Robust Deep Reinforcement Learning for Quadcopter Control

Aditya M. Deshpande, Ali A. Minai, Manish Kumar

University of Cincinnati, 2600 Clifton Ave., Cincinnati, Ohio 45221
(e-mail: deshpaad@mail.uc.edu, ali.minai@uc.edu, manish.kumar@uc.edu)

Abstract: Deep reinforcement learning (RL) has made it possible to solve complex robotics problems using neural networks as function approximators. However, the policies trained on stationary environments suffer in terms of generalization when transferred from one environment to another. In this work, we use Robust Markov Decision Processes (RMDP) to train the drone control policy, which combines ideas from Robust Control and RL. It opts for pessimistic optimization to handle potential gaps between policy transfer from one environment to another. The trained control policy is tested on the task of quadcopter positional control. RL agents were trained in a MuJoCo simulator. During testing, different environment parameters (unseen during the training) were used to validate the robustness of the trained policy for transfer from one environment to another. The robust policy outperformed the standard agents in these environments, suggesting that the added robustness increases generality and can adapt to nonstationary environments.

Keywords: Reinforcement learning control; Robust adaptive control; Robotics; Flying robots

1. INTRODUCTION

Unmanned aerial vehicles (UAVs) and their applications have shown an unprecedented increase in recent times. To name a few, drones have been used for surveillance and inspection [Bethke et al. (2008), Gonzalez et al. (2016), Gonçalves and Henriques (2015)], agriculture [Bethke et al. (2008)], package delivery [Magsino et al. (2020)], wildlife monitoring [Hodgson et al. (2016)] and search and rescue [Eid and Dol (2019)]. Furthermore, there have been continuous enhancements in quadcopter design with additional degrees of freedom such as tilt-rotor quadcopter design by Kumar et al. (2020), morphing quadcopter design by Falanga et al. (2018), and the sliding arm drone by Kumar et al. (2020) for dealing with various complex scenarios that may arise for different missions.

To enable these applications, the development of control software for UAVs, especially quadcopters, has been an active area of research [Santoso et al. (2017), Meier et al. (2015)]. The advent of deep learning and reinforcement learning has enabled development of neural controllers for these systems [Koch et al. (2019), Hwangbo et al. (2017)]. The neural policies are able to control the complex flying systems [Deshpande et al. (2020)] as well as work with high-dimensional observation spaces [Falanga et al. (2020)].

Although Deep RL has found wide applications for solving complex tasks in the UAV community, the training and adaptation of a policy for different environments remains challenging. Policies learned in stationary environments using RL are fragile and tend to fail with the change in environmental parameters such as mass, moment of inertia of the system, drag, or friction in the environment. Hence, the transfer of RL policies from one environment to another requires special attention [Zhao et al. (2020)]. Molchanov et al. (2019) and Polvara et al. (2020) used domain randomization techniques to learn stabilizing control policies for quadcopters. Domain randomization for RL
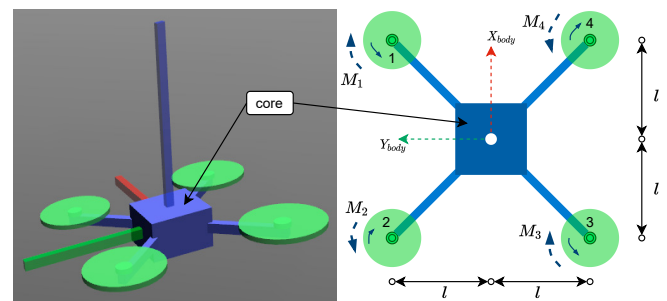

Fig. 1. Quadcopter schematic. (left) RGB-colored body-axes on mujoco model correspond to XYZ-axes in body frame respectively; (right) Top-down view of generalized quadcopter with X-configuration design.

requires maintaining distributions over the model parameters and/or perturbations which may vary across environments. This makes the policy robust only to scenarios that are encompassed by these distributions. Furthermore, these sets of distributions may not be always tractable and the range of uncertainties in which a policy would operate safely remains unknown. Therefore, the policies trained using domain randomization remain limited to within-the-distribution scenarios.

Even though domain randomization helps in generalization, most of the reinforcement learning data is sampled from simulated robot models. Due to the limitations of modeling environments, there are inconsistencies in the data from simulation and real-world even for the same task and initial conditions. Thus, the RL agents used for robot control would always experience some model mismatch and the trained agent would not perform optimally on the real robot [Ibarz et al. (2021)].

