Journal of Unmanned Vehicle Systems

Quadrotor Motion Control Using Deep Reinforcement Learning

Journal:	*Journal of Unmanned Vehicle Systems*
Manuscript ID	juvs-2021-0010.R2
Manuscript Type:	Article
Date Submitted by the Author:	15-Sep-2021
Complete List of Authors:	Jiang, Zifei; University of Alberta Department of Electrical and Computer Engineering Lynch, Alan; University of Alberta, Department of Electrical and Computer Engineering
Keyword:	Motion Control
Is the invited manuscript for consideration in a Special Issue, Collection, or competition? :	Unmanned Systems Canada student paper competition

Quadrotor Motion Control Using Deep Reinforcement Learning

Zifei Jiang and Alan F. Lynch

Abstract: We present a deep neural net-based controller trained by a model-free reinforcement learning (RL) algorithm to achieve hover stabilization for a quadrotor unmanned aerial vehicle (UAV). With RL, two neural nets are trained. One neural net is used as a stochastic controller which gives the distribution of control inputs. The other maps the UAV state to a scalar which estimates the reward of the controller. A proximal policy optimization (PPO) method, which is an actor-critic policy gradient approach, is used to train the neural nets. Simulation results show that the trained controller achieves a comparable level of performance to a manually-tuned PID controller, despite not depending on any model information. The paper considers different choices of reward function and their influence on controller performance.

1. Introduction

The problem of motion control for UAVs is a primary area of research which continues to generate interest. A survey of the area is given in Valavanis and Vachtsevanos (2015). The interest in motion control comes from a practical need for high performance trajectory tracking which is robust to external force disturbances and changing dynamics. For example, external disturbance forces arise in load transportation (Villa et al., 2019) or during non-destructive contact inspection (Ruggiero et al., 2018). UAV dynamics can change when rotor thrust is affected by nearby obstacles (Bangura, 2017). Other sources of challenge include underactuated nonlinear vehicle dynamics Hua et al. (2013), input bounds (Cao and Lynch, 2015), noise/delay in measurements and state estimates (Cao and Lynch, 2020, 2021; Dong et al., 2014), and limited sensing with autonomous motion control, *e.g.* position tracking in GPS denied environments or motion control relative to visual targets (Tang and Kumar, 2018). Much of the work on UAV motion control is traditionally model-based and the design procedure relies on a physics-based dynamics. Traditionally, a control law has a fixed structure influenced by the model equations and which has adjustable gains that affect the closed-loop performance. Tuning these gains provides a practical implementation challenge. Although a rigourous statement regarding convergence is typical in traditional control, it often holds under ideal modelling assumptions. Such shortcomings of traditional methods has led to interest in RL being applied to UAV motion control. RL has the advantage of requiring less domain knowledge in that the design method can be applied to a generic system. For example, the proposed method uses only simulation data to train the control law or policy which maximizes an accumulated reward to minimize regulation error. No particular controller structure is needed in advance other than it be a static state feedback. As well, controller tuning is performed automatically during training.

Reinforcement learning (RL) is concerned with how agents act within an environment in order to

Zifei Jiang and Alan F. Lynch.[*] Department of Electrical and Computer Engineering, University of Alberta, Edmonton, AB, Canada T6G 2V4 *Corresponding Author: alan.lynch@ualberta.ca

unknown **99**: 1–20 (2021) DOI: 10.1139/*Zxx-xxx*

maximize a scalar cumulative reward which measures long term performance. RL enables a robot to autonomously discover optimal behaviour through trial and error interactions with its environment. Recent research has shown impressive performance from so-called *Deep RL*, where "deep" relates to the use of neural nets (NNs) with multiple layers. Deep RL has been shown to outperform human experts in complex tasks. For example, playing Go (Silver et al., 2017), Atari games (Schulman et al., 2017), and solving a Rubik's Cube (OpenAI et al., 2019). These examples involve action and state spaces which are discrete. Although RL has been used in robotics for decades, it has mainly been confined to the high-level decisions (*e.g*. trajectory planning (Kober et al., 2013)) rather than control of low-level actuators. This is because high-level decisions are easier to discretize and thus admit tabular methods. Recently, it has been shown that deep RL can also be applied to continuous action and state spaces in robotics (Lillicrap et al., 2015) due to improved computational power.

Existing work on RL applied to UAVs can be divided into model-based and model-free methods. Model-based refers to learning an optimal controller indirectly by learning a model which can predict the future system state given the current state and input. Model-free methods learn an optimal controller directly by observing the reward and system state for a given input. Both model-based and model-free methods have attracted much attention in recently year. We begin by surveying some of the relevant model-based approaches. Earlier work (Abbeel et al., 2010) considers a design in three stages where initially an apprenticeship learning algorithm extracts a reference trajectory for the UAV from suboptimal expert pilot demonstrations. Secondly, a full nonlinear dynamic model for a helicopter UAV is identified. Thirdly, a receding horizon variation of linear quadratic control based on a linearized model is applied to the identified dynamics. Experimental results are provided. Work (Lambert et al., 2019) focuses on inner loop control which maps IMU measurements to PWM waveforms for the Electronic Speed Controllers (ESCs). During training, a NN identifies a model of the rotational dynamics which is combined with a Random Shooter Model Predictive Control (MPC) which finds an optimal control by simulation. Experimental results are provided on the Crazyflie platform and short term hover performance is demonstrated in experiment. No position loop control is considered. Work (Helder et al., 2021) uses the method in Zhou et al. (2018) to design two separate Incremental Dual Heuristic Programming (IDHP) controllers for collective and longitudinal cyclic pitch inputs of a full-sized helicopter. These inputs control the altitude and pitch Degrees-of-Freedom (DoF). The outer lateral position DoF loops and roll/yaw control are PID controllers. The reduced number of DoF controlled by RL control leads to faster learning of simpler dynamics with reduced coupling. An "incremental" linear model is estimated during training which corresponds to the linearization of the system dynamics about its trajectory. A detailed simulation model is used to train the controller and test it's performance. Work (Kaufmann et al., 2020) presents an end-to-end RL method for the full system dynamics with only IMU and a single monocular camera as sensor input. It uses an abstraction of camera images and IMU measurements in order to improve transfer from simulation to the real-world. These processed measurements are fed to a NN which is trained in a simulator to imitate MPC expert demonstrations with privileged simulation data (*i.e*. the full UAV state). Experimental results are demonstrated for aggressive trajectories. From the above we observe model-based RL has evolved to eliminate the need for expert demonstration. The adoption of NNs in model-based RL has allowed the solution of more complex problems such as end-to-end vision based control (Kaufmann et al., 2020).

Model-free RL has been investigated recently to develop NN-based controllers. These approaches are *model-free* as they do not try to estimate the transition dynamics associated with the Markov Decision Process (MDP). Representative work in this area is deep Q-learning (DQN) (Mnih et al., 2013), deterministic policy gradient methods (DDPG) (Silver et al., 2014), trust region policy gradient (TRPO) (Schulman et al., 2015b), and proximal policy optimization (PPO) (Schulman et al., 2017). Often model-free RL is a *policy-gradient method*. This means a parameterized policy (or control law) is optimized using gradient descent. The PPO method is a policy-gradient method and is used in this paper. This method was chosen based on recent demonstrated performance (Schulman et al., 2017).

