2021 18th International Conference on Electrical Engineering, Computing Science and Automatic Control (CCE).
Mexico City, Mexico. November 10-12, 2021

Low-Level Control of a Quadrotor using Twin Delayed Deep Deterministic Policy Gradient (TD3)

Mazen Shehab
Mechatronics Engineering Department
German University in Cairo
Cairo, Egypt
mazen.shehab@student.guc.edu.eg

Ahmed Zaghloul
Mechatronics Engineering Department
German University in Cairo
Cairo, Egypt
ahmed.diyaaeldeen@student.guc.edu.eg

Ayman El-Badawy
Mechatronics Engineering Department
German University in Cairo
Cairo, Egypt
ayman.elbadawy@guc.edu.eg

***Abstract*—Unmanned Aerial Vehicles (UAVs) like Quadrotors are inherently under-actuated and unstable systems. The mechanical complexity and non-linearity of such systems make it a difficult task to control the flight of the mentioned systems. However, due to recent advancements in the fields of data science and machine learning, new algorithms for flight stabilization and trajectory control were developed using Deep Reinforcement Learning. This paper presents two low-level Quadrotor controllers based on the same algorithm. The first designed controller aims to stabilize the Quadrotor at a certain preset point given any random initial position. The second is to track any target position given in the 3D space. Twin Delayed Deep Deterministic Policy Gradient (TD3) is used to train the agents to achieve the required tasks. This method is an off-policy Actor-Critic based method. It was used as it does not require a system model and works on environments with continuous action and state spaces. The superb performance of the trained policies is demonstrated in a simulation to illustrate the effectiveness of the proposed controllers.**

***Index Terms*—Quadrotor, Machine Learning, Reinforcement Learning, TD3**

I. INTRODUCTION

Over the last decade, Quadrotors have been seeing a lot of use in commercial and research fields more than any other type of micro aerial vehicle. Though they are under-actuated systems, they are very versatile. They are capable of vertical takeoffs and landings and performing aggressive maneuvers.

Since Quadrotors are under-actuated and non-linear systems, controlling such vehicles proves to be a difficult task. The field of flight controllers is still growing as more efficient control methods are introduced. Therefore, researchers have tried to exploit recent advances in Reinforcement Learning to train agents that could control Quadrotors to achieve different tasks. Such tasks include flight stabilization, takeoffs/landings, and position tracking.

One of the controllers designed to perform low-level control of Quadrotors was developed and implemented in [1]. A Deep Reinforcement Learning algorithm was used which contains two neural networks that take as an input the state of the agent and output either the state-value function or the optimal action. The two networks consist of 2 hidden layers with 64 nodes and a $tanh$ activation function. The state of the agent is 18-dimensional and contains the position, orientation, rotation matrix, linear and angular velocities. A new training algorithm that has low variance was devised based on deterministic policy optimization. A simple exploration strategy was used to obtain finite length trajectories for each episode. The value network target was calculated using Monte Carlo's method while the policy network used natural gradient descent. The resulting policy was able to respond to step input and stabilize the rotor under harsh initial conditions.

Another paper used Policy Gradient-based Actor-Critic [2]. Random trajectories from random initial states were sampled. The return error was calculated using Temporal Difference (TD) and used to train the value network from the stored trajectories. Stochastic Gradient Descent was used to optimize the policy network. The networks took as input the state of the Quadrotor and each network contained 2 hidden layers consisting of 16 nodes and activation function $ReLU$. The smaller size of the networks reduced computation time.

