Quadrotor motion control using deep reinforcement learning[1]

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 proportional-derivative (PD) controller, despite not depending on any model information. The paper considers different choices of reward function and their influence on controller performance.

Key words: motion control, unmanned aerial vehicles, learning system, reinforcement learning.

Résumé : Les auteurs présentent un contrôleur basé sur un réseau neuronal profond entraîné par un algorithme d'apprentissage par renforcement (AR) sans modèle afin d'obtenir la stabilisation en vol stationnaire d'un véhicule aérien sans pilote (UAV) quadricoptère. Deux réseaux neuronaux sont entraînés à l'aide de l'AR. Un réseau neuronal est utilisé comme un contrôleur stochastique qui donne la distribution des entrées de contrôle. L'autre met en correspondance l'état de l'UAV à un scalaire qui estime la récompense du contrôleur. Une méthode d'optimisation des politiques proximales (OPP), qui est une approche évaluateur–acteur de gradient des politiques, est utilisée pour entraîner les réseaux neuronaux. Les résultats de la simulation montrent que le contrôleur entraîné atteint un niveau de performance comparable à celui d'un régulateur proportionnel-différentiel (PD) réglé manuellement, même s'il ne dépend d'aucune information provenant d'un modèle. Le document examine différents choix de fonction de récompense et leur effet sur la performance du contrôleur. [Traduit par la Rédaction]

Mots-clés : commande de mouvement, véhicule aérien sans pilote, apprentissage par renforcement.

1. Introduction

Motion control for UAVs is a primary area of research that continues to generate interest. A survey of the field is given in Valavanis and Vachtsevanos (2015). The interest in motion control comes from a practical need for high-performance trajectory tracking that is robust to external force disturbances and changing dynamics. For example, external disturbance forces arise in load transportation (Villa et al. 2020) or during non-destructive

Received 13 May 2021. Accepted 17 September 2021.

Z. Jiang and A.F. Lynch. Department of Electrical and Computer Engineering, University of Alberta, Edmonton, AB T6G 1H9, Canada.
Corresponding author: Alan F. Lynch (e-mail: alan.lynch@ualberta.ca).

[1]This paper was the winner of the 2020 Unmanned Systems Canada Student Paper Competition.

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 or delay in measurements and state estimates (Dong et al. 2014; Cao and Lynch 2020, 2021), and 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 dynamics derived from physical laws. Traditionally, a control law has a fixed structure influenced by the model equations and has adjustable gains that affect 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 only under ideal modelling assumptions. Such shortcomings of traditional methods have led to interest in RL applied to UAV motion control. RL has the advantage of requiring less domain knowledge and the design method can be applied to a generic system. For example, the method considered here uses only simulation data to train the control law or policy that maximizes an accumulated reward to minimize position and yaw regulation error. No particular controller structure is specified in advance other than it is a static state feedback. As well, controller tuning is performed automatically during training.

RL is concerned with how agents act within an environment to maximize a scalar cumulative reward that 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 that are discrete. Although RL has been used in robotics for decades, it has mainly been confined to 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.

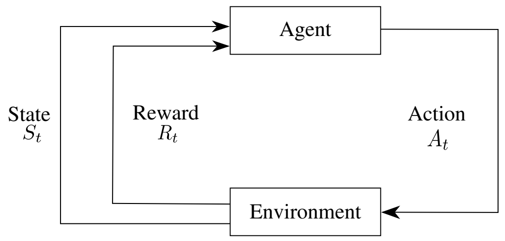
There has been recent attention on RL applied to UAVs. This work can be divided into model-based and model-free methods. Model-based refers to learning an optimal controller indirectly by learning a model that can predict the next system state given the current state and input. We begin by surveying some of the relevant model-based approaches. Abbeel et al. (2010) consider a design in three stages where initially apprenticeship learning extracts a reference trajectory for the UAV from suboptimal expert pilot demonstrations. Second, a full nonlinear dynamic model for a helicopter UAV is identified. Third, a receding horizon variation of linear quadratic (LQ) control based on a linearized model is applied to the identified dynamics. Experimental results are provided. Lambert et al. (2019) focus on inner loop control, which maps inertial measurement unit (IMU) data to pulse-width modulation (PWM) waveforms for the electronic speed controllers (ESCs). During training, a NN identifies a model of the rotational dynamics that is combined with a random shooter model predictive control (MPC) that determines an optimal control by simulation. Experimental results are provided on the Crazyflie platform and short-term hover performance is demonstrated in experiment. No control of the translational variables considered. 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). PID control is used for lateral position and roll-yaw. The reduced number of DoF controlled by RL leads to faster learning due to the simple dynamics with reduced coupling. A linearization of the dynamics about its trajectory is estimated during training. A detailed simulation model is used to train the controller and test its performance. Kaufmann et al. (2021) present 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 to improve transfer from simulation to the real world. These processed measurements are fed to a NN that 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. The above work on model-based RL shows these methods have evolved to eliminate the need for expert demonstration. Further, the adoption of NNs has led to more complex applications such as end-to-end vision-based control (Kaufmann et al. 2021).

Model-free RL has been investigated recently to develop NN-based controllers. These approaches are "model-free" as they do not 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. 2015*b*), 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 computationally efficient than model-based methods. However, model-free methods are generally easier to implement and currently receive more attention in the RL community. Polvara et al. (2018) applied 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 waveforms). Bøhn et al. (2019) used PPO for attitude control of a fixed-wing UAV. Here, the attitude dynamics are inherently stable, which makes the problem less challenging than quadrotor control. Zhang et al. (2021) improved the DDPG algorithm by using a double experience replay buffer (DERB) for training. This DERB-DDPG control is trained using linearized outer-loop system dynamics. Simulation results are presented using an inner–outer loop control. Hwangbo et al. (2017) proposed an improvement on the DDPG algorithm by accurately estimating the gradient of the objective function with respect to action. The authors use a singular value decomposition 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 attitude control is used to assist the RL method. Experiments show hover stabilization for a wide range of initial conditions.