In controls engineering, robustness refers to the controller performance in case of environment and/or parametric uncertainties [Lee et al. (2013)]. Along similar lines, for RL, the chal-
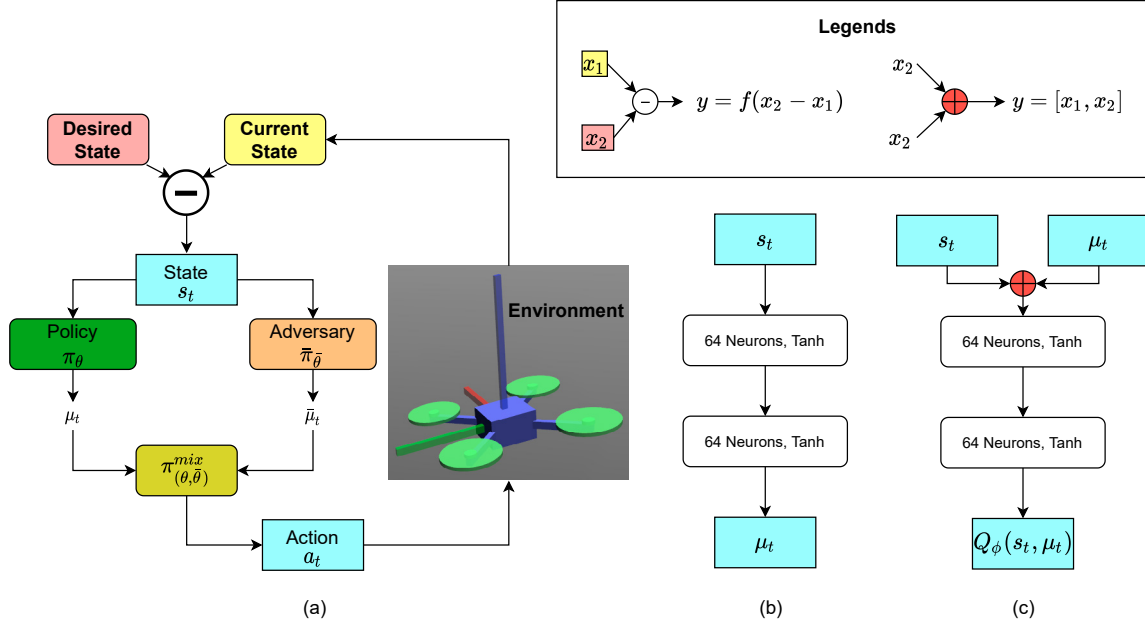
Fig. 2. Neural Network Architectures: (a) The network setup used for training robust policy where μ_t is the action by policy π_θ and $\bar{\mu}$ is the action by the adversary $\bar{\pi}_{\bar{\theta}}$; (b) Architecture of the neural networks used for policy π_θ and for adversary $\bar{\pi}_{\bar{\theta}}$; (c) Architecture of the neural network used as the critic (action-value function) $Q_\phi(s_t, \mu_t)$.

lenge of training a robust controller is addressed using Robust Markov Decision Processes [Iyengar (2005)]. In this, the RL objective is achieved using max-min optimization over a set of possible environment uncertainties. Contrary to the prior knowledge of domain uncertainties required for domain randomization approaches, the RMDP framework enables training a policy to maximize the reward with respect to the worst possible model of the environment. We adopt the robust deep reinforcement learning approach proposed by Tessler et al. (2019) to train a quadcopter control policy in continuous state and action spaces. Our results show the efficacy of this method by training a robust policy for a quadcopter waypoint navigation task considering the presence of both environment as well as parametric uncertainties.

CONTRIBUTIONS

In this article, we study the application of robust reinforcement learning to learn a robust control policy for a quadcopter. The controller is trained using the Robust Markov Decision Process (RMDP) framework. Our results demonstrate the adaptability of the robust learned policy to external and internal system uncertainties at test time. *It should be noted that the primary contribution of this work lies in the novel application of Robust-RL for learning quadcopter controller to enable functioning in non-stationary (uncertain) environmental conditions and not in the algorithm itself.*

The remainder of this article is organized as follows: Section-2 briefly describes the quadcopter dynamics, Section-3 provides the detailed approach used for learning the robust quadcopter controller and Section-4 provides the details about the setup used for training the control policies. In Section-5, we discuss our results. Finally, we conclude the article and provide future directions for research in Section-6.

2. QUADCOPTER DYNAMICS MODEL

In this section, we briefly discuss the dynamics of the quadcopter. A quadcopter of '**X**'-configuration is used in the work.