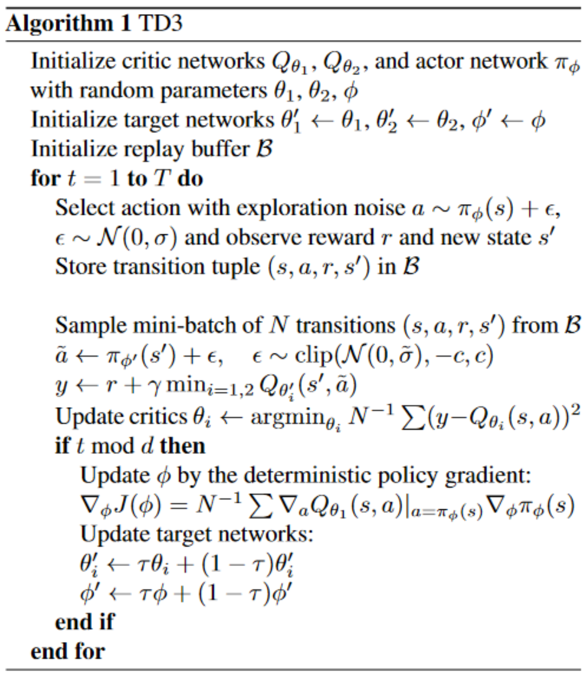
In other work, a stochastic control policy for a model-free agent to perform position control of a Quadrotor was proposed [3]. The paper used Proximal Policy Optimization (PPO). For training, the Quadrotor was initialized inside the environment in a random position and orientation while the reward was increased as it gets closer to the target. Two neural networks were used. Both neural networks had 2 hidden layers with 64 nodes. The quality of the controller was tested in simulation under three situations: Stochastic Policy, Deterministic Policy, Deterministic Policy with a moving target.

Furthermore, two algorithms were trained to control a Quadrotor and tested to see their performance in another paper [4]. The algorithms used are Advantage Actor-Critic (A2C) and Proximal Policy Optimization (PPO). Both methods successfully stabilized the Quadrotor at the determined target. PPO was found to learn faster and achieve better performance than A2C however a tuned PID controller performed better than both.

In this paper, two controllers were designed which are based on a Deep Reinforcement Learning technique. The controllers have the aim of stabilizing a Quadrotor at a certain preset point and position tracking. The method used is Twin Delayed Deep Deterministic Policy Gradient (TD3). The proposed controllers can be used to track any given trajectory of any shape with known points and to stabilize the Quadrotor at any given target. This was not proposed in the literature.

II. Twin Delayed Deep Deterministic Policy Gradient (TD3)

In this paper, TD3 is used to train an agent capable of stabilizing a Quadrotor at a point from a random initial position and another agent capable of tracking a target point. The TD3 algorithm (shown in Figure 1) was introduced to try to solve the value overestimation problem of the value networks in Actor-Critic based methods such as DDPG [5]. It introduced several new improvements to its predecessor. The first of these



```
Algorithm 1 TD3
Initialize critic networks $Q_{\theta_1}$, $Q_{\theta_2}$, and actor network $\pi_\phi$
with random parameters $\theta_1$, $\theta_2$, $\phi$
Initialize target networks $\theta'_1 \leftarrow \theta_1$, $\theta'_2 \leftarrow \theta_2$, $\phi' \leftarrow \phi$
Initialize replay buffer $\mathcal{B}$
for $t = 1$ to $T$ do
    Select action with exploration noise $a \sim \pi_\phi(s) + \epsilon$,
    $\epsilon \sim \mathcal{N}(0, \sigma)$ and observe reward $r$ and new state $s'$
    Store transition tuple $(s, a, r, s')$ in $\mathcal{B}$

    Sample mini-batch of $N$ transitions $(s, a, r, s')$ from $\mathcal{B}$
    $\tilde{a} \leftarrow \pi_{\phi'}(s') + \epsilon, \quad \epsilon \sim \text{clip}(\mathcal{N}(0, \tilde{\sigma}), -c, c)$
    $y \leftarrow r + \gamma \min_{i=1,2} Q_{\theta'_i}(s', \tilde{a})$
    Update critics $\theta_i \leftarrow \text{argmin}_{\theta_i} N^{-1} \sum (y - Q_{\theta_i}(s, a))^2$
    if $t$ mod $d$ then
        Update $\phi$ by the deterministic policy gradient:
        $\nabla_\phi J(\phi) = N^{-1} \sum \nabla_a Q_{\theta_1}(s, a)|_{a=\pi_\phi(s)} \nabla_\phi \pi_\phi(s)$
        Update target networks:
        $\theta'_i \leftarrow \tau \theta_i + (1 - \tau) \theta'_i$
        $\phi' \leftarrow \tau \phi + (1 - \tau) \phi'$
    end if
end for
```