The proposed PPO-based method is unique relative to existing approaches in that it 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) and Lambert et al. (2019). As well, no traditional control is used to assist the RL during training or testing. This should be compared to Hwangbo et al. (2017), where PD control assists RL. Our approach investigates the choice of reward function on time-domain tracking performance. Previous RL work does not focus on this design parameter and we show it is useful in improving closed-loop performance. Simulation results show hover and trajectory tracking performance that 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 the PPO method. Section 4 presents the training and simulation results. Section 5

Fig. 1. The agent–environment interaction.



presents the conclusion. The code and simulation data for this project are available in a GitHub repository (ZifeiFFF and Jiang 2021).

2. Preliminaries

2.1. Markov decision process (MDP)

Figure 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 that influences the environment state. The environment also provides a reward signal that 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 $S_t \in \mathcal{S}$ to compute an action $A_t \in \mathcal{A}$. Here, S_t and 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 $n = 12$ and $m = 4$ for a quadrotor. An MDP is normally used to model the dynamics of the environment. The MDP is described by the function $p: \mathcal{S} \times \mathcal{R} \times \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$ where

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

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 that 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 ordinary differential equation 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 $\pi(a|s)$ that is the probability that $A_t = a$ if $S_t = s$. In control systems terminology the policy π corresponds to the control law. In deep RL, the policy is parameterized using a NN with vector parameter θ. The parameterized policy is $\pi_\theta(a|s)$ and the parameterization is described in Subsection 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:

(2) $$v_\pi(s) = \mathbb{E}_\pi\left[\sum_{t=0}^{\infty}\gamma^t R_t \middle| S_0 = s\right]$$

where $\gamma < 1$ is the discounting factor. Similar to v_π we introduce the action–value function $q_\pi(s, a)$ for policy π as

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

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 an 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 $v_{\pi'}$, if $v_\pi(s) \geq v_{\pi'}(s)$ for all $s \in \mathcal{S}$. In this case we say $\pi \geq \pi'$. Policies π^* better than or equal to all other policies are called optimal. Optimal policies may not be unique and share the same value function 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. Methods for finding the optimal policy without knowledge of the environment dynamics are called model-free. The method described in this paper is model-free. Another challenge for continuous or very large state and action space is that it is impossible to work with extremely large table settings. Hence, in such situations function approximation techniques are often used. This approach is used in this paper.

3. RL method

The policy gradient method uses gradient descent optimization of a scalar performance measure $J_{\pi_\theta}(s)$ to determine an optimal policy parameter θ. A NN is used to implement the parameterization as described in Section 3.1. 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, Section 13.2), the gradient of J_{π_θ} with respect to θ satisfies

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

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

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

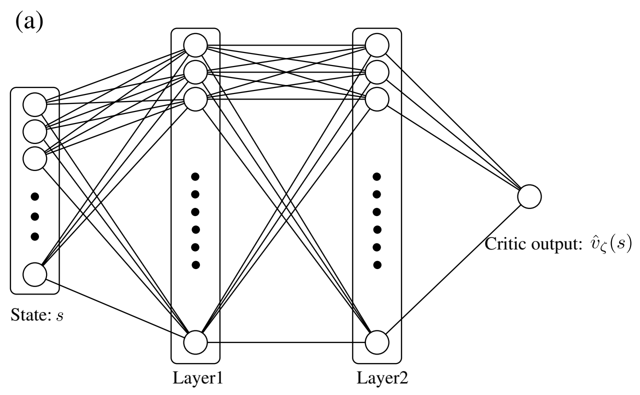
where $\alpha > 0$ is the update step size. Although there is no convergence guarantee for the policy gradient method with NN function approximation, it has been shown to perform in various benchmarks (Raffin et al. 2019).

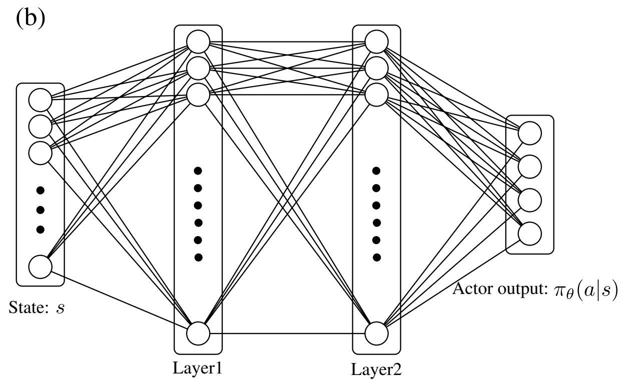
3.1. Parameterized neural policy

This subsection gives the expression for the feedforward NNs used. Notation is adopted from Goodfellow et al. (2016). An actor NN is used to approximate policy π. An activation function is used to transform linear features

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

Fig. 2. (*a*) Critic and (*b*) actor neural nets.





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

$$h^{(1)}(s) = g^{(1)}(\theta^{(1)\mathrm{T}}s + \theta_{\mathrm{b}}^{(1)}) \tag{7}$$

$$h^{(2)}(h^{(1)}(s)) = g^{(2)}(\theta^{(2)\mathrm{T}}h^{(1)}(s) + \theta_{\mathrm{b}}^{(2)}) \tag{8}$$

$$\mu(s) = \theta^{(3)\mathrm{T}}h^{(2)}(h^{(1)}(s)) + \theta_{\mathrm{b}}^{(3)} \tag{9}$$

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

The outputs of the first and second layers are denoted $h^{(1)}$ and $h^{(2)}$, respectively. The dimension of $h^{(1)}$ and $h^{(2)}$ is N, which is a design parameter. 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 $\theta_{\mathrm{b}}^{(k)}$, $k = 1, 2, 3$ with their dimension determined from (7)–(9). Parameter θ consists of $\theta^{(k)}$ and $\theta_{\mathrm{b}}^{(k)}$, $k = 1, 2, 3$. The output of the NN is a Gaussian probability density function (10) that evaluates π_θ. The standard deviation σ in (10) is taken as a small constant. A widely used expression for σ is $\sigma = \exp(-\omega)$ where ω is usually chosen on [0.5, 5] (Achiam 2018; Raffin et al. 2019). Some methods treat σ as a NN parameter that can be tuned during training (e.g., soft actor-critic method (Haarnoja et al. 2018)).

