the loss:

$$\mathcal{L}_{V^\pi} = r + \gamma \hat{V}^\pi(s_{t+1}) - V^\pi_\phi(s_t) \tag{14}$$

In which $\hat{V}^\pi$ refers to the Monte-Carlo estimate of V^π, and V^π_ϕ corresponds to the value function parameterized by ϕ. Gradient ascent is used to maximize the policy return using a stochastic gradient estimator:

$$\nabla_\theta J(\theta) = \mathbb{E}_\pi \left[\sum_{t=0}^{H-1} \nabla_\theta \log \pi_\theta(a_t|s_t) f^\pi(s_t, a_t) \right] \tag{15}$$

to learn on-policy [12], [14], or using an importance sampling estimator:

$$\nabla_\theta J(\theta) = \mathbb{E}_\beta \left[\sum_{t=0}^{H-1} \frac{\nabla_\theta \pi_\theta(a_t|s_t)}{\mu(a_t|s_t)} f^\mu(s_t, a_t) \right] \tag{16}$$

to learn off-policy [12], [15], [16]. $f^\pi(s_t, a_t)$ can take the form of Eqn. 14, an n-step return (noting that the value functions can be written recursively), or interpolation between the one-step return $r_t + \gamma V^\pi(s_t)$ and the infinite horizon return $r_t + \gamma r_{t+1} + \gamma^2 r_{t+1} + ... + \gamma^n V^\pi(s_{t+n})$ – known as TD-λ when interpolating over a value function [12], and Generalized Advantage Estimation when interpolating over the advantage function [16].

State-of-the-art policy gradient methods such as PPO and TRPO [18], [19] use the importance sampling estimator for on-policy learning, by bounding the update to a trust region over multiple update steps. The reason for this is that when updating the policy, we aim to take small steps, but small steps in the parameter space may not correspond to small steps in the action space. TRPO and PPO restrict the KL-Divergence between the old and new policies, which in theory guarantees monotonic policy improvement:

$$\begin{array}{ll} \text{maximize}_\theta & \mathcal{L}_{\theta_{old}}(\theta) \\ \text{subject to} & \bar{D}^{\rho\theta_{old}}_{KL}(\theta_{old}||\theta) \leq \delta \end{array} \tag{17}$$

PPO is a first-order method that approximates the second-order update used in TRPO, using either a clipped objective, or an adaptive penalty based on the KL-divergence. Such methods typically use generalized advantage estimation to interpolate between the one-step return and the infinite horizon return. Per the definition of the value function, $V^\pi(s_t) \neq V^\mu(s_t)$ for arbitrary policies π and μ [15], and so an off-policy correction should be made when taking multiple gradient steps using a single roll-out batch.

IV. ENVIRONMENTS

Our environments inherit from the OpenAI Gym Env class, and wrap the simulation so that we can step the physics forward and return a nominally consistent set of observations

across tasks. As well as allowing us to use Gym wrappers, this framework allows us to use existing implementations of common policy search methods with minimal modification, and rapidly benchmark our own implementations on other tasks. The environments included so far are:

1) Hover, in which the goal is to hover at a fixed point for the duration of the episode;
2) Static waypoint navigation, in which the objective is to navigate to a fixed goal position, and then hover there for the remainder of the episode;
3) Random waypoint navigation, in which the objective is to travel to a randomly generated goal position, and then hover there for the remainder of the episode;
4) Landing, in which the aircraft must land on a target platform from an initial starting position, without crashing, and without descending through its own rotor wash; and,
5) Target following, in which the aircraft must follow a moving target whilst keeping to a constant range.