Fig. 1. TD3 Algorithm [5]

improvements is target policy smoothing. Deterministic policy methods tend to have target values with high variance when updating the critic. This variance is decreased in TD3 by using a regularization technique known as target policy smoothing. This is achieved by adding a small amount of noise to the target. Thus the updated target becomes:

$$y = r + \lambda Q_{\theta'}(s', \pi_{\phi'}(s') + \epsilon) \tag{1}$$

$$\epsilon \sim clip(\mathcal{N}(0, \sigma), -c, c)$$

Another improvement is the use of twin critic networks where the target for Q-function optimization uses the network with the lowest value.

$$y_1 = r + \lambda \min_{\text{i}=1,2} Q'_{\text{i}}(s', \pi_{\phi_1}(s')) \tag{2}$$

This favors the underestimation of Q (action-state) values. This bias poses no problem as the low values would not be propagated through the algorithm, unlike overestimated values.

The last improvement added is delayed updates of the value networks. This implies that we update the policy network less frequently than the critic network. This adds more stability to the algorithm as the value network is allowed to become more stable and reduce errors before it is used to update the policy network.

III. Method

In this section, the method used to train the agents to obtain the best policy for stabilization and target tracking is discussed.

A. Model Used in Simulation

In this paper, the dynamic model of the Quadrotor in the simulation is a floating body model with four thrust forces of the rotors which can be seen in Figure 2 (F_1,F_2,F_3,F_4). The equation of motion can be represented as follows:

$$J^T T = Ma + h \tag{3}$$

where J is the Jacobian matrix, T is the rotors' thrusts, M is the inertia matrix, a is the generalized acceleration, and h is the coriolis and gravity effect [1].

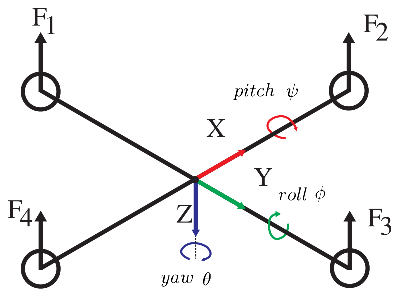


Fig. 2. Simplified Quadrotor Model

B. Simulation Environment

The environment used for training the agent is gym-pybullet-drones [6]. It is a simple gym environment based on the physics engine pybullet for single-agent and multi-agent Reinforcement Learning problems. The environment was implemented and modified to fit the task requirements.

C. Simulation Conditions

Since the environment is continuous, to limit the number of the states and to save time in the training process, episodes were terminated if certain conditions were met. These conditions are:

- $\phi\,,\,\theta\,,\,\psi > \frac{\pi}{2} \quad or \quad \phi\,,\,\theta\,,\,\psi < -\frac{\pi}{2}$
- $x\,,\,y\ >\ 3\,m \quad or \quad x\,,\,y\ <\ -3\,m$
- $z\ <\ 0 \quad or \quad z > 3\,m$

D. Network Structure

The TD3 method uses six networks; four critic networks and two actor networks. The two actor networks take as an input the state while the four critic networks take as an input both the state and the action. The state or observation consists of 15 elements which are position, orientation, linear velocity, angular velocity, and the difference between the drone's global x,y,z and the target's global x,y,z coordinates.

$$[x, y, z, \phi, \theta, \psi, v_x, v_y, v_z, \omega_x, \omega_y, \omega_z, \Delta x, \Delta y, \Delta z]$$

The output of the networks however will differ. For the four critic networks, the output is the Q (action-state) value while for the two actor networks, there will be four outputs which are the drone's motors' speeds in rpm. All of the networks consist of two hidden layers of 400 and 300 nodes respectively with $ReLU$ activations and an output layer with a $tanh$ activation function.

E. TD3 Hyperparameters

The hyperparameters used in training this agent have been chosen through trial and error. The discount factor γ was set to be 0.99. Furthermore, the learning rate was set to α = 7e - 4 for both the actor and critic networks. To ensure enough exploration is conducted, a noise N(0,σ) is added where σ was set to 0.2. A batch size of 256 was used along with a buffer size of 1e6. The maximum time of an episode was set to 10 seconds.