A number of researchers have applied model-free RL to UAV motion control. Model-free approaches are often considered less computational efficient than model-based methods. However, model-free methods are generally easier to implement and currently receive more attention in the RL community. Work (Polvara et al., 2018) applies Q-learning to the UAV landing problem. The controller outputs high-level commands (*i.e.* forward, backward, left, right, down) as opposed to low level physical inputs (*i.e.* ESC PWM). Work (Bøhn et al., 2019) uses the PPO method to solve the attitude control of fixed-wing UAV. The attitude dynamics are inherently stable which makes the problem less challenging than quadrotor control. Work (Zhang et al., 2021) improves the DDPG algorithm by using a double experience replay buffer for training (DERB). This DERB-DDPG control is trained using a linearized outer loop system dynamics. Simulation results are presented using an inner-outer loop control. Work (Hwangbo et al., 2017) proposes an improvement on the DDPG algorithm by accurately estimating the gradient of the objective function w.r.t. action. The authors use a Singular Value Decomposition (SVD) method to find the exact solution of the Hessian matrix inverse. This value is needed to update the actor NN parameters. Motion control is provided for the full quadrotor dynamics. A PD controller for attitude control is used to assist the learning and testing performance. Experiments show hover stabilization for a wide range of initial states.

The proposed PPO-based method is unique relative to existing approaches in that solves the motion control problem for the full dynamics with the control output being low-level system inputs: total thrust and torque. Actuating low level inputs and controlling all six UAV DoF is more challenging than controlling a subset of DoF as in (Zhang et al., 2021; Lambert et al., 2019). As well, no traditional control is used to assist the RL during training or testing. This should be compared to work Hwangbo et al. (2017) where PID assists the RL. Our approach investigates the choice of reward function on time-domain tracking performance. Previous RL work does not focus on this design parameter which we show is useful in improving actual closed-loop performance. Simulation results show hover and trajectory tracking performance which is comparable to a manually tuned inner-outer loop PD control. Hence, the method benefits from not requiring a tuning stage as in traditional control. Section 2 provides preliminary information on the method. Section 3 presents the details of PPO method. Section 4 presents simulations results. Section 5 presents the conclusion. The code and simulation data for this project are available at `https://github.com/ANCL/QuadPPO`.

2. PRELIMINARIES

2.1. Markov Decision Process

Fig. 1 shows the closed-loop system in an RL framework. We follow the notation of Sutton and Barto (2018). The learner or decision-maker is called an *agent*, which can be thought of as the controller in control systems. We remark that with the RL method considered here, the system operates in a training mode in which the controller design is performed adaptively based on system measurements. After this design stage is completed normal operation begins where the controller's parameters are fixed. The open-loop system or plant together with any disturbances is called the *environment*. The agent computes an action which influences the environment state. The environment also provides a reward signal which the agent maximizes over time. From a control systems perspective, the reward is included in the reference output which is usually generated in the controller. A discrete-time framework is adopted here as is customary with RL methods. For every time $t = 0, 1, 2, \ldots$, the agent measures the environment state $S_t \in \mathcal{S}$ in order to compute an action $A_t \in \mathcal{A}$. Here, S_t, A_t denote random variables for each t. We take the state space $\mathcal{S} = \mathbb{R}^n$, action space $\mathcal{A} = \mathbb{R}^m$, and reward space $\mathcal{R} = \mathbb{R}$ where $n = 12, m = 4$ for the case of the quadrotor. A Markov Decision Process (MDP) is normally used to model the dynamics of the environment. The MDP is described by the function $p : \mathcal{S} \times \mathcal{R} \times \mathcal{S} \times \mathcal{A} \to [0, 1]$ where

$$p(s', r|s, a) = \Pr\{S_t = s', R_t = r|S_{t-1} = s, A_{t-1} = a\} \tag{1}$$

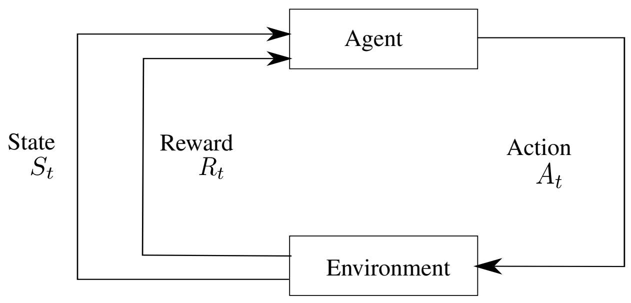


Fig. 1. The agent-environment interaction

where $s', s \in \mathcal{S}$, s' is the next state, $r \in \mathcal{R}$, $a \in \mathcal{A}$, and Pr denotes probability. Hence, (1) determines the probability a current state and reward occurs given a certain action and state at the previous time. Although quadrotor motion control is normally derived based on a deterministic ODE model, derivation of an RL method is performed in a stochastic framework.

2.2. Policy and Value Function

In this paper we use a stochastic policy $\pi(a|s)$ which is the probability that $A_t = a$ if $S_t = s$. In control systems terminology, policy π corresponds to the control law which is derived from the design method. In deep RL, the policy is parameterized using a neural net with a vector of parameters θ. The parameterized policy is $\pi_\theta(a|s)$ and the parameterization is described below in Section 3.2. The state-value function $v_\pi(s)$ of a state s under a policy π is the expected accumulated reward when starting in s and following policy π:

$$v_\pi(s) = \mathbb{E}_\pi[\sum_{t=0}^{\infty} \gamma^t R_t | S_0 = s], \text{for all } s \in \mathcal{S} \tag{2}$$

where $\gamma < 1$ is the discounting factor and $\mathbb{E}_\pi[\cdot]$ denotes the expected accumulated reward following policy π.

Similar to v_π we introduce the action-value function $q_\pi(s, a)$ for policy π as

$$q_\pi(s, a) = \mathbb{E}_\pi[\sum_{t=0}^{\infty} \gamma^t R_t | S_0 = s, A_0 = a] \tag{3}$$

This function is the expected accumulated reward when starting in s, taking action a initially, and then following policy π. We remark that policy π is normally derived using q_π.

Solving a RL task means finding π that achieves a large v_π. Policy π, with corresponding v_π, is defined to be better than or equal to policy π', with corresponding v'_π, if $v_\pi(s) \geq v_{\pi'}(s)$ for all $s \in \mathcal{S}$. In this case we say $\pi \geq \pi'$. Policies better than or equal to all other policies are called optimal policies π_*. An optimal policy may not be unique and all optimal policies share the same value function which is called the optimal value function $v_*(s) = \max_\pi v_\pi(s)$.

In practice, the optimal policy and optimal value function are hard to find if the environment dynamics is unknown. The methods for finding the optimal policy without the knowledge of the environment dynamics are called model-free. The method described in this paper is model-free. Also, for continuous or very large state and action space, it is impossible to work in extremely large table settings. Hence, in such situations function approximation techniques are often used.

3. RL Method

The policy gradient method uses gradient descent optimization of a scalar performance measure $J_{\pi_\theta}(s)$ to determine optimal an optimal policy parameter θ. By introducing $J_{\pi_\theta}(s)$ we generalize the value function $v_\pi(s)$ and this is an important factor in improving convergence speed in optimization-based methods. According to the Policy Gradient Theorem (Sutton and Barto, 2018, Sec. 13.2), the derivative of J_{π_θ} w.r.t. θ satisfies

$$\nabla_\theta J_{\pi_\theta} \propto \sum_{a,s} q_\pi(s,a) \nabla_\theta \, \pi_\theta(a|s) \tag{4}$$

Hence, the gradient ascent update to maximize J_{π_θ} is

$$\theta_{k+1} = \theta_k + \alpha \nabla_{\theta_k} J_{\pi_{\theta_k}} = \theta_k + \alpha \sum_{a,s} q_\pi(s,a) \nabla_{\theta_k} \pi_{\theta_k}(a|s) \tag{5}$$

where $\alpha > 0$ is the update step size. Although there is no convergence guarantee for the gradient method, it has been shown to perform in various benchmarks (Raffin et al., 2019).