Our observation space typically includes the position $\mathbf{x}$, the sin and cosine of the attitude ζ in the inertial frame, and the linear and angular velocities $\mathbf{v}$ and ω in the body frame. We also include the current speed of the rotors, normalized to the interval $[0, 1]$, and the vector difference between the aircraft's current state and goal state. Where attitude-based goals are used, we use $\sin(\zeta_{curr}) - \sin(\zeta_{targ})$, and $\cos(\zeta_{curr}) - \cos(\zeta_{targ})$. As our simulation does not currently include wind effects, the ground speed and airspeed are one and the same; accordingly, only the airspeed is currently included in the observation space.

For the purposes of this work, we assumed that the aircraft has access to the ground truth, or a sufficiently good estimate of the ground truth that the policy can use it directly; though this is unlikely to be the case, we consider the task of accurate localization to be a separate problem for the time being. For good examples of how one might localize the aircraft and apply a learned policy, see [6], [10].

The reward function is typically defined to be:

$$r_t = \alpha_t \Delta t + \sum_{c=1}^{C} \alpha_c \left(d_{t-1}^{(c)} - d_t^{(c)} \right) + \alpha_k^{(c)} \mathbb{I} \left(d_t^{(c)} \leq \epsilon \right) \quad (18)$$

Where $d_t^{(c)}$ denotes the distance measure of a metric $c \in \mathcal{C}$ (typically Euclidean distance) at a given time t, α_c denotes a constant scaling factor that determines how heavily we weight each component, and $\mathbb{I}$ is the indicator function, scaled by a constant $\alpha_k^{(c)}$ that provides a bonus reward for being within a certain threshold of the goal. For positional rewards, our distance measure is the L2 norm of $x_t - x_g$, whereas for attitude-based rewards, this corresponds to the L2 norm of the sin and cosine of the angles. We used the change in distance as the reward, as it empirically provided better performance than using the L2 norm directly, and is standard practice in many similar problems. We used a small penalty on actions to encourage the controller to produce smoother flight.

Actions are defined to be continuous rotor speeds normalized to the interval $\Omega_i \in [0, 1]$. We assume that for most flight regimes, action inputs are clustered around the hover RPM, which is to say that – independent of the attitude of the aircraft – the thrust force will generally counteract the weight force in the inertial frame. This is a reasonable assumption; since thrust is modeled as quadratic in Ω, controlled maneuvers with small roll and pitch values won't deviate significantly from the hover condition. An equivalent statement would be to say that of the total search space, only a small subset of states and actions constitute coordinated flight, and that many of these will be "close" to hover.

We take this insight into account by constructing our action as:

$$a_t = \Omega_h + \pi_\theta(\hat{a}_t | s_t)\beta \quad (19)$$

where $\pi_\theta(\hat{a}_t | s_t)$ is the action output of the policy, and β is a bandwidth term that controls how responsive the aircraft is. We clip rotor speed within the environment to keep actions bounded, but the policy output remains unclipped. We found this led to good exploratory behaviour early in training, whilst still allowing the policy to execute the full range of actions.

Finally, the termination conditions are task-dependent, but generally constrain the the aircraft to a region of interest. The primary exception to this is the landing task, for which we take into account "crashes" and "landings" into account. The criteria for landing is that the aircraft should reach a given threshold (generally 0.1m), with an additional reward for flying close to the desired descent rate. Landings are rewarded with a large positive reward. Crashes are detected by checking for rotor collisions with the ground, and receive a negative reward. In environments where we want the aircraft to hover after reaching a given goal, we don't terminate the episode once the goal is reached, allowing the aircraft to build up bonus reward for remaining within the given threshold.

V. EXPERIMENTS

We ran experiments using GAE, PPO and TRPO on the Hover, Static Waypoint, Random Waypoint, Target Follow, and Landing tasks. We used the same hyperparameters across all tasks, with settings given in Table I. Our algorithm implementations are not overly sophisticated; our policies output both the mean and log-variance of the action as a function of the state input, and – where possible – we used "two headed" configurations with the same network outputting both the action and value estimate. We assumed actions were independent of one another (that is, we do not output a covariance matrix over actions). For TRPO, our critic was optimized using Adam and a Huber loss function, and the policy was updated using the conjugate gradient method as outlined in [19]. For PPO, we used the clipped surrogate objective as described in [**?**].

