Volume 9, Issue 6
December 2020

ITEE Journal
Information Technology & Electrical Engineering

ISSN: - 2306-708X

# Comparing the performance of Reinforcement Learning Algorithms in Reducing Energy Consumption during Quadcopter Attitude Control

**[1]Varun Agarwal and [2]Rajiv Ranjan Tewari**
[1]Centre of Computer Education, IPS, University of Allahabad, Allahabad, India
[2]Department of Electronics and Communication, University of Allahabad, Allahabad, India
E-mail: [1]varunag18@gmail.com, [2]tewari.rr@gmail.com

## ABSTRACT

The rotational motion of quadcopters is referred to as attitude control. It is a high energy consuming task. We propose using deep reinforcement learning to automate the attitude control process. We test the performance of two deep reinforcement learning algorithms Proximal Policy Optimization (PPO) and Deep Deterministic Policy Gradients (DDPG) on the reward function designed to prioritize minimization of energy consumption. Since the moving parts of the quadcopter are only the motor/propellers, hence they consume the maximum energy and our reward function is created to optimize their movements. Our simulations using DDPG and PPO on an Open AI Gym environment show that PPO performs better in our energy optimization problem.

**Keywords:** *attitude control, quadcopters, reinforcement learning, energy efficiency, proximal policy optimization, deep deterministic policy gradients*

## 1. INTRODUCTION

Quadcopters, a class of Unmanned Aerial Vehicles (UAVs), are finding uses in tasks like parcel deliveries, aerial photography, relief in disaster affected areas, surveillance, etc. UAVs can be either fixed wing or multirotors. Given the advantages of multirotors like the ability to hover, they have been researched more than fixed wing UAVS. Quadcopters is a class of multirotors having four wings/motors. Other variations include six motors, eight motors, etc. However, their capability of carrying weight is much restricted as compared to fixed wing UAVs. Batteries, the source of energy to a quadcopter, are a major portion of its weight. Thus, minimizing the energy depletion is a critical requirement to enhance the flight duration of a quadcopter. There are six degrees of freedom to a quadcopter - three rotational defined by the three axes in which a quadcopter can rotate and three translational defined by the motion from one place to another in the 3D space.

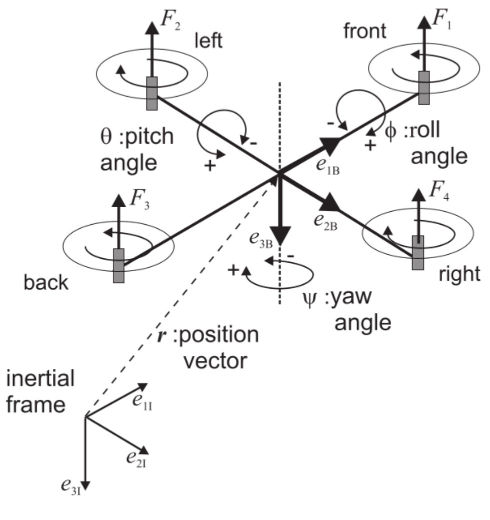

Fig. 1 Quadcopter's Roll, Pitch & Yaw Movements [1]

This movement, rotational or translational or both, is brought about by the rotors which are the only moving parts of a quadcopter. Altering the speed of the four rotating motors brings about the change in its position.

The angle at which a UAV flies relative to the ground is known as its attitude. The three axes of rotation of a quadcopter are the roll, pitch and yaw. Roll is the rotation about its longitudinal axis. When the quadcopter rotates laterally, it is called the pitch while yaw is the rotation when a quadcopter rotates in the clockwise or anticlockwise direction while remaining level to the ground. (Refer to Fig. 1).

Managing the rotation of the roll, pitch and yaw axes controls the overall quadcopter rotation and is called attitude control. Thus, a change in attitude simply requires changing the rotational speeds of the motors. Following a set pattern or combination in terms of the speeds of the individual motors can result in achieving attitude control. Now, this speed of rotation of the motors can be either managed by a human (which is an expensive proposition) or can be fed into the system from the start (which makes the system non robust and unable to adapt to environmental changes). The disadvantages of each of the above methods bring us to the novel idea of harnessing the ability of artificial intelligence in governing the quadcopter's movements.