3.1. Parameterized Neural Policy

This Section gives the expression for the feedforward neural nets used. Notation is adopted from Goodfellow et al. (2016). We consider the so-called *Actor Neural Net* for approximating policy π. An activation function is used to transform linear features to nonlinear features. We choose the $\tanh$ function:

$$g^{(1)}(s) = g^{(2)}(s) = \tanh s \tag{6}$$

where $g^{(k)}$ is evaluated component-wise on the quadrotor state s. The actor neural net is given by

$$h^{(1)}(s) = g^{(1)}(\theta^{(1)\top} s + \theta_b^{(1)}) \tag{7}$$

$$h^{(2)}(h^{(1)}(s)) = g^{(2)}(\theta^{(2)\top} h^{(1)}(s) + \theta_b^{(2)}) \tag{8}$$

$$\mu(s) = \theta^{(3)\top} h^{(2)}(h^{(1)}(s)) + \theta_b^{(3)} \tag{9}$$

$$\pi_\theta(a|s) = \frac{1}{\sqrt{2\pi}\sigma} \exp - \frac{(a-\mu(s))^\top (a-\mu(s))}{2\sigma^2} \tag{10}$$

The outputs of the first and second layers are denoted $h^{(1)}, h^{(2)}$, respectively. The dimension of $h^{(1)}, h^{(2)}$ is N which is a parameter to be adjusted. Thus, we have $\theta^{(1)} \in \mathbb{R}^{N\times n}$, $\theta^{(2)} \in \mathbb{R}^{N\times N}$, $\theta^{(3)} \in \mathbb{R}^{m\times N}$. The biases are denoted $\theta_b^{(k)}, k = 1,2,3$ with their dimension determined from (7)–(9). The output of the neural net determines a Gausian pdf in (10) which is used to evaluate π_θ. The standard deviation σ in (10) is taken as a small constant. A widely used σ is $\sigma = \exp(-\omega)$ where ω is usually chosen on $[0.5, 5]$ (Raffin et al., 2019; Achiam, 2018). Some methods treat σ as a NN parameter which can be tuned during training, *e.g.* Haarnoja et al. (2018).

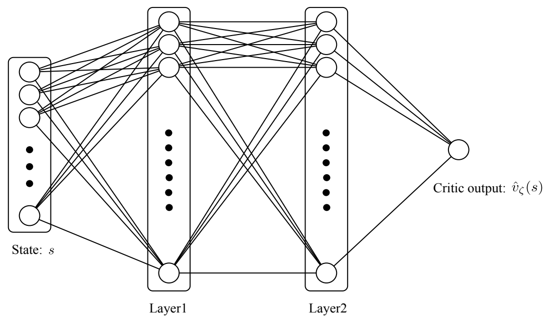
Similarly, the so-called *Critic Neural Net* which approximates $\hat{v}_\zeta$ is given by

$$h^{(1)}(s) = g^{(1)}(\theta^{(1)\top} s + \theta_b^{(1)}) \tag{11}$$

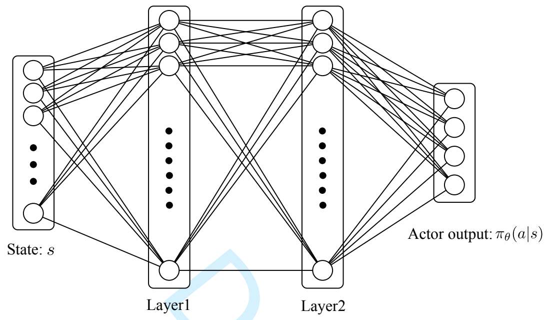
$$h^{(2)}(h^{(1)}(s)) = g^{(2)}(\theta^{(2)\top} h^{(1)}(s) + \theta_b^{(2)}) \tag{12}$$

$$\hat{v}_\zeta(s) = h^{(3)}(h^{(2)}(h^{(1)}(s))) = \zeta^{(3)\top} s + \zeta_b^{(3)} \tag{13}$$

where ζ is the parameters for the neural net and $\zeta^{(1)} \in \mathbb{R}^{N\times n}$, $\zeta^{(2)} \in \mathbb{R}^{N\times N}$, $\zeta^{(3)} \in \mathbb{R}^{1\times N}$.



(a) Critic Neural Net



(b) Actor Neural Net

Fig. 2. Actor and Critic Neural Nets.

3.2. Proximal Policy Optimization (PPO)

PPO is a gradient descent method based on an Actor-Critic model. It is currently considered a state-of-art algorithm in the RL community (Schulman et al., 2017). It benefits from a relatively simple implementation and has shown promising performance in practice (Heess et al., 2017). The innovation of PPO comes from the choice of performance measure

$$J_{\pi_\theta}(s) = \min(k_t(\theta)\hat{A}_t, \sigma(k_t(\theta))\hat{A}_t) \tag{14}$$

where $\hat{A}_t$ is the advantageous estimation (Schulman et al., 2015b) to the end of an episode of length T:

$$\hat{A}_t = \delta_t + \gamma\lambda\delta_{t+1} + \cdots + (\gamma\lambda)^{T-t+1}\delta_{T-1}, \quad \lambda \in (0,1], \tag{15}$$

where

$$\delta_t = r_{t+1}(a,s) + \gamma\hat{v}_\zeta(s_{t+1}) - \hat{v}_\zeta(s_t) \tag{16}$$

with r_t being a deterministic reward function, and $\hat{v}_\zeta(s)$ denotes the estimated value of v_π. Function

$$k_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)} \tag{17}$$

describes the difference between the previous policy with parameter θ_{old} and the present policy with parameter θ as optimization proceeds. The saturation function $\sigma : \mathbb{R} \to \mathbb{R}$ is

$$\sigma(\xi) = \begin{cases} 1+\varepsilon, & \xi > 1+\varepsilon \\ \xi, & 1-\varepsilon \le \xi \le 1+\varepsilon \\ 1-\varepsilon, & \xi < 1-\varepsilon \end{cases} \tag{18}$$

where the clip ratio $\varepsilon > 0$ is a small value.

We remark that strictly speaking quadrotor stabilization is not an episodic task. However, taking a sufficiently large episode length T allows us to approximate a continuing task. In (16), the estimated state value function $\hat{v}_\zeta(s)$ appears. The reason for introducing $\hat{v}_\zeta(s)$ and using $\hat{A}_t$ is to decrease the variance in the estimation of $\bigtriangledown_\theta J_\theta(s)$ and hence improve the convergence speed of training. Using the estimated value function $\hat{v}_\zeta(s)$ to help the convergence speed of policy function π_θ is called an Actor-Critic method. See Konda and Tsitsiklis (2003) for details on the benefits of using an Actor-Critic structure.

Using the Policy Gradient Theorem (*i.e.* (4)), we can take the derivative of J_{π_θ} relative to θ by taking the derivative of $\pi_\theta(a|s)$ relative to θ. Hence, assuming $1-\varepsilon \le k_t(\theta) \le 1+\varepsilon$ we obtain

$$\bigtriangledown_\theta J_{\pi_\theta} = \sum_{t=0}^{T} \hat{A}_t \bigtriangledown_\theta k_t(\theta) \tag{19}$$

which is used to estimate the gradient of J_{π_θ} from sampled action and state data obtained from the agent and environment interaction. The update equation for θ is

$$\theta_{k+1} = \theta_k + \alpha \sum_{t=0}^{T} \hat{A}_t \bigtriangledown_\theta k_t(\theta) \tag{20}$$

It has been shown in Schulman et al. (2017) that PPO is a refinement of TRPO (Schulman et al., 2015a) as it improves training efficiency by constraining the difference between π_θ and $\pi_{\theta_{\text{old}}}$. The saturation function σ limits the magnitude of the gradient $\bigtriangledown_\theta k_t(\theta)$ and ensures the updated θ is close to its previous value θ_{old}.