Figure 1 shows the schematic diagram of the quadcopter along with its physics engine simulated model. The body frame origin of the drone is placed at its center of mass. Thrust motors of this system rotate only in one direction and produce a positive thrust along the z_{body} axis. The translational motion of the UAV in the world frame and rotational motion in the body frame is represented by equations (1) and (2), respectively.

$$m \begin{pmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{z} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ -mg \end{pmatrix} + R \begin{pmatrix} 0 \\ 0 \\ \sum_{i=1}^{4} F_i \end{pmatrix} \tag{1}$$

$$I \begin{pmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{pmatrix} = \begin{pmatrix} l(F_1 + F_2 - F_3 - F_4) \\ l(-F_1 + F_2 + F_3 - F_4) \\ -M_1 + M_2 - M_3 + M_4 \end{pmatrix} - \begin{pmatrix} p \\ q \\ r \end{pmatrix} \times I \begin{pmatrix} p \\ q \\ r \end{pmatrix} \tag{2}$$

where $R \in SO(3)$ is the rotational matrix from body to world frame of reference and is given by (3).

$$R = \begin{bmatrix} c_\psi c_\rho & s_\xi s_\rho c_\psi - s_\psi c_\xi & s_\xi s_\psi + s_\rho c_\xi c_\psi \\ s_\psi c_\rho & s_\xi s_\psi s_\rho + c_\xi c_\psi & -s_\xi c_\psi + s_\psi s_\rho c_\xi \\ -s_\rho & s_\xi c_\rho & c_\xi c_\rho \end{bmatrix} \tag{3}$$

In equation (3), the sine and cosine angle terms are represented as s_a and c_a, respectively where a represents the roll ξ, pitch ρ and yaw ψ angles. The input thrust forces produced by each rotor are denoted by $F_i, \forall i \in 1, 2, 3, 4$. The mass of the system is represented by m and the acceleration due to gravity is represented by g. The acceleration of the system in world frame is denoted by $\ddot{x}, \ddot{y}, \ddot{z}$, and I is a diagonal matrix containing terms of moments of inertia about $x_{body} - y_{body} - z_{body}$-axes. The quadcopter length is $2 \times l$, and p, q and r are the roll-, pitch- and yaw-rates in the body frame, respectively. The torque produced by each propeller is given by $M_i, \forall i \in \{1, 2, 3, 4\}$. The thrust force and the torque produced by each motor are directly proportional to the square of the angular speed of the propeller with the constants of proportionality given by k_f and k_m, respectively [Michael et al. (2010)].

3. TRAINING ROBUST CONTROL POLICY

In this paper, we consider the case of continuous state space and a continuous action space for robots. A Markov decision process (MDP) is used to model an RL problem, and it is described by the tuple of elements (S, A, P, R, γ) where S is the state space, A is the action space, $P(s_{t+1}|s_t, a_t)$ is the probability distribution over state transitions from $s_t \in S$ to $s_{t+1} \in S$ by taking action $a_t \in A$, $R(s_t, a_t, s_{t+1})$ is the reward function, and $0 \le \gamma < 1$ is the discount factor [Sutton and Barto (2018)]. The objective of RL is to learn a control policy $\pi(a_t \mid s_t)$ to maximize the expected cumulative reward in (4), where $\tau = (s_0, a_0, s_1, a_1, \dots)$ is the trajectory sampled using policy π and the initial state is sampled from a fixed distribution $s_0 \sim p_{s_0}$.

$$
\begin{aligned}
\pi^* &= \arg\max_\pi \mathbb{E}_{\tau \sim \pi, p_{s_0}} \left[R(\tau) \right] \\
&= \arg\max_\pi \mathbb{E}_{\tau \sim \pi, p_{s_0}} \left[\sum_{t=0}^\infty \gamma^t r_t \mid \pi \right]
\end{aligned}
\tag{4}
$$

Unlike RL, for Robust Reinforcement Learning, we use a Robust Markov Decision Process (RMDP) to formulate the learning objective [Nilim and El Ghaoui (2005), Iyengar (2005)]. Mathematically, an RMDP is also modeled with a tuple of (S, A, P, R, γ), similar to an MDP, with the only exception being that the model uncertainty is incorporated in this setup by considering the transition probabilities P to be unknown but bounded in some prior known uncertainty set, i.e., $P \subseteq \mathcal{P}$. Model uncertainty is described by the set of transition models $\mathcal{P}$.

The RMDP version of the RL is an extension of the MDP by the addition of an adversarial agent. The RL agent selects actions $a_t \in A$ at each time step and aims to maximize returns. But another agent, with an adversarial nature, selects transition models from $\mathcal{P}$ each time step with an aim to minimize returns resulting in a sequential two-player zero-sum game. The resultant policy obtained in this game is given by equation (5).

$$
\pi^* = \arg\max_\pi \min_{\bar{\pi}} \mathbb{E}_{\tau \sim \pi, \bar{\pi}, p_{s_0}} \left[R(\tau) \right]
\tag{5}
$$

where π^* represents the optimal control policy. Note, here we select the optimal policy as the maximizing policy, i.e., $\pi^* = \pi$. The policies π^* (maximizer) and $\bar{\pi}^*$ (minimizer or adversary) learned after this optimization are said to be in Nash Equilibrium and neither of them may improve the game's outcome further.