F. Reward Function

Generally, there are two types of reward functions in Reinforcement Learning: sparse reward functions and dense reward functions. Sparse reward functions give rewards in only some specific states and these rewards are generally high. On the other hand, dense reward functions give smaller rewards much more often. In this paper, dense reward functions are used due to the continuous nature of the tasks at hand. If we used sparse reward functions, however, the agent will not be able to reach the goal due to the insufficient rewards. Two different reward functions were used; one for each objective. For the first objective, the reward function is defined as follows:

$$r = \begin{cases} tanh[1 - p_1 \Delta(x,y,z)^2], & \text{if } r \geq 0 \\ 0, & \text{otherwise} \end{cases} \tag{4}$$

$\Delta(x,y,z)^2$ is the difference between the target position and the drone's current position while p_1 is just a weight parameter and is set to [10,10,20] to have a high negative reward on position control. For the second objective which is tracking given target points, a similar reward function was used but an additional term was added. The new reward function is defined as follows:

$$r = \begin{cases} tanh[1 - p_1 \Delta(x,y,z)^2] - p_2(\frac{\partial u}{\partial t})^2, & \text{if } r \geq 0 \\ 0, & \text{otherwise} \end{cases} \tag{5}$$

p_1 was also set to [10,10,20]. The result however will now be diminished by the derivative of action with respect to time squared. This decreases the oscillations in the action signals resulting in a smooth action signal that puts less effort on the motors. p_2 was set to 0.8 to give small negative rewards in the case of high changes in the control output of the agent. On the other hand, a high value of p_2 results in a 'lazy' agent.

G. Simulation Objective and Initial Conditions

Two different agents were trained. The first agent should be able to stabilize the Quadrotor at a certain preset target point starting from any random position. While the aim of the second agent is to track any given target point. The simulation conditions for both pieces of training will be mentioned next.

1) Stabilization Training: This agent aims to reach the intended point (0,0,1) starting from any random position. So, the Quadcopter should always go to this predetermined point regardless of the starting position. The initial x,y positions of the Quadrotor varies between (-2.5,2.5) while the initial z position varies between (0.2,2).

2) Trajectory Tracking Training: The objective of this agent is to track any given point. So, in this training, the target point was not fixed at a certain position during the entire training. The target position was chosen randomly every 500 time steps where the target x,y positions vary between (-2.5,2.5) while the target z position varies between (0.2,2). Furthermore, the initial position of the Quadrotor was changed randomly every episode the same way it did in the previous training.

H. Network Training

The neural networks were trained using the aforementioned conditions and parameters. The training was done with the use of NVIDIA CUDA, on a GeForce RTX 3090 GPU. Furthermore, implementation was done using the python package Torch. The training for both policies was run for 20 million time steps each.

IV. Results

In this section, the results obtained during training are presented and analyzed. Moreover, the response of both trained policies was tested and the obtained results are stated to show the effectiveness of the trained policies.

A. Stabilization Training Results

The aim of this training was to obtain a policy that is able to stabilize the Quadrotor at a given preset point which is (0,0,1) in this case starting from any random initial position. Figure 3 shows the mean reward obtained per time step. The maximum mean reward reached by the algorithm was 156.49 at 1.685e6 time steps. Furthermore, Figure 4 shows the mean episode length per time step. The maximum episode length was achieved nearly at 1,700,000 time steps. This shows that the best policy was obtained at 1.7 million time steps.