In order to compute the estimated state value function $\hat{v}_\zeta$ in (16) we optimize the following loss function for the critic neural net:

$$L_\zeta = \sum_{t=0}^{T} (r(a_t, s_t) + \gamma \hat{v}_\zeta(s_{t+1}) - \hat{v}_\zeta(s_t))^2 \tag{21}$$

The loss function minimizes the one-step look ahead error in $\hat{v}_\zeta$. As with (19), sampled data (s, a, r) is used in (21) to estimate the error of $\hat{v}_\zeta$. The update for ζ is

$$\zeta_{k+1} = \zeta_k - \beta \bigtriangledown_\zeta L_\zeta \tag{22}$$

where β is step size.

The pseudocode code of the RL method is in Algorithm 1. Since the update step size α, β in (20) and (22) are usually set very small, to accelerate the training process, we usually update parameter θ, ζ multiple times for the same set of data. After the training process is finished then the quadrotor's motion can be controlled using the policy $\pi_\theta(a|s)$ which maps the measured state to a four-dimensional Gaussian distribution which can be sampled to determine the physical inputs of quadrotor.

Algorithm 1: Proximal Policy Optimization (PPO)

```
1  Input: a differentiable policy parameterization $\pi_\theta(a|s)$;
2  Input: a differentiable estimated state value function parameterization $\hat{v}_\zeta(s)$;
3  for $j = 0, 1, 2, \ldots$ do
4      Run policy $\pi_\theta$ for $K$ timesteps and collect $\{s_t, a_t, r_t\}$. Collect trajectories $D_j = \{\tau_i\}$. ;
5      Calculate $\delta_t, \hat{A}_t$ using (16) ;
6      for $k$ = 1:M do
7          Calculate $\bigtriangledown_\theta J_{\pi_\theta}$ with the collected $\{s, a, r\}$ with (19);
8          Update $\theta$ with using (20) ;
9          $\theta_{k+1} = \theta_k + \alpha \frac{1}{|D_j|T} \sum_{\tau \in D_j} \sum_{t=0}^{T} \hat{A}_t \bigtriangledown_\theta k_t(\theta)$
10     end
11     for $k$ = 1:B do
12         Calculate $\bigtriangledown_\zeta L_\zeta$ with the collected $\{s, a, r\}$ data with (21) ;
13         Update $\zeta$ using (22) ;
14         $\zeta_{k+1} = \zeta_k - \beta \frac{1}{|D_j|T} \sum_{\tau \in D_j} \bigtriangledown_\zeta L_\zeta$
15     end
16 end
```

4. Experiment

4.1. Simulated Quadrotor Dynamics

We consider a traditional quadrotor UAV as shown in Figure 3. Two reference frames are needed for the modelling: a fixed inertial navigation frame $\mathcal{N}$ with orthonormal basis $\{n_1, n_2, n_3\}$ and a body frame $\mathcal{B}$ whose origin is at the vehicle's center of mass (CoM) and with orthonormal basis $\{b_1, b_2, b_3\}$. We define b_1 to point in the forward direction of vehicle, b_2 pointing right, and b_3, n_3 pointing down. The configuration of the quadrotor belongs to the special Euclidean group SE(3), and includes the position $p = [p_1, p_2, p_3]^\top \in \mathbb{R}^3$ of the origin of $\mathcal{B}$ relative to $\mathcal{N}$, and the rotation matrix $R \in$ SO(3) which describes the orientation of $\mathcal{B}$ and $\mathcal{N}$. $\eta = [\phi, \theta, \psi] \in \mathbb{R}$ is Euler-angles corresponding to the rotation matrix. We assume each propeller generates thrust in the $-b_3$ direction and denote the total thrust due to all propellers by the scalar input $u \geq 0$. Controlling individual propeller speeds creates an input torque denoted $\tau = [\tau_1, \tau_2, \tau_3]^\top \in \mathbb{R}^3$ which is expressed in $\mathcal{B}$. To ease the presentation of the control design we take torque τ and thrust u as system inputs. However, in practice the physical input to the UAV is PWM signals to the ESC.

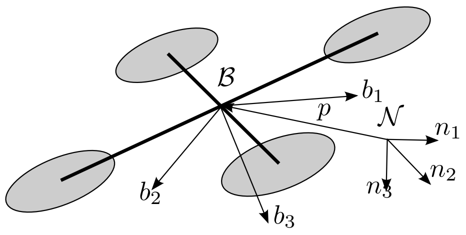


Fig. 3. Quadrotor modelling and reference frames

The UAV dynamics is

(23a) $\dot{p} = v$

(23b) $m\dot{v} = mgn_3 - uRn_3$

(23c) $\dot{R} = RS(\omega)$

(23d) $J\dot{\omega} = -\omega \times J\omega + \tau$

where $v \in \mathbb{R}^3$ is linear velocity expressed in $\mathcal{N}$, $\omega \in \mathbb{R}^3$ is angular velocity in $\mathcal{B}$, m is mass, J is inertia, g is the gravity constant, and $n_3 = [0, 0, 1]^\top$. The skew operator $S(\cdot) : \mathbb{R}^3 \to so(3)$ is given by

$$S(x) = \begin{bmatrix} 0 & -x_3 & x_2 \\ x_3 & 0 & -x_1 \\ -x_2 & x_1 & 0 \end{bmatrix}, \quad \text{where } x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}.$$

The simulator uses a discretized system model

(24a) $v_{t+1} = v_t + \dot{v}_t \mathrm{d}t$

(24b) $\omega_{t+1} = \omega_t + \dot{\omega}_t \mathrm{d}t$

(24c) $p_{t+1} = p_t + v_t \mathrm{d}t + \frac{1}{2}\dot{v}_t \mathrm{d}t^2$

(24d) $\eta_{t+1} = \eta_t + W(\eta_t)\omega_t \mathrm{d}t + W(\eta_t)\dot{\omega}_t \mathrm{d}t^2$

(24e) $W(\eta) = \begin{bmatrix} 1 & \sin\phi\tan\theta & \cos\phi\tan\theta \\ 0 & \cos\phi & -\sin\phi \\ 0 & \frac{\sin\phi}{\cos\theta} & \frac{\cos\phi}{cos\theta} \end{bmatrix}$

which has a sampling interval of $\mathrm{d}t$. This model assumes a constant $\dot{\omega}$ and $\dot{v}$ between sampling instants. The simulator runs at $\mathrm{d}t = 1\,\mathrm{ms}$. The parameters for the quadrotor are $m = 0.5\,\mathrm{kg}$, $g = 9.81\,\mathrm{m/s^2}$, $J = \mathrm{diag}(J_1, J_2, J_3)$, $J_1 = 0.0135\,\mathrm{kg\,m^2}$, $J_2 = 0.0135\,\mathrm{kg\,m^2}$, and $J_3 = 0.024\,\mathrm{kg\,m^2}$.