We propose the use of Deep Reinforcement learning, a branch of artificial intelligence, for achieving attitude control while minimizing energy consumption in quadcopters. Reinforcement learning is a technique where objects learn from experience. The different components that make up a quadcopter include a flight controller, radio receiver, battery, and four arms. Each of the arms has an Electronic Speed Controller (ESC), motor and a propeller. The flight controller is made up of the Global Positioning System, Inertial Measurement Unit (IMU) while the IMU is itself made up of the Gyroscope, Magnetometer, Accelerometer, etc.

The Gyroscope records the roll, pitch and yaw values and hence our study shall be focused on it. Each of the ESCs passes the speed instructions to their respective motors driving the propellers. Thus, the ESCs dictate the rotational speeds and manage the attitude which is then measured by the Gyroscope.

In reinforcement learning, there is a problem with a set of probable solutions to that problem [2]. Reinforcement learning algorithms are designed to select one action out of the available options in order to maximize the output. Reinforcement learning maps situations to actions so as to achieve the best reward. Using deep reinforcement learning, deep neural networks are used to create an understanding of the world and then act upon it. The learning takes place through experience, i.e. the system is not trained by a human. Reinforcement learning is a closed loop technique where the output of one step becomes the input of the next. It is used in problems where the entire task is divided over multiple steps and decision making takes place sequentially.

Our contribution, in this research, can be broadly stated as: -

1. Selecting our Energy Efficient Attitude Control (EEAC) algorithm and applying it on quadcopter attitude control using an Open AI Gym environment.
2. Applying Deep Deterministic Policy Gradients (DDPG) and Proximal Policy Optimization (PPO) on the quadcopter attitude control problem.
3. Comparing the performances of both the techniques and displaying the results.

## 2. RELATED WORK

Across the literature, attitude control has been handled as a standalone problem or along with navigation control. We, first, discuss some papers tackling attitude control alone.

Pitch control has been tested by Jiang et al. [3] by implementing Aggregated Multi Reinforcement Learning. Alexis et al. [4] consider the influence of wind and other factors. The effect of drag and thrust are considered. [5] maps sensory data into motor velocity values. A deterministic, on-policy approach has been used. Lambert et al. [6] make changes in firmware, model adaptations and system design to deal with system and dynamic limitations.

Next, we shall discuss papers where attitude control is dealt as an inner loop and navigation control as the outer loop of the entire problem. Abbeel et al. in [7] use a hybrid algorithm on an approximate model. Real world scenarios are used to test the policy. A two host methodology is the approach used by Santos et al. in [8] where one host runs control loops and the other host runs the robot model. A Proportional Derivative equation is used to link linear acceleration and attitude control in [9]. TEXPLORE implements decision trees and forests to create learned models in [10].

Rodriguez-Ramos et al. [11] implement vision based landing using cameras, sensors and Inertial Measurement Units (IMUs). Li et al. [12] suggest a comprehensive reinforcement learning algorithm that is quarternion based. In [13], Nie et al. propose pitch angle in terms of velocity of quadcopter. AirSim's Woodland package has been used in [14] with two convolutional neural networks. In [15], a two step approach is deployed in which the weight of the centre of quadcopter and its inertial matrix is considered while calculating attitude control. In [16], Bekal et al. use Deep Deterministic Policy Gradients (DDPG) for pitch control and Proportional Integral Derivative (PID) controllers for roll and yaw. In [17], Proximal Policy Optimization (PPO) is used. Zhou et al. [18] present attitude control in mathematical terms implementing feedback linearization and using different channels for pitch, roll and yaw.

The rest of the paper is organized as follows: Section 3 details the terms used in our work. Section 4 is dedicated to the implementation details. In section 5, we evaluate our experiments and display results while the last section is used for the conclusion remarks.

## 3. METHODOLOGY

Here, we discuss the ideas employed in this research along with the terms used.

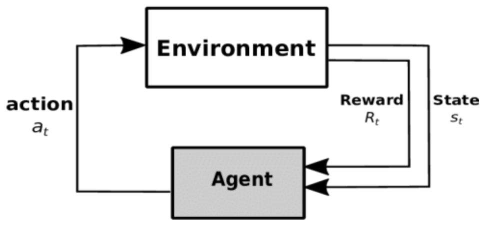

Fig. 2 Interaction between agent and environment in Reinforcement learning [19]

### 3.1 Terms used in Deep Reinforcement Learning

*3.1.1 Agent(Fig. 2)*

The agent is the heart of the deep reinforcement learning process. It is the neural network consisting of an input layer, an output layer and multiple hidden layers. A layer consists of nodes called neurons and has an activation function [2].