Similarly, the critic NN approximates v_{π_θ} by $\hat{v}_\zeta$ which is evaluated using

(11) $h^{(1)}(s) = g^{(1)}(\zeta^{(1)^{\mathrm{T}}}s + \zeta_{\mathrm{b}}^{(1)})$

(12) $h^{(2)}(h^{(1)}(s)) = g^{(2)}(\zeta^{(2)^{\mathrm{T}}}h^{(1)}(s) + \zeta_{\mathrm{b}}^{(2)})$

(13) $\hat{v}_\zeta(s) = h^{(3)}(h^{(2)}(h^{(1)}(s))) = \zeta^{(3)^{\mathrm{T}}}h^{(2)}(h^{(1)}(s)) + \zeta_{\mathrm{b}}^{(3)}$

where $\zeta^{(1)} \in \mathbb{R}^{N\times n}$, $\zeta^{(2)} \in \mathbb{R}^{N\times N}$, $\zeta^{(3)} \in \mathbb{R}^{1\times N}$. Parameter ζ consists of $\zeta^{(k)}$ and $\zeta_{\mathrm{b}}^{(k)}$, $k = 1, 2, 3$. The structure of actor and critic NN is shown in Fig. 2.

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-the-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

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

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

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

where

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

with r_t being a deterministic reward. Function

(17) $k_t(\theta) = \dfrac{\pi_\theta(a_t|s_t)}{\pi_{\theta_1}(a_t|s_t)}$

describes the ratio of policies corresponding to parameters θ and θ_1 where θ_1 is the initial parameter of the optimization. The saturation function $\sigma:\mathbb{R} \to \mathbb{R}$ is

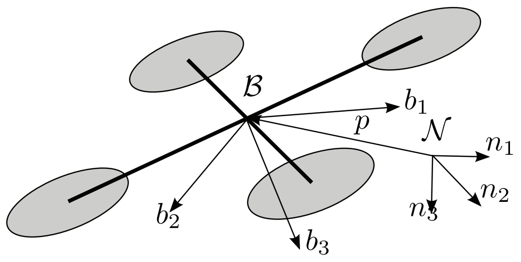
(18) $\sigma(\xi) = \begin{cases} 1+\varepsilon & \xi > 1+\varepsilon \\ \xi & 1-\varepsilon \le \xi \le 1+\varepsilon \\ 1-\varepsilon & \xi < 1-\varepsilon \end{cases}$

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

Strictly speaking, quadrotor stabilization is not an episodic task; however, taking a sufficiently large episode length T allows us to approximate a continuing task. A trajectory is a sequence of states and actions, denoted as $\tau_i = \{s_0, a_0, s_1, a_1, ..., s_{\mathrm{T}}\}$, $i \ge 1$. A set of multiple trajectories is denoted as $D = \{\tau_1, \tau_2, ...\}$. In (16), the estimated state value function $\hat{v}_\zeta(s)$ appears. The reason for introducing the estimated state-value function $\hat{v}_\pi(s)$ and using $\hat{A}_t$ in (16) is to decrease the variance in the estimation of $\nabla_\theta J_\theta(s)$ 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 gradient 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

Fig. 3. Quadrotor modelling and reference frames.



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

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

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

It has been shown in Schulman et al. (2017) that PPO is a refinement of TRPO (Schulman et al. 2015*a*) as it improves training efficiency by constraining the difference between $\pi_{\theta_{k+1}}$ and π_{θ_k}. The saturation function σ limits the magnitude of the gradient $\nabla_\theta k_t(\theta)$ and ensures the updated θ_{k+1} is close to its previous value θ_1.

To compute the estimated state value function $\hat{v}_\zeta$ in (16) we optimize the critic NN loss function

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

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

(22) $$\zeta_{k+1} = \zeta_k - \beta \nabla_\zeta L_{\zeta_k}$$

where β is step size.

The pseudocode of the RL method is given in Algorithm 1. PPO is an off-policy method, which means that the update steps of the actor and critic NNs are separated from the data collection step. 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 the quadrotor.

Algorithm 1: PPO

```
1 Input: policy parameterization $\pi_\theta(a|s)$;
2 Input: estimated state-value function parameterization $\hat{v}_\zeta(s)$;
3 Randomly choose $\theta_1$, $\zeta_1$.
4 for $j = 0, 1, 2, \ldots$ do
5       Run policy $\pi_{\theta_\lambda}$ to generate a set of trajectories of length $T$: $D_j = \{\tau_i, 1 \le i \le K/T\}$;
6       Calculate $\delta_t$, $\hat{A}_t$ using (15) and (16);
```

```
7    for $k$ = 1:$M$ do
8        Calculate $\nabla_\theta J_{\pi_\theta}$ with collected data using (19);
9        Update $\theta$ using (20);
10       $$\theta_{k+1} = \theta_k + \alpha \frac{1}{|D_j|T} \sum_{\tau \in D_j} \sum_{t=0}^{T} \hat{A}_t \nabla_\theta k_t(\theta_k)$$
11   $|D_j|$ is the cardinality of $D_j$;
12   end
13   $\theta_1 = \theta_{M+1}$;
14   for $k$ = 1:$B$ do
15       Calculate $\nabla_\zeta L_\zeta$ with collected data $\{s_t, a_t, r_t\}$ using (21):
16       Update $\zeta$ using (22):
17       $$\zeta_{k+1} = \zeta_k - \beta \frac{1}{|D_j|T} \sum_{\tau \in D_j} \nabla_\zeta L_{\zeta_k}$$
18   end
19   $\zeta_1 = \zeta_{B+1}$
20 end
```