4.2. Reward Design

An important factor in the RL-based control's motion control performance is the choice of dependence of the reward function $r(a, s)$ on control input $a = [u, \tau^\top]^\top$ and $s = [p^\top, \eta^\top, v^\top, \omega^\top]^\top$. In this paper we considered three different reward designs to investigate the influence of reward function on

Hyperparameter Set	$\{\lambda, \varepsilon, \alpha, \beta\}$
1	$\{0.95, 0.3, 3\times10^{-4}, 1\times10^{-3}\}$
2	$\{0.95, 0.3, 3\times10^{-5}, 1\times10^{-4}\}$
3	$\{0.95, 0.2, 3\times10^{-4}, 1\times10^{-3}\}$
4	$\{0.95, 0.2, 3\times10^{-5}, 1\times10^{-4}\}$
5	$\{0.95, 0.1, 3\times10^{-4}, 1\times10^{-3}\}$

Table 1. Hyperparameters sets corresponding to successful training. Training progress is shown in Figure 5

closed-loop performance:

(25) $$r_a(a,s) = r_0 - \|v\|_2 - \|\omega\|_2$$

(26) $$r_b(a,s) = r_0 - \|v\|_2 - \|\omega\|_2 - \|p\|_2$$

(27) $$r_c(a,s) = r_0 - \|v\|_2 - \|\omega\|_2 - \|p\|_2 - \|\tau\|_2$$

(28) $$r_0 = \begin{cases} 1, & -0.5 \le p_3 \le 0.5 \\ -1, & \text{otherwise} \end{cases}$$

r_0 used here is to accelerate the training process. Reward r_a aims to control hover using only velocity. Reward r_b includes an extra term $\|p\|_2$ to reduce drift in position due to unmodelled disturbances. Reward r_c includes $\|\tau\|_2$ to limit control effort. The yaw rate of the vehicle is regulated to zero by including the angular velocity in the reward.

4.3. Simulation Results

4.3.1. Training

In order to optimize training, the hyperparameters λ, ε, and learning rates α, β were adjusted for reward function r_c. Parameter λ appears in the generalized advantage estimator (15). As $\lambda \to 1$ the bias of the estimator decreases. Although the bias will increase as $\lambda \to 0$, smaller λ might accelerate the training process. Parameter ε is the clip ratio in (18). It constrains the difference between the old and new policy. A grid of parameter values was created from $\lambda \in \{0.35, 0.65, 0.95\}$, $\varepsilon \in \{0.1, 0.2, 0.3\}$, and $(\alpha, \beta) \in \{(3\times10^{-3}, 1\times10^{-2}), (3\times10^{-4}, 1\times10^{-3}), (3\times10^{-5}, 1\times10^{-4})\}$. This yielded 27 different parameter combinations. The training was defined to fail if there is convergence to a local maximum which is unable to stabilize the quadrotor. The hyperparameter search led to successful training when $\lambda = 0.95$ for certain values of $\varepsilon, \alpha, \beta$ as indicated in Table 1. The average cumulative reward for a single trajectory collected in the training process using 6 different random seeds is denoted V_{avg}, and its value is shown in Figure 5 for the 5 cases of successful training. Parameter Set 3 has the fastest convergence speed and achieves a relatively high V_{avg} at around 500 iterations. It is therefore chosen for the final training process. The remaining design parameter values used are in Table 2. The training process is about 12 hours and is run on a single core of a E5-2683 v4 Intel Broadwell CPU.

Training progress for the three reward functions is shown in Figure 4. The average cumulative reward V_{avg} for a single trajectory using 6 different random seeds. We observe V_{avg} for r_a and r_b converge to a higher value than r_c. This is expected as r_c includes position error and control input as negative reward. The learning rate for r_c is the fastest and steady state is achieved at around 500 iterations. Although the algorithms can achieve convergence for the three reward cases, the time domain performances varies between different reward functions.

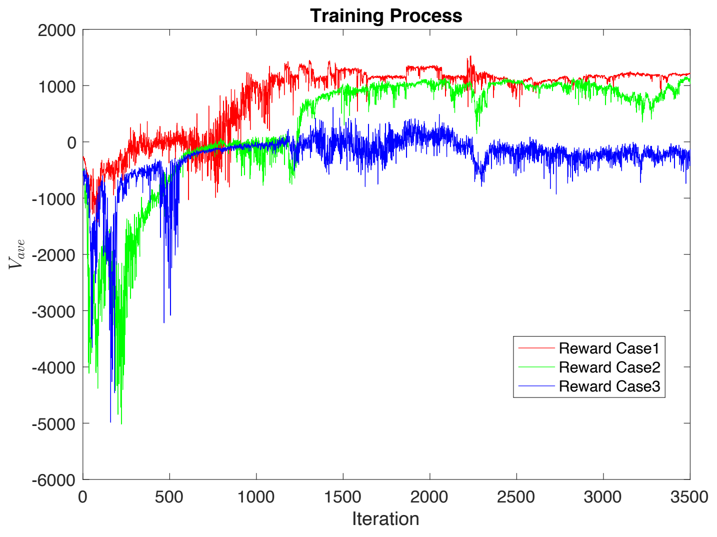


Fig. 4. Training progress for V_{avg} for different rewards

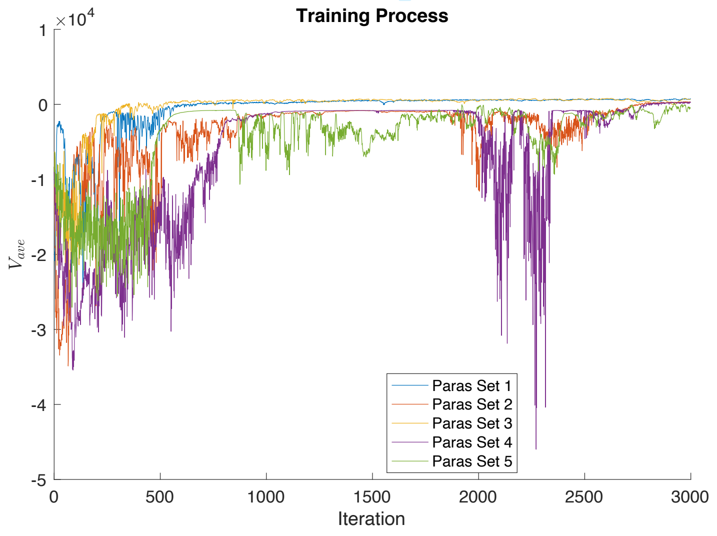


Fig. 5. Training progress for V_{avg} for different hyperparameters

Parameters	Values
α	3×10^{-4}
β	1×10^{-3}
γ	0.99
λ	0.95
ε	0.2
K	4000
M	80
B	80
T	1000
N	64

Table 2. Algorithm Parameters

4.3.2. Performance Testing

Figures 6–9 present the simulation results after training using the three reward functions. As well, a traditional manually tuned inner-outer loop PD control is provided for comparison. This control is defined by

$$\begin{aligned}
\phi_r &= k^o_{1p}(p^r_1 - p_1) + k^o_{1d}\frac{d}{dt}(p^r_1 - p_1)\\
\theta_r &= k^o_{2p}(p^r_2 - p_2) + k^o_{2d}\frac{d}{dt}(p^r_2 - p_2)\\
\tau_1 &= k^i_{1p}(\phi_r - \phi) + k^i_{1d}\frac{d}{dt}(\phi_r - \phi)\\
\tau_2 &= k^i_{2p}(\theta_r - \theta) + k^i_{2d}\frac{d}{dt}(\theta_r - \theta)\\
\tau_3 &= -k_{3p}\psi\\
u &= k_{hp}(p^r_3 - p_3) + k_{hd}\frac{d}{dt}(p^r_3 - p_3)
\end{aligned} \tag{29}$$

where ϕ_r, θ_r are roll and pitch set points from the outer loop. The control gains are chosen as $k^o_{1p} = k^o_{2p} = k^i_{1p} = k^i_{2p} = 0.07, k^o_{1d} = k^o_{2d} = k^i_{1d} = k^i_{2d} = 0.09, k_{3p} = 0.1, k_{hp} = 2, k_{hd} = 8$.