*3.1.2 Action*

Depicted by $A_t \in \mathcal{A}(S_t)$, action is the output of the agent. $(S_t)$denotes one of the states of the environment and is discussed later. The action may be an action value function $(q_\pi(s,a))$ or a state value function $(v_\pi(s))$. The value function defines how good an action or state is. The agent, at each step, creates a mapping between states and the probabilities of selecting an action. This mapping is called the *policy*.

*3.1.3 Policy*

The policy is denoted by $\pi_t$ or $\pi_t(a|s)$, implying that $A_t = a$ if $S_t = s$. A policy is: -

- Deterministic:
  Deterministic policies have a single action for a state. It is possible in environments having no uncertainty.
  $\pi_t(s) = a_t$

- Stochastic:
  Used in uncertain environments, stochastic policies have a probability distribution over actions.
  $\pi_t(a|s) = \mathcal{P}(a_t|s_t)$

*3.1.4 Environment:*

Anything external to the neural network is the environment. The agent interacts with it by sending the action and receiving the response. The quadcopter is the environment, in our case.

*3.1.5 State:*

It is the condition the environment is in. It forms the input of the agent.

*3.1.6 Reward:*

The reward is a number returned by the environment as a response to the agent's input. The agent tries to maximize the value of the reward at the end of the process. The reward is a function of the action taken by the agent and the current state of the environment.

The agent, depending on its representation of the state and the reward, creates a model.

Reinforcement learning algorithms are of two kinds.

<u>Model based:</u>

The agent estimates a model of the environment and decides the future plan based on how its actions change the environment. Model based approaches are very sample efficient. It is used in reinforcement learning algorithms used in playing games like Chess, Go, etc.

<u>Model free:</u>

Here, the model of the environment is not created. Such reinforcement learning algorithms can be:

- Value Based:

  Here, value based agents decide the goodness of an action performed at a given state and that model is used to behave optimally. Value based agents do not learn the policy, instead they focus on how good a state is and depending on that, select a policy. This method is adopted only in the case of deterministic policies.

- Policy Based:
  The policy function, that maps states and actions, is learnt. The action is learnt without focusing on the value function. This method can be used in stochastic policies as well.

**3.2 Discrete and continuous action spaces**

There are certain areas where reinforcement learning algorithms have given astonishing results. Computer games, Go and Chess are some of the problems solved effectively using Deep Q Network (DQN) algorithm [20], [21] and [22]. DQN is useful to solve problems having discrete action spaces and low dimensions. The problem of attitude control is high dimensional and works in a continuous action space. Lillicrap et al. in [23] gave Deep Deterministic Policy Gradients (DDPG), a landmark algorithm applying the concept of DQN on continuous control problems.

**3.3 Policy gradient**

Policy gradient approach is used in on-policy problems. The policy space is optimized so that actions that give higher rewards have a better probability of being selected and actions that lead to low rewards are rarely performed. The policy network takes the environment's space as the input and outputs an estimate of the different actions' probabilities. $J(\theta)$, the expected reward, is the product of the probability of trajectory and the corresponding reward.

$$J(\theta) = \sum_{\tau} \mathcal{P}\,(\tau;\theta)R(\tau)$$

where θ denotes the policy employed to create trajectory τ, and

$$\tau = (s_1, a_1, s_2, a_2, \dots, s_t, a_t)$$

Now, a value of θ for maximizing the reward.

$$\max_{\theta} J(\theta) = \max_{\theta} \sum \tau\, \mathcal{P}(\tau;\theta)R(\tau)$$

Since policy gradient approaches directly learn the policy, they work in complex environments where DQN approaches fail. They converge faster and can learn stochastic policies, a place where value based approaches fail. They are also effective in continuous action spaces.

However, they are very unstable and not sample efficient. In case rewards are delayed, their credit assignment is also poor. As newer policy gradient techniques are proposed and researched, we have to avoid actions that lead to less optimal results as one wrong decision affects the remaining process of optimization.

**3.4 Role of IMU and ESC**

For the implementation of EEAC on PPO and DDPG, we have relied on Open AI Gym environment which provides the set of environments for testing reinforcement learning algorithms. We have used GymFC, an Open AI Gym based flight controller environment. As is the standard, it has an action space and an observation space. The action space consists of the reinforcement learning algorithm's instructions to the quadcopter while the observation space is the output of the quadcopter which is received by the agent.