The reason for these simplifications was to gradually build up complexity. We wanted to avoid excessive tuning

of hyperparameters, and found that most implementations were able to learn decent control policies despite these simplifications.

TABLE I
POLICY SEARCH PARAMETERS

Parameter	GAE	PPO	TRPO
Hidden Neurons	128	128	128
Gamma	0.99	0.99	0.99
Lambda	0.92	0.92	0.97
Epsilon	N/A	0.1	N/A
Max KL	N/A	N/A	1e-2
Damping	N/A	N/A	1e-1
Batch Size	256	256	256
Epochs	2	4	N/A
Policy Iterations	5000	5000	5000
Learning Rate	1e-4	1e-4	1e-4

VI. RESULTS

We show results in the hover, random waypoint, target follow, and landing tasks. We omit learning curves for the Static Waypoint task, as the Random Waypoint task is a similar, more difficult, benchmark. Learning curves for the Hover task are emblematic of those on the Static Waypoint task, and we provide video footage of the Static Waypoint task at along with other results. We plot the average reward for these tasks in Fig. 2 and Fig. 2, and we present trajectory plots in Fig. 4. Video footage of our policies can be found at [address withheld].

We found that we were typically able to solve the tasks within around 5000 policy iterations, with a batch size of 256 timesteps. PPO was generally our most consistent algorithm, followed by TRPO. Interestingly, we found that TRPO produced very smooth flight performance in comparison to the other algorithms, but often converged to a slightly worse return. This is most readily apparent in the hover task, where TRPO was by far the fastest learning algorithm, but tended to produce slightly lower returns than PPO when tested. In the random waypoint task, TRPO produced the smoothest waypoint navigation, but had a harder time reaching the goal.

We are still determining the underlying causes of this phenomenon, and note recent research indicating that Adam often converges to sharper local minima [17], [18]. If this is in fact the case, we believe that the optimizer may be pushing PPO and GAE into sharper local minima that produce better returns at the cost of more erratic flight behavior. It is also possible that this is the result of TRPO's invariance to the scale of the rewards (more concretely, TRPO conducts a line search to determine the step length, whereas other methods take a step length that is proportional to the scale of the return).

Surprisingly, we found that GAE often learned faster than PPO, but was less stable over long training runs even when using single update steps. We didn't anneal the learning rate during training, and believe this to be the likely cause. The high variability of GAE generally resulted in worse policies than either PPO or TRPO, and for some tasks, it struggled to learn at all. This is strong validation of the theory underlying PPO and TRPO, and suggests that these two algorithms are more suited to learning nonlinear flight controllers.