To learn robust drone policies, we consider the robustness over the action uncertainty in drones. The actions correspond to thrust forces generated by the quadcopter. We use the Action Robust Deep Deterministic Policy Gradient (AR-DDPG) algorithm by Tessler et al. (2019) to train the robust policies. This approach requires only a scalar value α to describe the influence of the adversary on the action-outcome as presented in Algorithm 1. The parameter α implicitly defines an uncertainty set $\mathcal{P}$ for the model. Thus, we do not have to provide the uncertainty set explicitly while training the policy. Here, while training the control policy π_θ, the adversary $\bar{\pi}_{\bar{\theta}}$ takes the control and acts on the system with probability α (step 7 in Algorithm 1). In the case of the drone, this adversarial attack can be thought of as noise or external disturbances such as wind or model uncertainties such as errors in system identification. Thus, when the controller takes some action, the adversary makes the system act differently than expected. Thus, the optimization strategy of AR-DDPG used here enables the training of a robust optimal policy with respect to uncertain scenarios. The objective of the work reported here is to demonstrate the application of robust RL for training drone control policies. Readers are referred

---

Algorithm 1: Action Robust Deep Deterministic Policy Gradient Pseudo code

Input: Actor (a.k.a. Policy) training steps n, uncertainty value α and discount factor γ, maximum number of episodes N, maximum length of each episode T, learning rate η

1   Randomly initialize actor, adversary and critic networks as π_θ, $\bar{\pi}_{\bar{\theta}}$ and $Q_\phi(s, a)$ respectively.

2   Initialize corresponding target network weights as: $\theta^{target} \leftarrow \theta$, $\bar{\theta}^{target} \leftarrow \bar{\theta}$ and $\phi^{target} \leftarrow \phi$.

3   Initialize replay buffer $\mathcal{R}$

4   **for** $k = \{0, 1, \dots, N\}$ **do**

5   Initial state $s_0 \sim p_{s_0}$.

6   **for** $t = \{0, \dots, T\}$ **do**

7   Action $a_t = \begin{cases} \pi(s_t), & \text{with probability } (1 - \alpha) \\ \bar{\pi}(s_t), & \text{otherwise} \end{cases}$

8       $\tilde{a}_t = a_t + $ exploration noise

9   Execute $\tilde{a}_t$, observe next state s_{t+1}, reward r_t and episode termination flag d.

10   Dump $(s_t, a_t, s_{t+1}, r_t, d)$ in $\mathcal{R}$.

11   **for** $i = \{0, \dots, n\}$ **do**

12   Sample batch $B = \{(s, a, s', r, d)\}$ from $\mathcal{R}$.

13   Update θ by one step gradient ascent using $\nabla_\theta \frac{1}{|B|}(1 - \alpha) \sum_{s \in B} Q_\phi(s, \pi_\theta(s))$.

14   Compute targets: $\tilde{Q} = (1-\alpha)Q_{\phi^{target}}(s', \pi_{\theta^{target}}(s')) + \alpha Q_{\phi^{target}}(s', \bar{\pi}_{\bar{\theta}^{target}}(s'))$; $y(r, s', d) = r + \gamma(1 - d)\tilde{Q}$

15   Update ϕ by one step gradient descent using $\nabla_\phi \frac{1}{|B|} \sum_{(s,a,s',r,d) \in B} \left(Q_\phi(s, a) - y(r, s', d) \right)$

16   Sample batch $B = \{(s, a, s', r, d)\}$ from $\mathcal{R}$.

17   Update $\bar{\theta}$ by one step gradient descent using $\nabla_{\bar{\theta}} \frac{1}{|B|} \alpha Q_\phi(s, \bar{\pi}_{\bar{\theta}}(s))$

18   Compute targets: $\tilde{Q} = (1-\alpha)Q_{\phi^{target}}(s', \pi_{\theta^{target}}(s')) + \alpha Q_{\phi^{target}}(s', \bar{\pi}_{\bar{\theta}^{target}}(s'))$; $y(r, s', d) = r + \gamma(1 - d)\tilde{Q}$

19   Update ϕ by one step gradient descent using $\nabla_\phi \frac{1}{|B|} \sum_{(s,a,s',r,d) \in B} \left(Q_\phi(s, a) - y(r, s', d) \right)$

20   Update target networks: $\theta^{target} \leftarrow \tau\theta + (1 - \tau)\theta^{target}$ $\bar{\theta}^{target} \leftarrow \tau\bar{\theta} + (1 - \tau)\bar{\theta}^{target}$ $\phi^{target} \leftarrow \tau\phi + (1 - \tau)\phi^{target}$

---

to Tessler et al. (2019) and Lillicrap et al. (2016) for further mathematical elaboration of the action-robust MDPs and deep deterministic policy gradients.