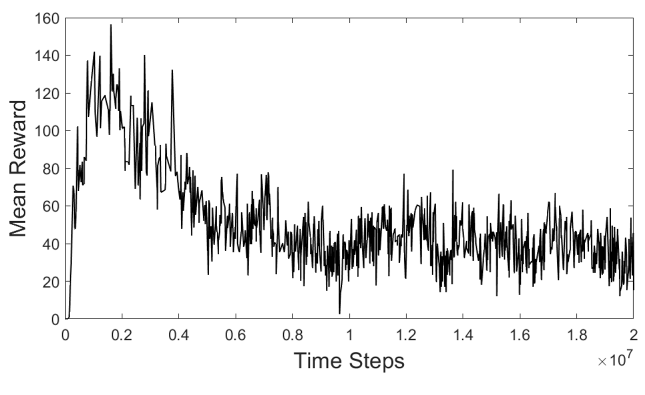


Fig. 3. Mean Reward During Stabilization Training

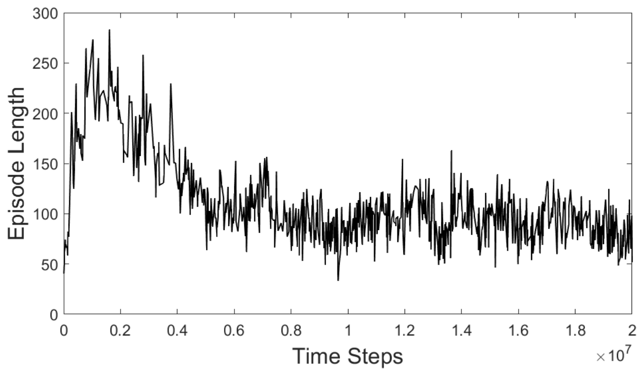


Fig. 4. Mean Episode Length During Stabilization Training

The trained policy is capable of stabilizing the drone at the specified target starting from any random initial point. The agent started stabilizing at the target point after 250,000 time steps. After that, the agent reaches the target faster and stabilizes faster till a certain point which is at about 1.7 million time steps; this is showed by the mean reward.

Figure 5 shows the response of the Quadrotor following the best policy when starting from an arbitrary random position till reaching the intended point which is (0,0,1) and stabilizing there.

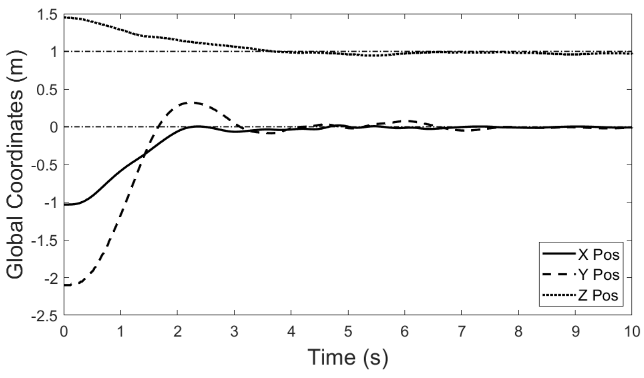


Fig. 5. Quadrotor's Position (Stabilization Testing)

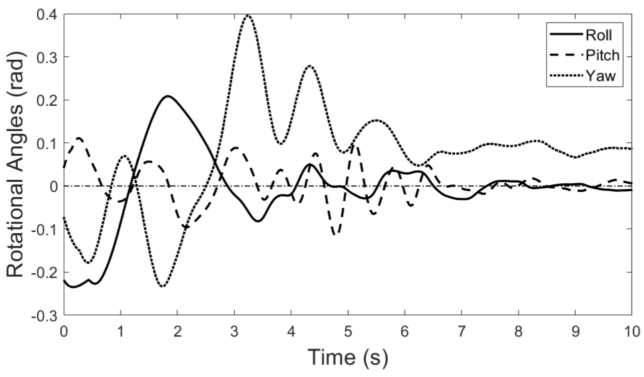


Fig. 6. Quadrotor's Orientation (Stabilization Testing)

Figure 6 shows the orientation change of the Quadrotor. The roll and pitch angles approached zero while the yaw angle approached a certain steady-state value. This means that after reaching the target position, the Quadrotor stabilizes since the roll and pitch angles approached zero. Also, the yaw angle reached a certain steady-state value which means that the Quadrotor is trying to keep pointing in the same direction and not rotate.

B. Position Tracking Training Results

The objective of this training is to obtain a policy capable of making the Quadrotor reach any given target position starting from a random initial position. Figure 7 shows the mean reward obtained per time step. As can be seen from the graph, the maximum mean reward reached was 199.6 at 13.35e6 time steps.