4. Experiment

4.1. Simulated quadrotor dynamics

Following the dynamic model in Bouabdallah (2007), consider a traditional quadrotor UAV as shown in Fig. 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 and with orthonormal basis $\{b_1, b_2, b_3\}$. We define b_1 to point in the forward direction of the 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 $p = [p_1, p_2, p_3]^T \in \mathbb{R}^3$ 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 $\mathcal{N}$. The ZYX Euler angles are roll ϕ, pitch θ, yaw ψ, and we define $\eta = [\phi, \theta, \psi]^T \in \mathbb{R}^3$. 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 $\tau = [\tau_1, \tau_2, \tau_3]^T \in \mathbb{R}^3$ 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 inputs to the UAV are ESC PWM signals which are algebraically related to u and τ.

The UAV dynamics is

$$\dot{p} = v \tag{23a}$$

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

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

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

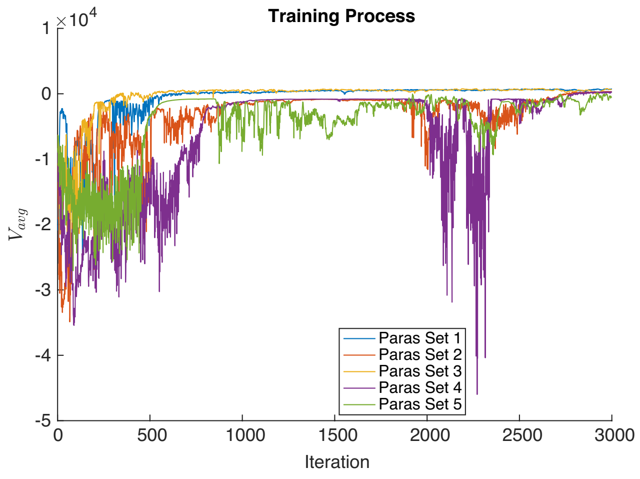
where $v \in \mathbb{R}^3$ is linear velocity expressed in $\mathcal{N}$, $\omega \in \mathbb{R}^3$ is angular velocity in $\mathcal{B}$, m is mass, J is inertia, g is the gravity constant, and $n_3 = [0, 0, 1]^T$. The skew operator $S(\cdot) : \mathbb{R}^3 \rightarrow so(3)$ is given by

Table 1. Hyperparameter sets corresponding to successful training.

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	$\{0.95, 0.1, 3\times10^{-4}1, \times10^{-3}\}$

Note: Training progress is shown in Fig. 5.

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



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

The simulator uses a discretized system model

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

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

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

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

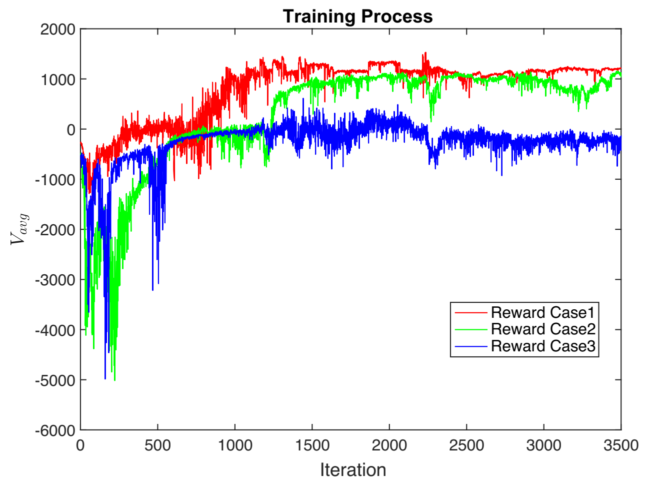
(24*e*) $W(\eta) = \begin{bmatrix} 1 & \sin\phi\tan\theta & \cos\phi\tan\theta \\ 0 & \cos\phi & -\sin\phi \\ 0 & \dfrac{\sin\phi}{\cos\theta} & \dfrac{\cos\phi}{\cos\theta} \end{bmatrix}$

which has a sampling interval of dt. This model assumes a constant $\dot{\omega}$ and $\dot{v}$ between sampling instants. The simulator runs at d$t = 1$ ms. The parameters for the quadrotor are

Table 2. Algorithm parameters.

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

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



$m = 0.5$ kg, $g = 9.81$ m/s^2, $J = \text{diag}(J_1, J_2, J_3)$, $J_1 = 0.0135$ kgm^2, $J_2 = 0.0135$ kgm^2, and $J_3 = 0.024$ kgm^2.

4.2. Reward design

An important factor in the RL motion control performance is the choice of the reward function's dependence on $a = [u, \tau^T]^T$ and $s = [p^T, \eta^T, v^T, \omega^T]^T$. We consider three different reward functions to investigate this dependence:

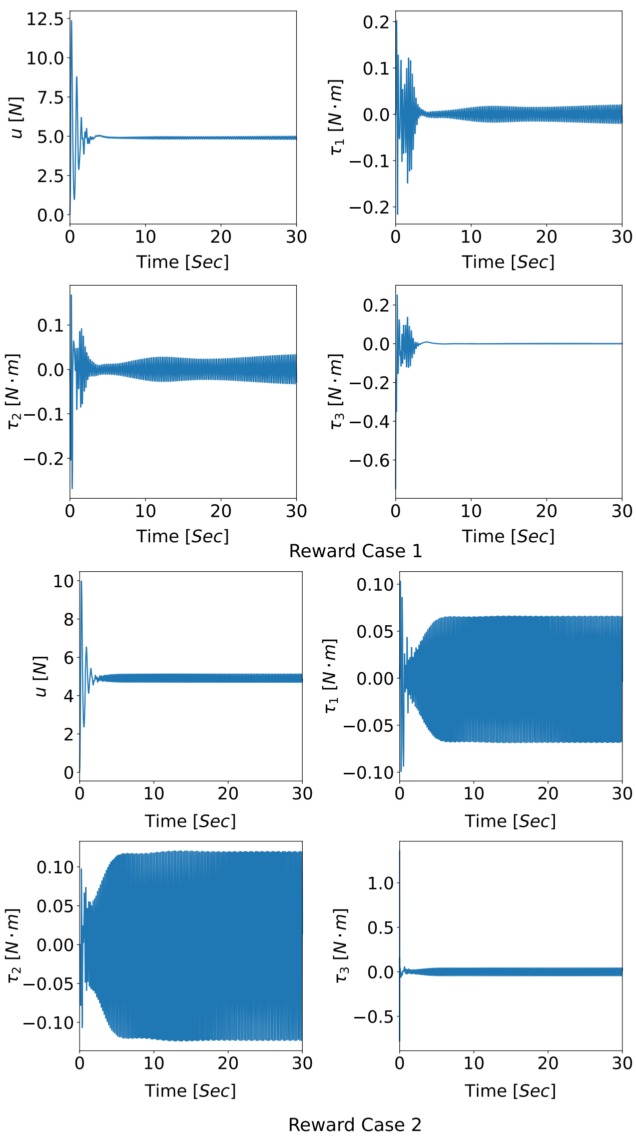
$$r_a(a,s) = r_0 - \|v\|_2 - \|\omega\|_2 \tag{25}$$

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

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

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

Fig.6. Simulation results for hover stabilization. Input trajectories for reward cases 1 and 2.



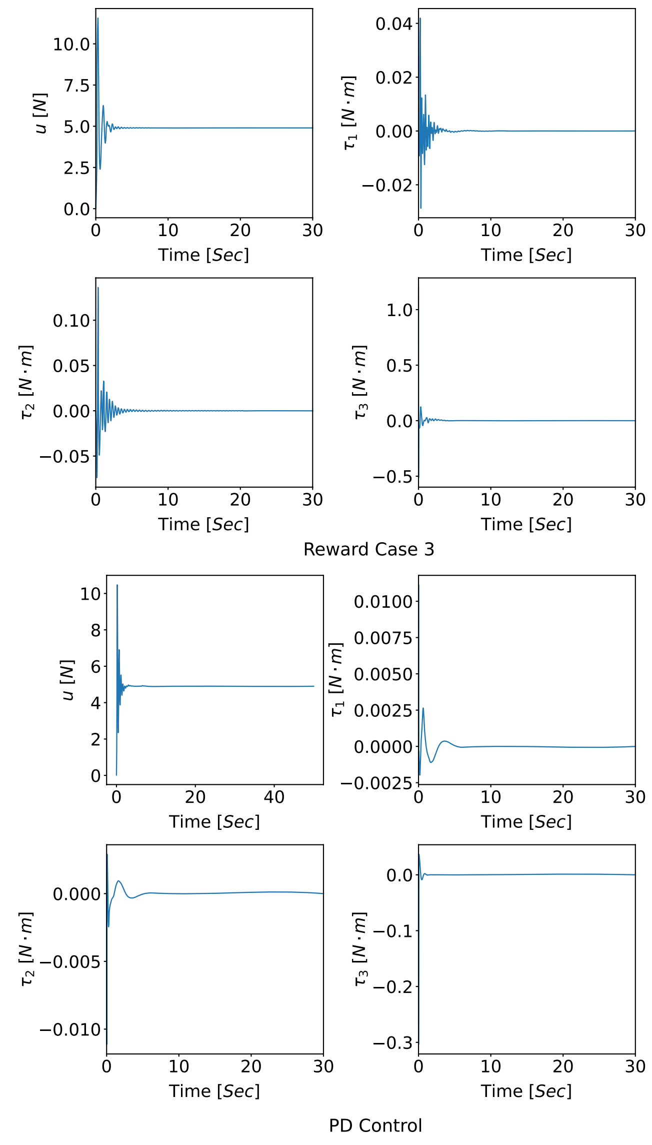
where r_0 is used to accelerate training. Reward r_a aims to control hover using only velocity. Reward r_b includes an extra term $||p||_2$ to reduce drift in position due to unmodelled disturbances. Reward r_c includes $||\tau||_2$ to limit control effort. The yaw rate of the vehicle is regulated to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