We trained the control policy in an off-policy manner. Although the adversary $\bar{\pi}$ induces a stochastic setting for training the policy (step 7 in Algorithm 1), noise was introduced into the action space to encourage exploration (step 8 in Algorithm 1). We use time-correlated Ornstein–Uhlenbeck noise for exploration as recommended by Lillicrap et al. (2016). Furthermore, to improve exploration at the start of training, we allow babbling of the quadcopter for 500 episodes where, in each episode, the environment state is initialized randomly as described in 3.2,

and the actions are sampled from a uniform distribution in the range $[-1, 1]$ and scaled appropriately before being applied to the system (Eq. (8)).

The neural networks were trained in the actor-critic style. Figure 2 shows the setup and architectures of the neural networks used for learning the policy for the quadcopter. Figure 2-(a) illustrates the policy setup used during the training where

$$\pi_{\theta,\bar{\theta}}^{mix} = \begin{cases} \pi(s_t), & \text{with probability } (1 - \alpha) \\ \bar{\pi}(s_t), & \text{otherwise} \end{cases} \quad (6)$$

The architectures of the control policy and the adversary are identical and shown in Fig. 2-(b) and the critic or action-value function $Q(s, \mu)$ architecture is shown in Fig. 2-(c). We did not apply any neural architecture optimization and did not consider the effects of variation in activation functions in the present work. The state or observation space used for learning quadcopter control is given by the tuple $s_t = (e_{p_t}, e_{v_t}, R_t, e_{\omega_t})$. The control policy maps the current state of the system to the actuator commands represented by the vector $a_t = \{F_{1,t}, F_{2,t}, F_{3,t}, F_{4,t}\}$. We drop the subscript of time t to avoid clutter of notation. Here, $e_p \in \mathbb{R}^3$ is the position error, $e_v \in \mathbb{R}^3$ is the velocity error, $e_\omega \in \mathbb{R}^3$ is the error in body rates, and R is the flattened 3×3 rotational matrix (R is converted to a 9-dimensional vector by flattening). Thus, the input state vector to the policy π and the adversary $\bar{\pi}$ is of 18 dimensions. For the quadcopter system under consideration, the action space $a \in \mathbb{R}^4$ corresponds to the number of thrust motors, and this also forms the 4-dimension output vector for the policy and adversary networks. The output values are centered at the hovering condition of the quadcopter given by the equation (7).

$$F_h = \frac{mg}{4} \quad (7)$$

where F_h is the required thrust force by each rotor for hovering state of the quadcopter also shown by Michael et al. (2010). We scale the policy (and adversary) network outputs from $[-1, 1]$ to the corresponding actuator limits before applying it to the system (a tanh activation generates outputs in [-1, 1] - see Fig. 2-(b)). A linear mapping is used to convert neural outputs to quadcopter actuator thrusts given by (8) [Deshpande et al. (2020)].

$$F_i = F_h + \frac{a_i(F_{max} - F_{min})}{2} \quad (8)$$

where F_{min} and F_{max} are the minimum and maximum thrust forces possible for the actuators respectively, and $i \in \{1, 2, 3, 4\}$ represents the thrust actuator index.

3.1 Reward Function

We train the drone to reach a fixed waypoint. This waypoint is defined in the world coordinate frame. The desired linear velocity, attitude and body rates are zero at the goal location. The reward earned by the system during each time step is given by equation (9).

$$r_t = \beta - \alpha_a \|a\|_2 - \sum_{k \in \{p, v, \omega\}} \alpha_k \|e_k\|_2 - \sum_{j \in \{\xi, \rho\}} \alpha_j \|e_j\|_2 \quad (9)$$

where $\beta \geq 0$ is a reward for staying alive and coefficients $\alpha_{\{\cdot\}} \geq 0$ represent the weights of various terms in (9). The second term in (9) represents the penalty imposed to avoid excessive actions, followed by the terms representing the cost of error in the quadcopter state. The third term is based on the position error e_p, the velocity error e_v, and the error in body rates e_ω. The errors in roll and pitch angles, given by e_ξ and

e_ρ, respectively, are represented in the last term. The RL agent was not penalized for yaw error to prioritize the learning of waypoint navigation irrespective of the drone heading.