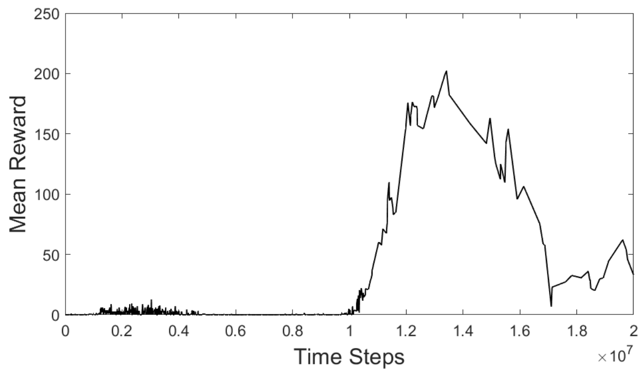


Fig. 7. Mean Reward During Position Tracking Training

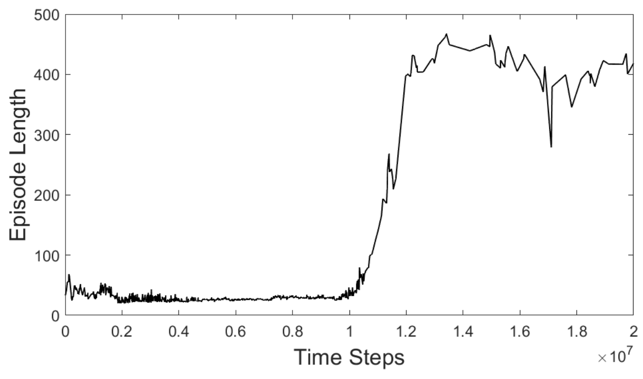


Fig. 8. Mean Episode Length During Position Tracking Training

Also, Figure 8 shows the mean episode length per time step. Same as the mean reward did, the maximum mean episode length was achieved at 13.35 million time steps. This tells us that the optimal policy is achieved at 13.35 million time steps.

The trained policy is capable of making the drone track any given target position starting from a random position. To demonstrate this, the response of a single episode was plotted. Figure 9 shows the response of the trained policy at 13.35 million time steps when starting from a random initial position till reaching the predefined target which is also random in this case.

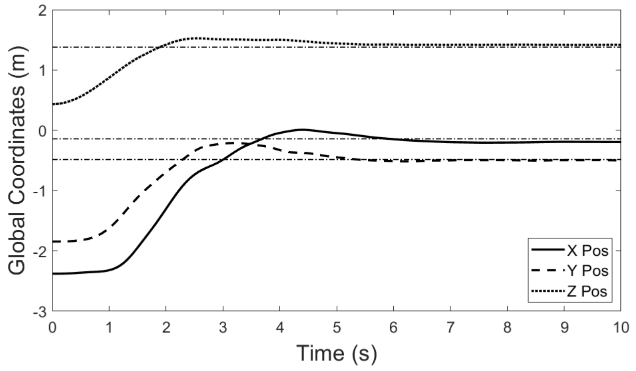


Fig. 9. Quadrotor's Position (Position Tracking Testing)

The plots show that the Quadrotor was able to reach the specified random target after 5.5 seconds which is similar to the stabilization training.

Furthermore, Figure 10 shows the orientation change of the Quadrotor. These figures show that the Quadrotor was able to hover at the specified target after reaching it. This can be seen as the roll and pitch angles of the Quadrotor approached zero while the yaw is approaching a certain steady-state value.

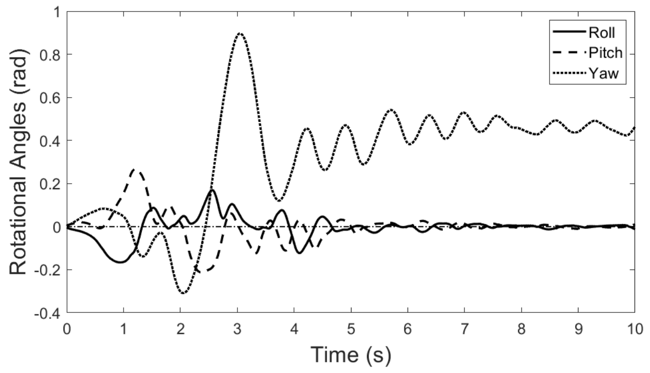


Fig. 10. Quadrotor's Orientation (Position Tracking Testing)

To test the trained network in trajectory tracking, a trajectory was generated. The trajectory used to analyze the flight behavior is a figure 8 trajectory. In this trajectory, the Quadrotor flies at the same height which is 1 meter. The points that compose the path of the drone are derived from the Lemniscate of Bernoulli which has the following parametric equations:

$$x = \frac{2\cos(t)}{1+\sin^2(t)} \qquad y = \frac{2\sin(t)\cos(t)}{1+\sin^2(t)} \tag{6}$$

So, the points to build this trajectory were defined and passed to the trained policy. A single target position is passed to the Quadrotor at once and this target is changed when the distance between the target and the Quadrotor's position is less than 0.01 m. The upcoming figure shows the path that the Quadrotor followed.