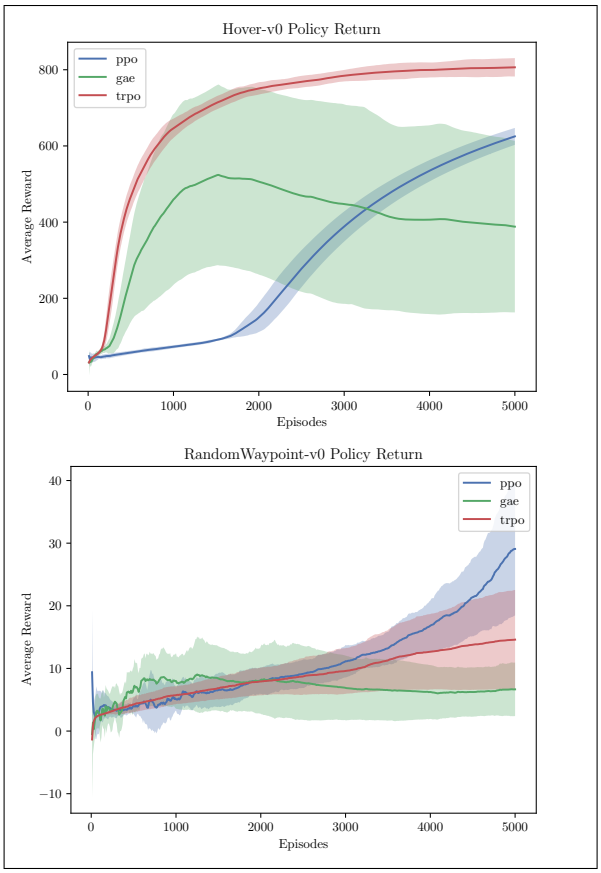

Fig. 2. Average returns for the Hover and Random Waypoint tasks over 5000 iterations (approximately 1.3 million time steps). GAE exhibited very high variance on the Hover task, and was unable to effectively learn the Random Waypoint task.

For the random waypoint task, our best controllers learned to navigate to the goal within the time-frame of the episode. We observed over-corrective behavior in some policies (notably GAE), and believe this was caused by the episodic nature of the algorithms. Monte-Carlo RL uses the empirical return to estimate $Q^{\pi}(s, a)$, meaning that the value of the terminal state is not taken into account. As a result, states towards the end of the trajectory that might result in future instability don't have as strong a value estimate as states earlier in the trajectory. This is alluded to in [6], but the overall impact of including the tail cost is not explored.

We noted that for the landing and target following tasks, the algorithms had a much harder timem with landing proved to be particularly difficult. We wanted to incentivize the aircraft to get within a small threshold of the target landing position, whilst punishing it for crashing; a consequence of this was that the policy learned to hover a small distance above the ground without reaching the threshold. This is expected behaviour since the value function averages out trajectories taken under the policy. If the aircraft receives a negative reward for crashing and a positive reward for

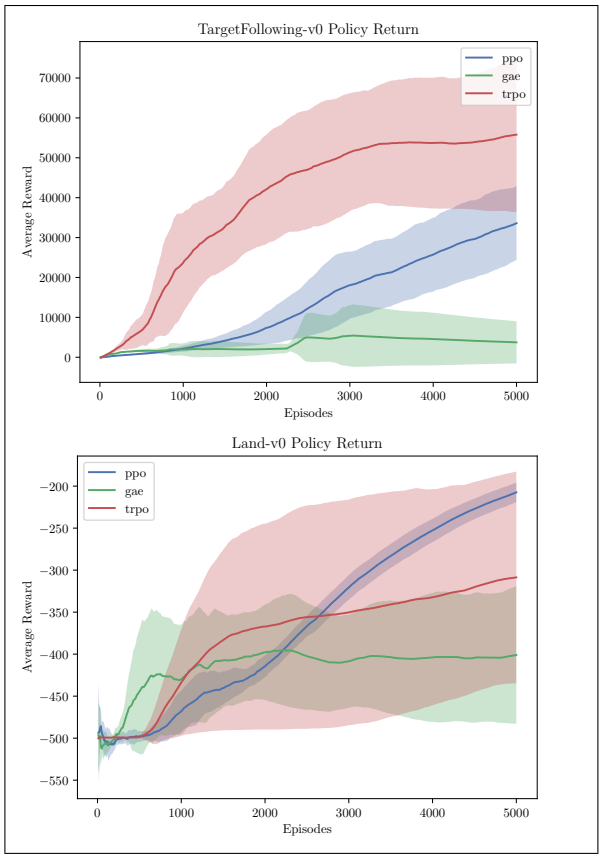

Fig. 3. Average return for the Target Following and Landing tasks over 5000 iterations (approximately 1.3 million time steps). TRPO exhibited high variance on the Land task – it wasn't able to learn an effective landing policy on all runs.

landing, there will be a trivial local minima at which these rewards are in equilibrium.