4.3. Simulation results

4.3.1. Training

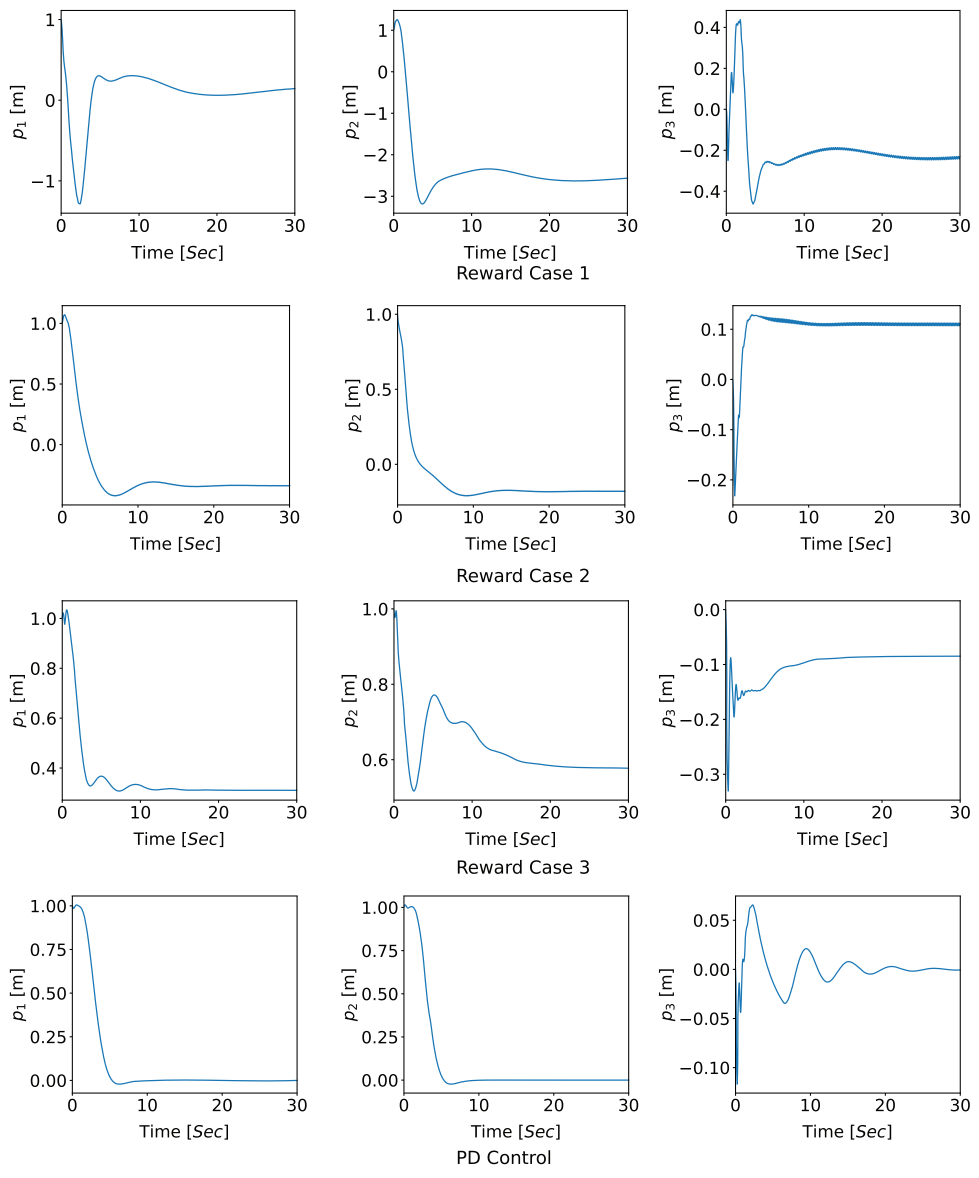
To optimize training, the hyperparameters λ and ε, and learning rates α and β were adjusted for reward function r_c. Parameter λ appears in the generalized advantage estimator (15). As $\lambda \to 1$ the bias of the estimator decreases. Although the bias will increase as $\lambda \to 0$, smaller λ may accelerate the training process. Parameter ε is the clip ratio in (18).

Fig. 7. Simulation results for hover stabilization. Input trajectories for reward case 3 and PD control.



It constrains the difference between the old and new policies. 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 $(\alpha, \beta) \in \{(3 \times 10^{-3}, 1 \times 10^{-2}), (3 \times 10^{-4}, 1 \times 10^{-3}), (3 \times 10^{-5}, 1 \times 10^{-4})\}$. This yielded 27 parameter combinations. The training was defined to fail if there is convergence to a local maximum that is unable to stabilize the quadrotor. The hyperparameter search led to successful training when $\lambda = 0.95$ for the values of ε, α, and β given in Table 1. We denote V_{avg} as the average cumulative reward for a single trajectory collected in the training process using six different random seeds. Its value is shown in Fig. 4 for the five 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

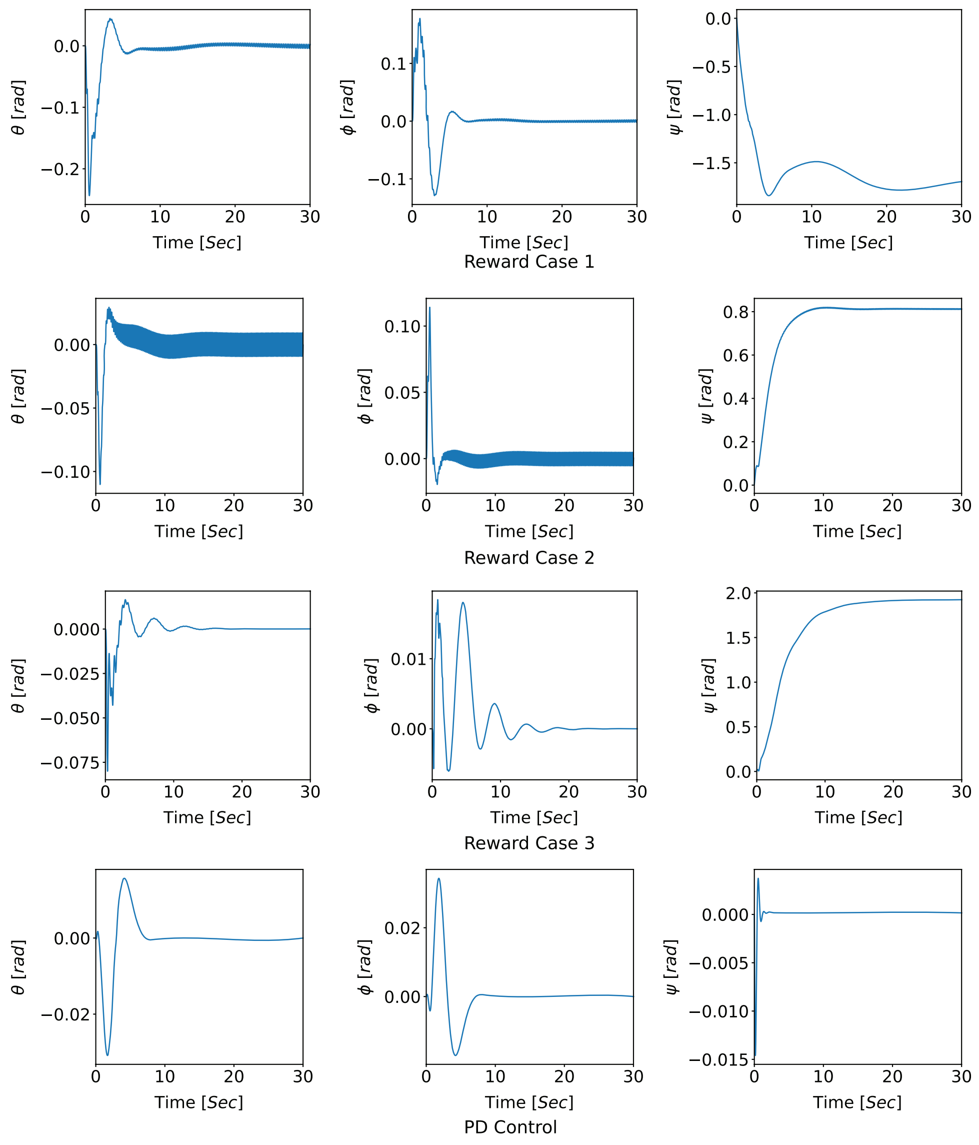
Fig. 8. Simulation results for hover stabilization. Position trajectories.