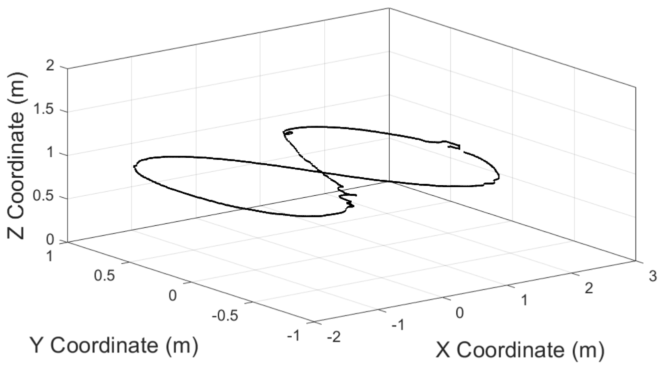


Fig. 11. Figure 8 Trajectory (constant elevation)

Figure 12 shows the x and y positions of the Quadrotor. These response plots show that the Quadrotor followed a smooth path and was able to track the given path. This ensures that the trained policy can track any given points or any given path smoothly.

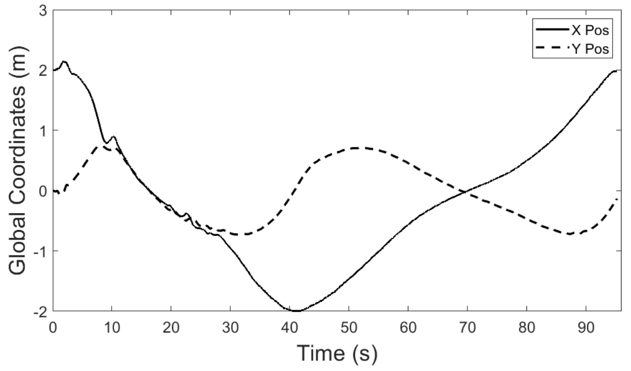


Fig. 12. Quadrotor's Position (Constant Elevation Figure 8 Trajectory)

To further test the trained policy, position tracking for a different figure 8 trajectory was implemented however this time the trajectory did not have a constant elevation. This means that now not only the x and y coordinates will differ like the previous trajectory, but also the z coordinate will differ. The trajectory is defined by the following parametric equation which is also derived from the Lemniscate of Bernoulli:

$$x = \frac{0.5\cos(t)}{1+\sin^2(t)} \qquad y = \frac{0.5\sin(t)\cos(t)}{1+\sin^2(t)} \qquad z = x + 1.5 \tag{7}$$

The equations were used to define the points to construct this trajectory and then they were passed to the trained policy as target positions. Figure 13 shows the path followed by the Quadrotor.

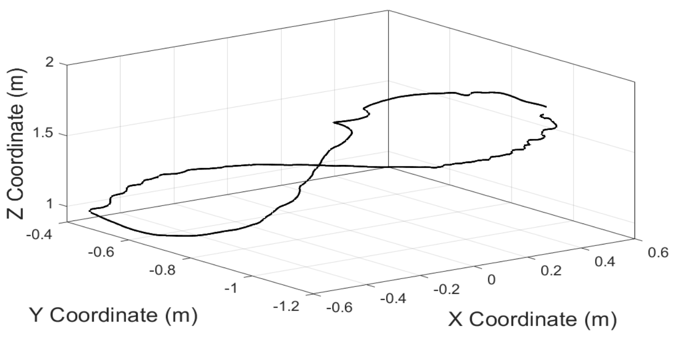


Fig. 13. Figure 8 Trajectory (different elevation)

From Figure 13, the Quadrotor successfully followed the path provided while remaining stable throughout the process. Figure 14 shows the x, y, and z positions of the Quadrotor as it tries to follow the given trajectory.