One way around this problem might be to include a discrete terminal switch that the agent learns to activate within a certain threshold of the ground. Another might be to use reward shaping using a harmonic potential reward function of the form outlined in [23], [24]. In spite of these difficulties, we were still able to learn continuous control policies capable of "landing" the aircraft without entering a vortex ring state, and our policies demonstrated smooth, controlled forward descent – and even spiral patterns – when landing. The modifications above are considered for future work.

VII. CONCLUSION

Progress in deep reinforcement learning relies on the existence of easy-to-use environments for both conducting and replicating experiments. In this paper, we have presented a set of environments for learning continuous-action quadrotor flight tasks that we hope can spur further development of intelligent flight controllers. Using this environment, we have shown that we can use deep reinforcement learning to train controllers for tasks such as hovering, static and random waypoint navigation, landing, and target following.

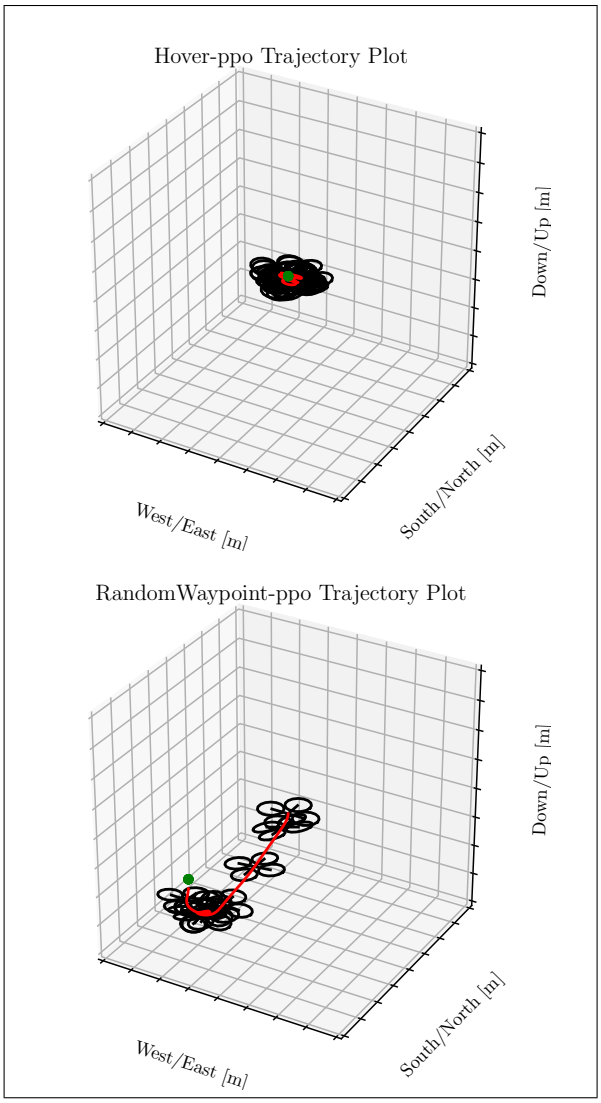

Fig. 4. Trajectory plots for PPO on the Hover and Random Waypoint tasks. Since the policy is stochastic, the aircraft always has a slight wobble when hovering in place. Video footage of the policies can be found at [address withheld].

ACKNOWLEDGMENTS

Acknowledgements withheld.

REFERENCES

[1] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D. and Riedmiller, M., 2013. Playing atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602.

[2] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G. and Petersen, S., 2015. Human-level control through deep reinforcement learning. Nature, 518(7540), p.529.

[3] Andrychowicz, M., Wolski, F., Ray, A., Schneider, J., Fong, R., Welinder, P., McGrew, B., Tobin, J., Abbeel, O.P. and Zaremba, W., 2017. Hindsight experience replay. In Advances in Neural Information Processing Systems (pp. 5048-5058).