The control objective is to stabilize the position $[0, 0, 0]^\top$ m with the vehicle initialized to $p(0) = [1, 1, 0]^\top$ m. The results are given in Figures 6–9. We observe for r_a that velocity is regulated to zero and roll and pitch have a small oscillation in steady-state. These oscillations also appear in the inputs. As expected, there is a large steady-state position error as r_a does not measure this error. Using r_b the amplitude of steady-state position error is reduced relative to r_a since it includes position in the reward. The amplitude of steady-state oscillation in input and roll/pitch is increased. This is likely due to position being added to r_b. Reward r_c leads to the best performance with the smallest steady-state input and roll/pitch oscillation. The increased steady-state error position compared to r_b is due to control input added.

A time-varying trajectory tracking is tested using r_c and compared with the PD controller. A figure-8 reference trajectory $p_r = [1.5\sin(2t) + 1,\ 0.75\sin(4t),\ -4 + 2\sin(2t)]^\top$ is used. The tracking error is shown in Figure 10. The RL controller performs better than the PD controller designed for hover. Hence, the proposed method exhibits performance in tasks more general than what it was trained for.

5. Conclusion

This paper applied the deep RL method PPO to control the full 6 DoF system dynamics of a quadrotor UAV. Relative to the existing method, the proposed method considers the *full dynamics* of the UAV and this makes the design challenging. The work explored the effect of reward functions on closed-loop performance of the trained neural net controller. We observed that although different rewards functions can achieve a stable hover. Including input effort into the reward makes the closed-loop less sensitive (*e.g*. reduced impractical oscillation in the input signals). A simulation comparison of the proposed RL-based control and a classical inner-outer loop PD controller was presented. The results demonstrate similar hover stabilization and output tracking performance without requiring manual tuning.

Future work includes testing the control in actual flight, improving the training speed of the algorithm, and developing a systematic procedure for designing the reward function. We investigate methods which provide monotone performance improvement during training. Finally, we plan to develop algorithms which are robust to parameter updates. This will enable on-line training and improve robustness to environment change.

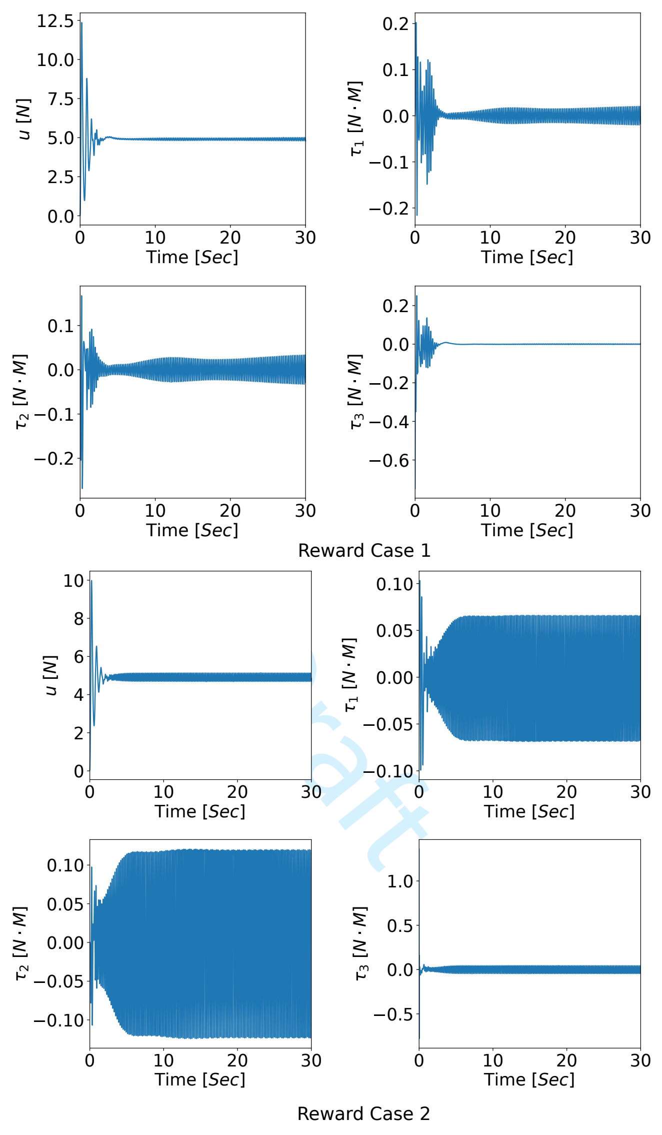
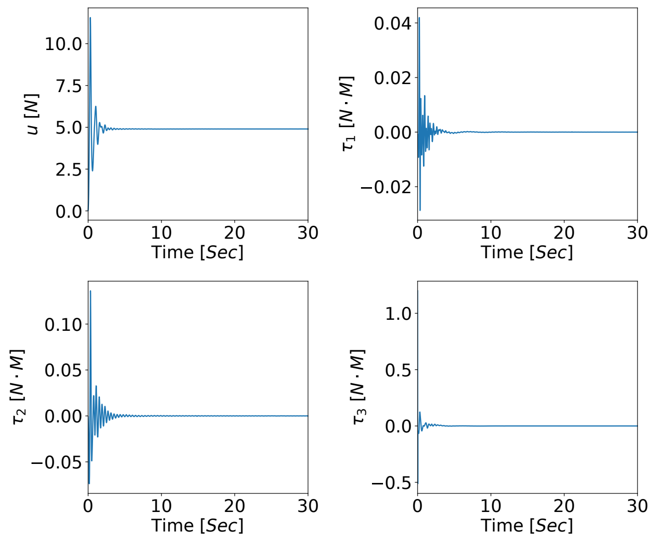
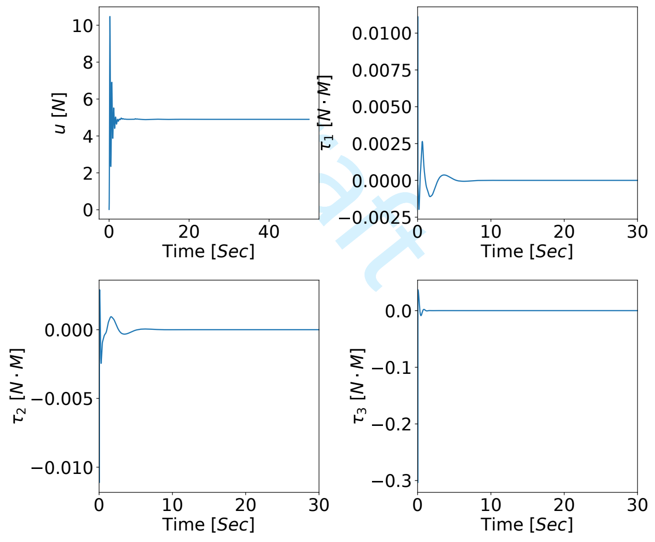