3.2 Training Strategy

We train the control policy to maximize the expected return for the waypoint navigation task. The initial state of the system was sampled randomly from uniform distribution p_{s_0}. This enables learning the control policy independent of the initial state of the quadcopter. The sampling strategy described by Deshpande et al. (2020) was used to initialize the environment for each episode during the training. The drone was trained to reach and hover at fixed waypoint $(0, 0, 5m)$ in the world frame. The initial position (x, y, z) of the quadcopter was sampled uniformly from the cube spanning $2m$ around this location. Initial magnitudes of linear velocity (in m/s) and angular velocity (in rad/s) were sampled from a uniform distribution in the range of $[0, 1]$. The quadcopter may not be able to recover from certain initial states such as a completely upside down initial orientation, so the uniform sampling of UAV orientation from $SO(3)$ makes the policy training difficult and convergence took longer. Thus, roll, pitch and yaw angles were drawn uniformly from bounded range $[-\frac{\pi}{3}, \frac{\pi}{3}]$. The quadcopter desired orientation, linear velocities and angular velocities were kept zero for the task of waypoint navigation. Each episode in training was terminated if its length exceeded the maximum number of time-steps set in the simulation. Additionally, the episode was terminated if the robot went beyond the bounds defined in the simulation. These bounds were defined as the cube of $2m$ around its target position.

4. ENVIRONMENT SETUP

We used an RL environment as developed by Deshpande et al. (2020) for training the control policy. The environment was built using the Mujoco physics engine [Todorov et al. (2012)] and OpenAI Gym [Brockman et al. (2016)]. Table-1 provides the physical parameters used to simulate this robot. The coefficients in equations (9) were: $\alpha_p = 1.0$, $\alpha_v = 0.05$, $\alpha_\omega = 0.001$, $(\alpha_\xi, \alpha_\rho) = (0.02, 0.02)$, $\beta = 2.0$, $\alpha_a = 0.025$. The simulations were performed on a machine with an Intel-i7 processor, 16GB RAM, and NVIDIA RTX 2070. The training of the policy developed using Pytorch[1] took approximately 2 hours. The values of hyperparameters used in AR-DDPG are provided in Table-2. We used the Adam optimizer [Kingma and Ba (2014)] to optimize neural networks.

Table 1. Quadcopter physics model parameters

Parameter	Value
Timestep	0.01 sec
Motor lag	0.001 sec
Mass m	1.5 kg
Length $2l$	0.26 m
Motor thrust range $[F_{min}, F_{max}]$	[0, 15.0] N

5. RESULTS

We trained two sets of policies, one using robust RL with AR-DDPG and the other non-robust RL using DDPG. In the case of plain DDPG, no adversary was used and the other parameters were kept the same as shown in Table-2. Although $0 \leq \alpha \leq 1$,
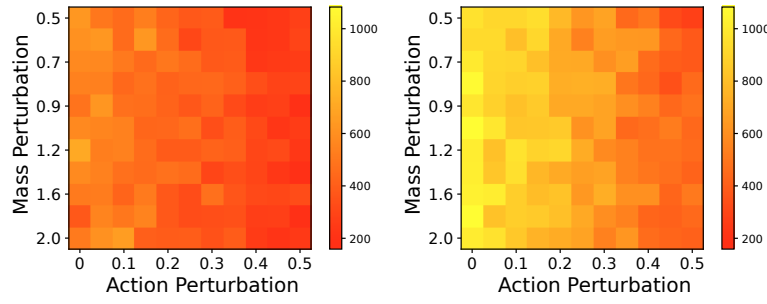
---

[1] https://pytorch.org/

Fig. 3. Comparison of policies trained using DDPG (left) and AR-DDPG (right); the color maps represent the average episode return over 10 episodes for corresponding tuple of (Mass Perturbation, Action Perturbation).

Table 2. AR-DDPG Hyperparameters

Parameter	Value
Training iterations	2×10^6
Learning rate $\eta_{\pi,\bar{\pi}}$ for policy and adversary networks	$2e-5$
Learning rate η_Q for critic network	$2e-4$
Neural network optimizer	Adam
Adam parameters (β_1, β_2)	$(0.9, 0.999)$
Discount factor γ	0.95
Max. episode length T	1500
Batch size $|B|$	64
Adversary action probability α	0.1
Replay buffer size $|\mathcal{R}|$	8×10^5
Policy training steps n	20

Table 3. Test parameters

Parameter	Set of Values
Quadcopter relative mass $\frac{m_{test}}{m_{train}}$ (Mass Perturbations)	{0.5, 0.6, 0.7, 0.8, 0.9, 1.0, 1.2, 1.4, 1.6, 1.8, 2.0}
Probability of external perturbations δ (Action Perturbations)	{0, 0.05, 0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4, 0.45, 0.5}

we chose $\alpha = 0.1$. For higher values of α, the adversary gets more control and training a policy becomes difficult. This was also reported by Tessler et al. (2019). To check the robustness of the control policy trained using the proposed method of AR-DDPG for the UAV, we measured the performance of the drone for waypoint navigation using the reward function (9) to evaluate the cumulative reward per episode. While testing this policy, both the external and internal parameters to the quadcopter were changed while the trained policy weights were kept unchanged. We did not use the adversary network or the critic network in the testing phase.