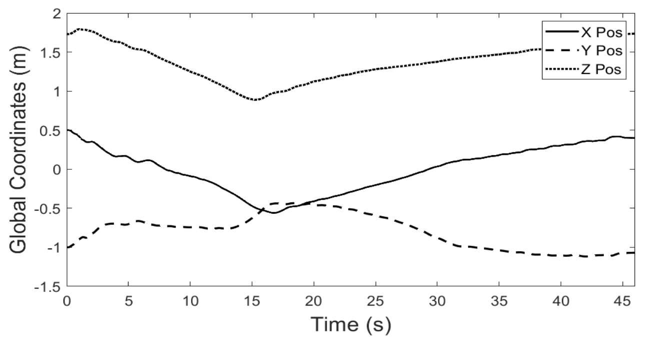


Fig. 14. Quadrotor's Position (Different Elevation Figure 8 Trajectory)

During training, the performance drop apparent in the reward graphs could be attributed to several reasons. One of the main reasons is catastrophic forgetting in which after reaching a good performance, the agent will only visit certain states, skewing the state distribution in the replay buffer. This will greatly affect gradient updates of the value network which then provides wrong value estimates of certain states [7]. It may also be attributed to overfitting the neural networks as they are large or to the discontinuities of the Q function.

A comparison between the different approaches in this paper and the literature is presented in "Table I". However, it should be noted that in both pieces of training, the Quadrotor started in a random position each episode while in [1] and [2] the position, orientation, linear velocity, and angular velocity were all randomized each episode and in [3] the position and orientation were randomized each episode.

V. Conclusion

In this paper, a neural network policy for Quadrotors was trained using TD3 in a model-free manner. The trained policy showed good performance in stabilization and trajectory tracking. The policy tracked two different trajectories successfully with small steady-state error. TD3 allowed less training time as it's more sample efficient than other methods discussed in the literature.

Future work should consider using better models in simulation to capture more aspects of the system dynamics. Also, new neural network models should be considered. Using RNN and LSTM could help capture errors in modeling and adapt to new conditions. Another aspect is the online learning of the agent in real-world settings to adapt to real-world uncertainties.

TABLE I
Results Comparison

Comparison Parameters	**This Paper**	**Paper [1]**	**Paper [2]**	**Paper [3]**
Stabilization Training (Time Steps)	1.7 million	About 150 million (25 minutes of training, each iteration took less than 10 seconds, 1 million time steps per iteration)	10 million	7.5 million
Position Tracking Training (Time Steps)	13.35 million	———	About 20 million (0.1 million episodes, maximum episode length is 200 time steps)	———
Steady-State Error (X)	0.0164 m	0.013 m Displacement	———	0.34 m
Steady-State Error (Y)	0.0308 m		———	0.225 m
Steady-State Error (Z)	0.0652 m		———	0.41 m
Settling Time	2 seconds	———	2 seconds	———

References

[1] Jemin Hwangbo, Inkyu Sa, Roland Siegwart, and Marco Hutter. "Control of a Quadrotor with Reinforcement Learning", IEEE Robotics and Automation Letters, vol. 2, pp. 2096–2103, 2017.
[2] Chen-Huan Pi, Kai-Chun Hu, Stone Cheng, and I-Chen Wu. "Low-level autonomous control and tracking of quadrotor using reinforcement learning", Control Engineering Practice, vol. 95, pp. 104–222, 2020.
[3] Guilherme Cano Lopes, Murillo Ferreira, Alexandre da Silva Simões, and EstherLuna Colombini. "Intelligent Control of a Quadrotor with Proximal Policy Optimization Reinforcement Learning", in Proceedings of the 2018 Latin American Robotic Symposium, 2018 Brazilian Symposium on Robotics (SBR) and 2018 Workshop on Robotics in Education (WRE), pp. 503–508, 2018.
[4] Fang-I Hsiao, Chiang Cheng-Min, and Alvin Hou. "Reinforcement Learning Based Quadcopter Controller", Stanford University, 2019.
[5] Scott Fujimoto, Herke van Hoof, and David Meger. "Addressing Function Approximation Error in Actor-Critic Methods", in Proceedings of the 35th International Conference on Machine Learning, vol. 80, pp. 1587–1596, July 2018.
[6] Jacopo Panerati, Hehui Zheng, SiQi Zhou, James Xu, Amanda Prorok, and Angela P. Schoellig. "Learning to Fly—a Gym Environment with PyBullet Physics for Reinforcement Learning of Multi-agent Quadcopter Control", 2021.
[7] Zachary C. Lipton, Kamyar Azizzadenesheli, Abhishek Kumar, Lihong Li, Jianfeng Gao and Li Deng. "Combating Reinforcement Learning's Sisyphean Cursewith Intrinsic Fear", 2016.