Fig. 6. Simulation Results for hover stabilization. Input trajectories



Reward Case 3



PD Control

Fig. 7. Simulation Results for hover stabilization. Input trajectories

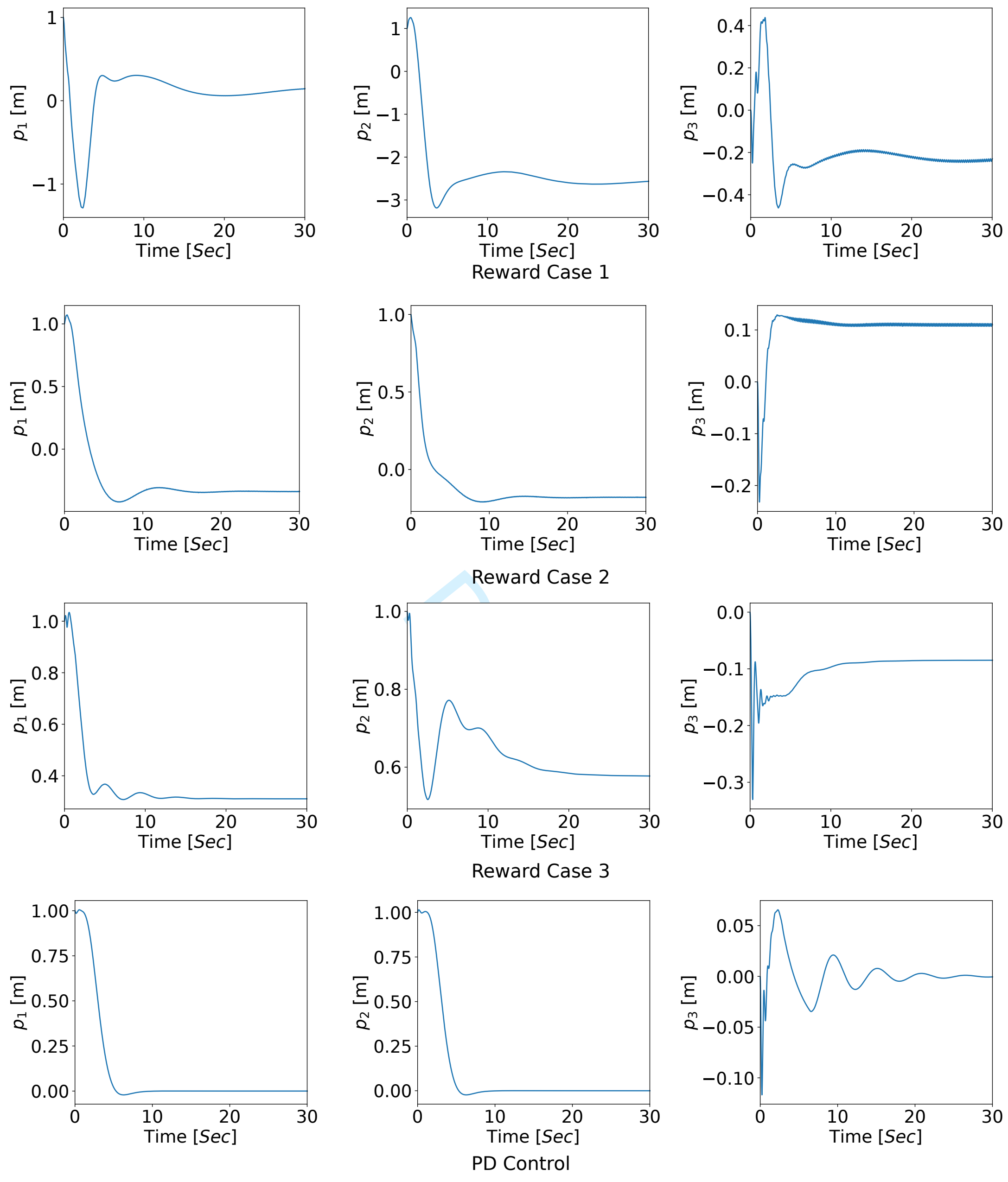


Fig. 8. Simulation Results for hover stabilization. Position trajectories

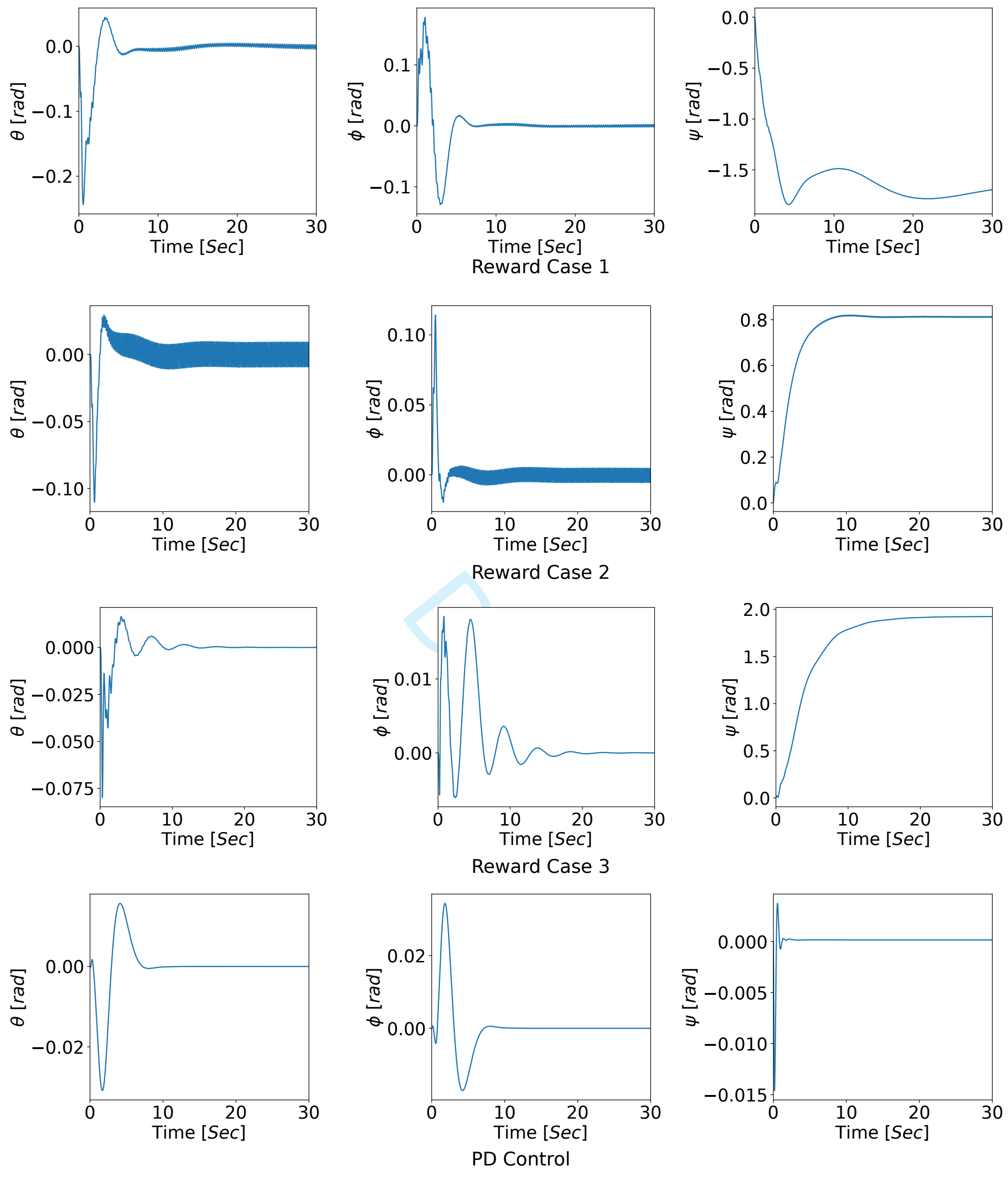


Fig. 9. Simulation Results for hover stabilization. Attitude trajectories

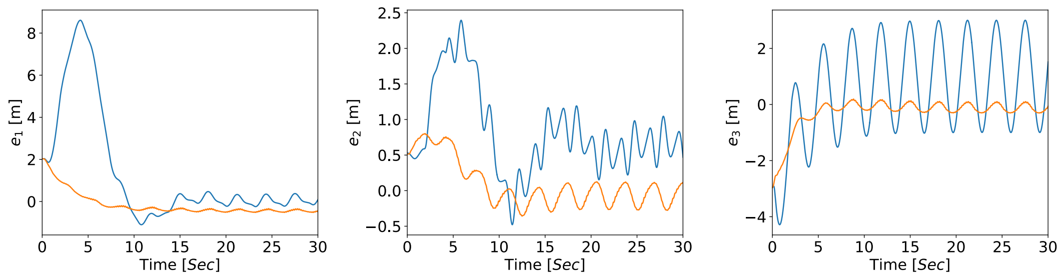


Fig. 10. Simulation Results for Trajectory Tracking Error(Orange line is for NN control based on r_c, Blue line is for PD Controller).

References

Abbeel, P., Coates, A., and Ng, A. Y. Autonomous helicopter aerobatics through apprenticeship learning. *The International Journal of Robotics Research*, 29(13):1608–1639, jun 2010. doi: 10.1177/0278364910371999.

Achiam, J. Spinning Up in Deep Reinforcement Learning. 2018.

Bangura, M. *Aerodynamics and Control of Quadrotors*. PhD thesis, College of Engineering and Computer Science, The Australian National University, 2017.

Bøhn, E., Coates, E. M., Moe, S., and 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)*, pages 523–533. IEEE, 2019.

Cao, N. and Lynch, A. Time-delay robustness analysis of a nested saturation control for uav motion control. *Control Strategy for Time-Delay Systems: Part II: Engineering Applications*, page 69, 2020.

Cao, N. and Lynch, A. Predictor-based control design for uavs: robust stability analysis and experimental results. *International Journal of Control*, 94(6):1529–1543, 2021.

Cao, N. and Lynch, A. F. Inner-outer loop control with constraints for rotary-wing UAVs. In *2015 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 294–302. IEEE, 2015.

Dong, W., Gu, G.-Y., Zhu, X., and Ding, H. High-performance trajectory tracking control of a quadrotor with disturbance observer. *Sensors and Actuators A: Physical*, 211:67–77, 2014.

Goodfellow, I., Bengio, Y., Courville, A., and Bengio, Y. *Deep learning*, volume 1. MIT press Cambridge, 2016.

Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pages 1861–1870. PMLR, 2018.

Heess, N., TB, D., Sriram, S., Lemmon, J., Merel, J., Wayne, G., Tassa, Y., Erez, T., Wang, Z., Eslami, S., et al. Emergence of locomotion behaviours in rich environments. *arXiv preprint arXiv:1707.02286*, 2017.

Helder, B., Van Kampen, E.-J., and Pavel, M. Online adaptive helicopter control using incremental dual heuristic programming. In *AIAA Scitech 2021 Forum*, page 1118, 2021.

Hua, M.-D., Hamel, T., Morin, P., and Samson, C. Introduction to feedback control of underactuated vtolvehicles: A review of basic control design ideas and principles. *IEEE Control systems magazine*, 33(1):61–75, 2013.

Hwangbo, J., Sa, I., Siegwart, R., and Hutter, M. Control of a quadrotor with reinforcement learning. *IEEE Robotics and Automation Letters*, 2(4):2096–2103, oct 2017. doi: 10.1109/lra.2017.2720851.

Kaufmann, E., Loquercio, A., Ranftl, R., Müller, M., Koltun, V., and Scaramuzza, D. Deep drone acrobatics. *arXiv preprint arXiv:2006.05768*, 2020.

Kober, J., Bagnell, J. A., and Peters, J. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32(11):1238–1274, aug 2013. doi: 10.1177/0278364913495721.