the final training process for all reward functions considered. The remaining design parameter values are shown in Table 2. The training process took about 12 h on a single core of a E5-2683 v4 Intel Broadwell CPU.

The training progress for the three reward functions is in Fig. 5, which shows the average cumulative reward V_{avg} for a single trajectory using six different random seeds. We observe that 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. Training convergence is achieved for the three reward cases, and their time domain performance differs as discussed below.

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

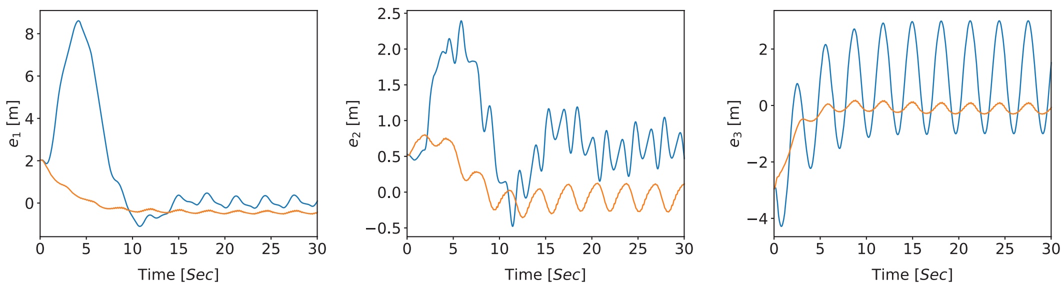


4.3.2. *Performance testing*

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

$$\begin{aligned}
&\phi_r = k^{\mathrm{o}}_{1\mathrm{p}}(p_{r1} - p_1) + k^{\mathrm{o}}_{1\mathrm{d}}\frac{d}{dt}(p_{r1} - p_1) \qquad \theta_r = k^{\mathrm{o}}_{2\mathrm{p}}(p_{r2} - p_2) + k^{\mathrm{o}}_{2\mathrm{d}}\frac{d}{dt}(p_{r2} - p_2)\\
&\tau_1 = k^{\mathrm{i}}_{1\mathrm{p}}(\phi_r - \phi) + k^{\mathrm{i}}_{1\mathrm{d}}\frac{d}{dt}(\phi_r - \phi) \qquad \tau_2 = k^{\mathrm{i}}_{2\mathrm{p}}(\theta_r - \theta) + k^{\mathrm{i}}_{2\mathrm{d}}\frac{d}{dt}(\theta_r - \theta)\\
&\tau_3 = -k_{3\mathrm{p}}\psi \qquad u = k_{\mathrm{hp}}(p_{r3} - p_3) + k_{\mathrm{hd}}\frac{d}{dt}(p_{r3} - p_3)
\end{aligned} \tag{29}$$

Fig. 10. Simulation results for trajectory tracking error: orange line for NN control based on r_c; blue line for PD controller.



where $p_r = [p_{r1}, p_{r2}, p_{r3}]^T$ is the reference position, ϕ_r and θ_r are roll and pitch set points, respectively. The control gains are chosen as $k^o_{1p} = k^o_{2p} = k^i_{1p} = k^i_{2p} = 0.07$, $k^o_{1d} = k^o_{2d} = k^i_{1d} = k^i_{2d} = 0.09$, $k_{3p} = 0.1$, $k_{hp} = 0.1$, $k_{hd} = 8$.

The control objective is to stabilize the position $p_r = [0, 0, 0]^T$ m with the vehicle initialized to $p(0) = [1, 1, 0]^T$ m. The results are given in Figs. 6–9. We observe for r_a that velocity is regulated to zero and ϕ, θ have small oscillations in steady state. These oscillations also appear in the input. 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 ϕ, θ is increased. This is likely due to position being added to r_b. As reward r_c includes the input, it provides less oscillatory input trajectories. Reward r_c provide acceptable position regulation.

Time-varying trajectory tracking is tested using r_c and compared with the PD controller. A figure-eight reference trajectory $p_r(t) = [1.5\sin(2t) + 1, 0.75\sin(4t), -4 + 2\sin(2t)]^T$ m is used. The tracking error is shown in Fig. 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 PPO to control the full six DoF system dynamics of a quadrotor UAV. Relative to existing work, 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. We observed that although different reward functions achieve stable hover, including input effort in the reward makes the closed-loop less sensitive (e.g., reduces impractical oscillation in the input). A simulation comparison of the 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 that provide monotone performance improvement during training.

References

Abbeel, P., Coates, A., and Ng, A.Y. 2010. Autonomous helicopter aerobatics through apprenticeship learning. Int. J. Rob. Res. **29**(13): 1608–1639. doi: 10.1177/0278364910371999.

Achiam, J. 2018. Spinning up in deep reinforcement learning. OpenAI. Available from https://spinningup.openai.com/en/latest/.