[4] Arulkumaran, K., Deisenroth, M.P., Brundage, M. and Bharath, A.A., 2017. A brief survey of deep reinforcement learning. arXiv preprint arXiv:1708.05866.

[5] Tang, S., Kumar, V.; 2017; Autonomous Flying; Annual Review of Control, Robotics, and Autonomous Systems

[6] Hwangbo, J., Sa, I., Siegwart, R. and Hutter, M., 2017. Control of a quadrotor with reinforcement learning. IEEE Robotics and Automation Letters, 2(4), pp.2096-2103.

[7] Zhang, T., Kahn, G., Levine, S. and Abbeel, P., 2015. Learning deep control policies for autonomous aerial vehicles with mpc-guided policy search. arXiv preprint arXiv:1509.06791.

[8] Furrer, F., Burri, M., Achtelik, M. and Siegwart, R., 2016. Rotors A modular gazebo mav simulator framework. In Robot Operating System (ROS) (pp. 595-625). Springer, Cham.

[9] Kahn, G., Zhang, T., Levine, S. and Abbeel, P., 2017, May. Plato: Policy learning using adaptive trajectory optimization. In Robotics and Automation (ICRA), 2017 IEEE International Conference on (pp. 3342-3349). IEEE.

[10] Andersson, O., Wzorek, M. and Doherty, P., 2017, February. Deep Learning Quadcopter Control via Risk-Aware Active Learning. In AAAI (pp. 3812-3818).

[11] OpenAI, 2018, August. OpenAI Five. https://blog.openai.com/openai-five/

[12] Sutton, R.S. and Barto, A.G., 1998. Reinforcement learning: An introduction. MIT press.

[13] Russell, S.J. and Norvig, P., 2016. Artificial intelligence: a modern approach. Malaysia; Pearson Education Limited,.

[14] Peters, J. and Schaal, S., 2006, October. Policy gradient methods for robotics. In Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on (pp. 2219-2225). IEEE.

[15] Degris, T., White, M. and Sutton, R.S., 2012. Off-policy actor-critic. arXiv preprint arXiv:1205.4839.

[16] Wang, Z., Bapst, V., Heess, N., Mnih, V., Munos, R., Kavukcuoglu, K. and de Freitas, N., 2016. Sample efficient actor-critic with experience replay. arXiv preprint arXiv:1611.01224.

[17] Schulman, J., Moritz, P., Levine, S., Jordan, M. and Abbeel, P., 2015. High-dimensional continuous control using generalized advantage estimation. arXiv preprint arXiv:1506.02438.

[18] Schulman, J., Wolski, F., Dhariwal, P., Radford, A. and Klimov, O., 2017. Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347.

[19] Schulman, J., Levine, S., Abbeel, P., Jordan, M. and Moritz, P., 2015, June. Trust region policy optimization. In International Conference on Machine Learning (pp. 1889-1897).

[20] Schaul, T., Horgan, D., Gregor, K. and Silver, D., 2015, June. Universal value function approximators. In International Conference on Machine Learning (pp. 1312-1320).

[21] Wilson, A.C., Roelofs, R., Stern, M., Srebro, N. and Recht, B., 2017. The marginal value of adaptive gradient methods in machine learning. In Advances in Neural Information Processing Systems (pp. 4148-4158).

[22] Keskar, N.S., Mudigere, D., Nocedal, J., Smelyanskiy, M. and Tang, P.T.P., 2016. On large-batch training for deep learning: Generalization gap and sharp minima. arXiv preprint arXiv:1609.04836.

[23] Ng, A.Y., Harada, D. and Russell, S., 1999, June. Policy invariance under reward transformations: Theory and application to reward shaping. In ICML (Vol. 99, pp. 278-287).

[24] Kim, J.O. and Khosla, P.K., 1992. Real-time obstacle avoidance using harmonic potential functions. IEEE Transactions on Robotics and Automation, 8(3), pp.338-349.