Konda, V. R. and Tsitsiklis, J. N. On actor-critic algorithms. *SIAM journal on Control and Optimization*, 42(4):1143–1166, 2003.

Lambert, N. O., Drew, D. S., Yaconelli, J., Levine, S., Calandra, R., and Pister, K. S. Low-level control of a quadrotor with deep model-based reinforcement learning. *IEEE Robotics and Automation Letters*, 4(4):4224–4230, 2019.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. Continuous control with deep reinforcement learning. *arXiv:1509.02971 [cs, stat]*, July 2015. URL `http://arxiv.org/abs/1509.02971`. arXiv: 1509.02971.

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

OpenAI, Akkaya, I., Andrychowicz, M., Chociej, M., Litwin, M., McGrew, B., Petron, A., Paino, A., Plappert, M., Powell, G., Ribas, R., Schneider, J., Tezak, N., Tworek, J., Welinder, P., Weng, L., Yuan, Q., Zaremba, W., and Zhang, L. Solving rubik's cube with a robot hand. *arXiv preprint arXiv:1910.07113*, 2019.

Polvara, R., Patacchiola, M., Sharma, S., Wan, J., Manning, A., Sutton, R., and Cangelosi, A. Toward end-to-end control for uav autonomous landing via deep reinforcement learning. In *2018 International conference on unmanned aircraft systems (ICUAS)*, pages 115–123. IEEE, 2018.

Raffin, A., Hill, A., Ernestus, M., Gleave, A., Kanervisto, A., and Dormann, N. Stable baselines3. `https://github.com/DLR-RM/stable-baselines3`, 2019.

Ruggiero, F., Lippiello, V., and Ollero, A. Aerial manipulation: A literature review. *IEEE Robotics and Automation Letters*, 3(3):1957–1964, 2018.

Schulman, J., Levine, S., Abbeel, P., Jordan, M., and Moritz, P. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897. PMLR, 2015a.

Schulman, J., Moritz, P., Levine, S., Jordan, M., and Abbeel, P. High-dimensional continuous control using generalized advantage estimation. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2015b.

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

Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., and Riedmiller, M. Deterministic Policy Gradient Algorithms. In *International Conference on Machine Learning*, pages 387–395, Jan. 2014. URL `http://proceedings.mlr.press/v32/silver14.html`.

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., Chen, Y., Lillicrap, T., Hui, F., Sifre, L., van den Driessche, G., Graepel, T., and Hassabis, D. Mastering the game of go without human knowledge. *Nature*, 550(7676):354–359, oct 2017. doi: 10.1038/nature24270.

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

Tang, S. and Kumar, V. Autonomous flight. *Annual Review of Control, Robotics, and Autonomous Systems*, 1(1):29–52, 2018.

Valavanis, K. P. and Vachtsevanos, G. J. *Handbook of unmanned aerial vehicles*, volume 2077. Springer, 2015.

Villa, D. K., Brandao, A. S., and Sarcinelli-Filho, M. A survey on load transportation using multirotor uavs. *Journal of Intelligent & Robotic Systems*, pages 1–30, 2019.

Zhang, Y., Zhang, Y., and Yu, Z. Path following control for uav using deep reinforcement learning approach. *Guidance, Navigation and Control*, 1(01):2150005, 2021.

Zhou, Y., van Kampen, E.-J., and Chu, Q. P. Incremental model based online dual heuristic programming for nonlinear adaptive control. *Control Engineering Practice*, 73:13–25, 2018.