Bangura, M. 2017. Aerodynamics and control of quadrotors. Ph.D. thesis, College of Engineering and Computer Science, The Australian National University.

Bøhn, E., Coates, E.M., Moe, S., and Johansen, T.A. 2019. Deep reinforcement learning attitude control of fixed-wing UAVs using proximal policy optimization. *In* 2019 International Conference on Unmanned Aircraft Systems (ICUAS), Atlanta, GA, 11–14 June 2019. IEEE. pp. 523–533.

Bouabdallah, S. 2007. Design and control of quadrotors with application to autonomous flying. Ph.D. thesis. École polytechnique fédérale de Lausanne. doi: 10.5075/epfl-thesis-3727.

Cao, N., and Lynch, A. 2020. Time-delay robustness analysis of a nested saturation control for UAV motion control. Control Strategy for Time-Delay Systems, Part II: Engineering Applications. *Edited by* M.-H. Khooban and T. Dragicevic. Academic Press. pp. 69–93. doi: 10.1016/B978-0-32-385347-7.00008-0.

Cao, N., and Lynch, A. 2021. Predictor-based control design for UAVs: Robust stability analysis and experimental results. Int. J. Cont. **94**(6): 1529–1543. doi: 10.1080/00207179.2019.1660408.

Cao, N. and Lynch, A.F. 2015. Inner-outer loop control with constraints for rotary-wing UAVs. *In* 2015 International Conference on Unmanned Aircraft Systems (ICUAS). IEEE. pp. 294–302. doi: 10.1109/ICUAS.2015.7152303.

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

Goodfellow, I., Bengio, Y., Courville, A., and Bengio, Y. 2016. Deep learning. Vol. 1. MIT Press, Cambridge.

Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. 2018. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *In* International Conference on Machine Learning, Stockholm, Sweden, 10–15 July 2018. PMLR. pp. 1861–1870.

Heess, N., Dhruva, T.B., Sriram, S., Lemmon, J., Merel, J., Wayne, G., et al. 2017. Emergence of locomotion behaviours in rich environments. arXiv preprint. arXiv: 1707.02286.

Helder, B., Van Kampen, E.-J., and Pavel, M. 2021. Online adaptive helicopter control using incremental dual heuristic programming. *In* AIAA Scitech 2021 Forum, 11–15 and 19–21 January 2021. p. 1118. doi: 10.2514/6.2021-1118

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

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

Kaufmann, E., Loquercio, A., Ranftl, R., Müller, M., Koltun, V., and Scaramuzza, D. 2021. Deep drone acrobatics. *In* Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, (IJCAI-21), Montreal, QC, 19–26 August 2021. *Edited by* Z.-H. Zhou. pp. 4780–4783. doi: 10.24963/ijcai.2021/650.

Kober, J., Bagnell, J.A., and Peters, J. 2013. Reinforcement learning in robotics: A survey. Int. J. Rob. Res. **32**(11): 1238–1274. doi: 10.1177/0278364913495721.

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

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

Lillicrap, T.P., Hunt, J.J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., et al. 2015. Continuous control with deep reinforcement learning. arXiv preprint arXiv: 1509.02971.

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.

OpenAI, Akkaya, I., Andrychowicz, M., Chociej, M., Litwin, M., McGrew, B. et al. 2019. Solving Rubik's cube with a robot hand. arXiv preprint arXiv:1910.07113.

Polvara, R., Patacchiola, M., Sharma, S., Wan, J., Manning, A., Sutton, R., and Cangelosi, A. 2018. Toward end-to-end control for UAV autonomous landing via deep reinforcement learning. *In* 2018 International Conference on Unmanned Aircraft Systems (ICUAS), Dallas, TX, 12–15 June 2018. IEEE. pp. 115–123.

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

Ruggiero, F., Lippiello, V., and Ollero, A. 2018. Aerial manipulation: A literature review. IEEE Rob. Autom. Lett. **3**(3): 1957–1964.

Schulman, J., Levine, S., Abbeel, P., Jordan, M., and Moritz, P. 2015*a*. Trust region policy optimization. *In* International Conference on Machine Learning. Lille, France, 7–9 July 2015. PMLR. pp. 1889–1897.

Schulman, J., Moritz, P., Levine, S., Jordan, M., and Abbeel, P. 2015*b*. High-dimensional continuous control using generalized advantage estimation. *In* Proceedings of the International Conference on Learning Representations (ICLR), San Juan, Puerto Rico, 2–4 May 2016.

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

Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., and Riedmiller, M. Jan. 2014. Deterministic policy gradient algorithms. *In* International Conference on Machine Learning, Beijing, China, 21–26 June 2014. pp. 387–395. Available from http://proceedings.mlr.press/v32/silver14.html.

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., et al. 2017. Mastering the game of Go without human knowledge. Nature **550**: 354–359. doi: 10.1038/nature24270.

Sutton, R.S. and Barto, A.G. 2018. Reinforcement learning: an introduction. MIT Press.

Tang, S., and Kumar, V. 2018. Autonomous flight. Annu. Rev. Control Rob. Auton. Syst. **1**(1): 29–52.

Valavanis, K.P. and Vachtsevanos, G.J. 2015. Handbook of unmanned aerial vehicles. Vol. 2077. Springer, Dordrecht. doi: 10.1007/978-90-481-9707-1.

Villa, D.K., Brandao, A.S., and Sarcinelli-Filho, M. 2020. A survey on load transportation using multirotor UAVs. J. Intell. Robot. Syst. **98**(2): 267–296. doi: 10.1007/s10846-019-01088-w.

Zhang, Y., Zhang, Y., and Yu, Z. 2021. Path following control for UAV using deep reinforcement learning approach. Guid. Navigat. Cont. **10**(1): 2150005.

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

ZifeiFFF and Jiang, M. 2021. ANCL/QuadPPO. v1.0.0. Available from https://github.com/ANCL/QuadPPO. doi: 10.5281/zenodo.5669006.