We chose the drone mass as the internal parameter and perturbation noise added to the actions as the external parameter to the system. The change in mass simulates the scenarios of the quadcopter system identification errors. We modify the mass of the system *core* shown in Fig. 1 while testing our policies. The appropriate value was added or subtracted from the mass of the *core* while testing. The external perturbations inserted in the actions can be viewed as the effect of wind on the drone. Table-3 represent the values of the parameters used for testing the policies. Here, the relative mass is the ratio of the quadcopter masses used during testing m_{test} and training m_{train}, where the $m_{train} = m$ is given in Table-1. The external perturbations were sampled from a uniform distribution and included in actions from trained policy as $a_i = a_i + \text{Uniform}(-1, 1) \ \forall i \in \{1, 2, 3, 4\}$ with probability δ (given in Table-3). It should be noted that the values in this table were not shown to the system during training.

Figure 3 contains the color-maps of the results representing the averaged episode returns over all test runs of the trained policies. For each episode i, the episode return is calculated as $R_i = \sum_{t=0}^{t_{final}} r_{t,i}$ and $r_{t,i}$ is obtained using equation (9). For each

value of the external and internal parameters, the parameters of the policies trained using AR-DDPG and DDPG were kept frozen. The drone was then initialized at a state sampled using the strategy described in Section-3.2 and allowed to perform the task of reaching the goal location. This test was performed 10 times for each pair of internal and external parameters. It can be observed from these results that the robust policy trained using AR-DDPG acquires higher rewards compared to the DDPG policy when the environment parameters are varied from the parameters seen by the policy during training. This clearly shows the robustness of the trained neural policy to unseen environmental factors. The robust policy was able to perform comparatively better than the policy trained using DDPG even when no internal or external perturbations are introduced in the environment. Another interesting point to note from these results is that the training of robust policy did not require any knowledge about the model uncertainties in set $\mathcal{P}$ except the scalar value α. The adversary $\bar{\pi}$, trained for taking the worst possible action, allowed the controller to be implicitly prepared for the range of uncertainties that may arise in the drone model.

6. CONCLUSION AND FUTURE WORK

In this work, we investigated robust deep reinforcement learning for training control policies for quadcopters. We compared the robust control policy trained using AR-DDPG with the policy trained with the DDPG algorithm. The results suggest that the learned robust policies are able to account for uncertainties in the drone model that are not considered at training time. The results also show that the policy is able to perform better compared to the non-robust policy even in the absence of noise and achieve a higher reward. The drone trained with this approach would also be tolerant to external disturbances in the environment such as wind. Although the value of α implicitly considers uncertainty set for the model, the robust policy may not be able to handle any kind of uncertainty. This is also observable from the results in Fig. 3 where the increase in uncertainty degraded robust policy performance. Thus, depending on the application, one may consider domain randomization as a complement to robust RL.

The approach of using adversarial agents in reinforcement learning could be helpful to mitigate the gap in simulation to

real-world transfer of learning-based approaches used to train robots. Robust control policies could account for model uncertainties making the simulation-to-real-world transfer of agents implicitly possible [Loquercio et al. (2019)]. This work only investigated the symbolic state space for the drones. It would be interesting to see the use of this approach in applications such as intelligent visual servoing using aerial vehicles [Bonatti et al. (2020)] that would consist of high dimensional raw sensor data such as images.

REFERENCES

Bethke, B., How, J.P., and Vian, J. (2008). Group health management of UAV teams with applications to persistent surveillance. In *2008 American Control Conference*, 3145–3150. IEEE.

Bonatti, R., Madaan, R., Vineet, V., Scherer, S., and Kapoor, A. (2020). Learning Visuomotor Policies for Aerial Navigation Using Cross-Modal Representations. *arXiv preprint arXiv:1909.06993*.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. (2016). OpenAI Gym.

Deshpande, A.M., Kumar, R., Minai, A.A., and Kumar, M. (2020). Developmental Reinforcement Learning of Control Policy of a Quadcopter UAV With Thrust Vectoring Rotors. In *Dynamic Systems and Control Conference*. doi: 10.1115/DSCC2020-3319. URL https://doi.org/10.1115/DSCC2020-3319.

Eid, S.E. and Dol, S.S. (2019). Design and development of lightweight-high endurance unmanned aerial vehicle for offshore search and rescue operation. In *2019 Advances in Science and Engineering Technology International Conferences (ASET)*, 1–5. IEEE.

Falanga, D., Kleber, K., Mintchev, S., Floreano, D., and Scaramuzza, D. (2018). The foldable drone: A morphing quadrotor that can squeeze and fly. *IEEE Robotics and Automation Letters*, 4(2), 209–216.

Falanga, D., Kleber, K., and Scaramuzza, D. (2020). Dynamic obstacle avoidance for quadrotors with event cameras. *Science Robotics*, 5(40).

Gonçalves, J. and Henriques, R. (2015). UAV photogrammetry for topographic monitoring of coastal areas. *ISPRS Journal of Photogrammetry and Remote Sensing*, 104, 101–111.

Gonzalez, L.F., Montes, G.A., Puig, E., Johnson, S., Mengersen, K., and Gaston, K.J. (2016). Unmanned aerial vehicles (UAVs) and artificial intelligence revolutionizing wildlife monitoring and conservation. *Sensors*, 16(1), 97.

Hodgson, J.C., Baylis, S.M., Mott, R., Herrod, A., and Clarke, R.H. (2016). Precision wildlife monitoring using unmanned aerial vehicles. *Scientific reports*, 6(1), 1–7.

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

Ibarz, J., Tan, J., Finn, C., Kalakrishnan, M., Pastor, P., and Levine, S. (2021). How to train your robot with deep reinforcement learning: lessons we have learned. *The International Journal of Robotics Research*. URL https://doi.org/10.1177/0278364920987859.

Iyengar, G.N. (2005). Robust dynamic programming. *Mathematics of Operations Research*, 30(2), 257–280.

Kingma, D.P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Koch, W., Mancuso, R., West, R., and Bestavros, A. (2019). Reinforcement learning for uav attitude control. *ACM Transactions on Cyber-Physical Systems*, 3(2), 1–21.

Kumar, R., Deshpande, A.M., Wells, J.Z., and Kumar, M. (2020). Flight Control of Sliding Arm Quadcopter with Dynamic Structural Parameters. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 1358–1363. doi:10.1109/IROS45743.2020.9340694.

Kumar, R., Bhargavapuri, M., Deshpande, A.M., Sridhar, S., Cohen, K., and Kumar, M. (2020). Quaternion Feedback Based Autonomous Control of a Quadcopter UAV with Thrust Vectoring Rotors. In *2020 American Control Conference (ACC)*, 3828–3833. IEEE.

Lee, T., Leok, M., and McClamroch, N.H. (2013). Nonlinear robust tracking control of a quadrotor UAV on SE (3). *Asian Journal of Control*, 15(2), 391–408.

Lillicrap, T., Hunt, J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. (2016). Continuous control with deep reinforcement learning. *CoRR*, abs/1509.02971.

Loquercio, A., Kaufmann, E., Ranftl, R., Dosovitskiy, A., Koltun, V., and Scaramuzza, D. (2019). Deep drone racing: From simulation to reality with domain randomization. *IEEE Transactions on Robotics*, 36(1), 1–14.

Magsino, E.R., Say, M.F., and Tan, J.A. (2020). Achieving Complete UAV Delivery in the Presence of Motor Failures. In *2020 IEEE 10th Symposium on Computer Applications & Industrial Electronics (ISCAIE)*, 1–5. IEEE.

Meier, L., Honegger, D., and Pollefeys, M. (2015). PX4: A node-based multithreaded open source robotics framework for deeply embedded platforms. In *2015 IEEE international conference on robotics and automation (ICRA)*, 6235–6240. IEEE.

Michael, N., Mellinger, D., Lindsey, Q., and Kumar, V. (2010). The GRASP multiple micro-UAV testbed. *Robotics & Automation Magazine, IEEE*, 17(3), 56–65.

Molchanov, A., Chen, T., Hönig, W., Preiss, J.A., Ayanian, N., and Sukhatme, G.S. (2019). Sim-to-(Multi)-Real: Transfer of low-level robust control policies to multiple quadrotors. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 59–66. doi:10.1109/IROS40897.2019.8967695.

Nilim, A. and El Ghaoui, L. (2005). Robust control of Markov decision processes with uncertain transition matrices. *Operations Research*, 53(5), 780–798.

Polvara, R., Patacchiola, M., Hanheide, M., and Neumann, G. (2020). Sim-to-Real quadrotor landing via sequential deep Q-Networks and domain randomization. *Robotics*, 9(1), 8.

Santoso, F., Garratt, M.A., and Anavatti, S.G. (2017). State-of-the-art intelligent flight control systems in unmanned aerial vehicles. *IEEE Transactions on Automation Science and Engineering*, 15(2), 613–627.

Sutton, R.S. and Barto, A.G. (2018). *Reinforcement learning: An introduction*. MIT press.

Tessler, C., Efroni, Y., and Mannor, S. (2019). Action robust reinforcement learning and applications in continuous control. In *International Conference on Machine Learning*, 6215–6224. PMLR.

Todorov, E., Erez, T., and Tassa, Y. (2012). Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 5026–5033. IEEE.

Zhao, W., Queralta, J.P., and Westerlund, T. (2020). Sim-to-real transfer in deep reinforcement learning for robotics: a survey. In *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*, 737–744. IEEE.