A component called Gyroscope within the Inertial Measurement Unit (IMU) records and sends the angular velocity $\sigma(t)$ of the quadcopter at each step. $\sigma(t) = [\sigma_{\phi}(t), \sigma_{\theta}(t), \sigma_{\psi}(t)]$ where $\sigma_{\phi}(t), \sigma_{\theta}(t)$ *and* $\sigma_{\psi}(t)$ are the angular velocities of each of the roll, pitch and yaw axes. The Electronic Speed Controllers (ESCs) record the speed of rotations of the four motors $\mu(t) = [\mu_0(t), \mu_1(t), \mu_2(t), \mu_3(t)]$
where $\mu_0(t), \mu_1(t), \mu_2(t)$ and $\mu_3(t)$ are individual motor speeds measured in Revolutions per Minute (RPMs). When the agent receives these values, it applies the reinforcement learning algorithm to come up with a response which consists of Pulse Width Modulation (PWM) signals $a(t) = [a_0(t), a_1(t), a_2(t), a_3(t)]$. These instructions are sent to the ESCs.

## 3.5 Neural Networks

The function of the agent where it takes a state as input and provides an action as the output can be represented in the form of a table where the x axis represents possible actions and y axis denotes the set of states. The cells of the table would denote the received reward. The reinforcement learning algorithm updates this table at each step while following a policy. However, multi-dimensional problems in continuous spaces cannot be represented in a table of viable size as the number of possibilities are many. Hence, the need of deep reinforcement learning. The deep part is constituted by the neural network consisting of an input and output layer along with multiple hidden layers. The purpose of the neural network is to sum together all the information that has been accessed by the agent in the past and use it to come up with information that is useful for the task to be done. The agent gets trained in this process to act in a real world scenario.

## 3.6 Layers and activation function

Each of the layers of the neural network has nodes called neurons and it is the place where the computation takes place (Fig. 3). The neurons are such called because they mimic the behaviour of the neurons of the human brain which perform an action when triggered by a stimulus of sufficient proportion.

A neuron takes the product of the input *$x_i$* and the weight *$w_i$*. The weight magnifies or reduces the input, depending on the amount of significance each input needs to be given. The sum of all such products is then sent through an activation function $f(\sum_{i=1}^{n} x_i\, w_i)$, which decides whether to allow or disallow a signal to pass through. Commonly used activation functions are Tanh, ReLU, Sigmoid, etc.

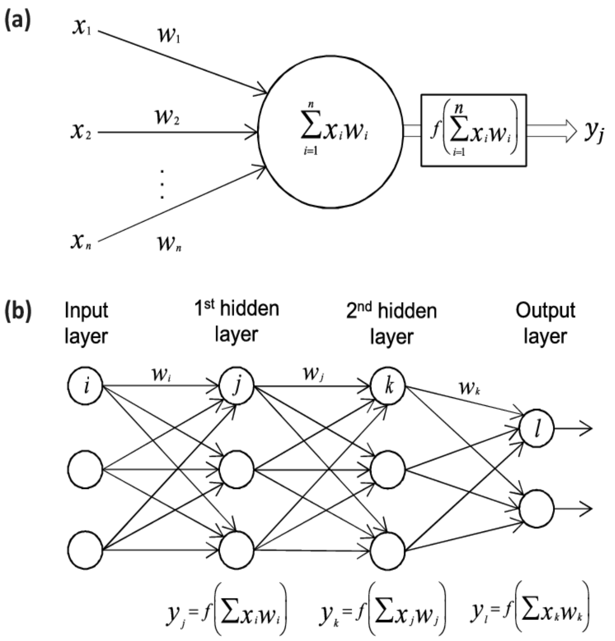

Fig. 3 (a) A neuron 3 (b) A neural network having multiple layers [24]}

## 3.7 Deep Deterministic Policy Gradients (DDPG)

Q learning or DQN cannot handle continuous action spaces. It outputs only a discrete number of actions. However, we need continuous action spaces as the PWM commands have to be sent by the agent to the ESCs. It is inappropriate to discretize the continuous action spaces as the six degrees of freedom of the quadcopter would lead to too many discrete action spaces.
However, innovations to Q learning can be applied to the actor critics methods. In DDPG [23], we use

Replay Memory

Instead of learning just from the current state the agent is experiencing, it keeps track of the total of all experiences and randomly samples some batch of memories to update the weights of the deep neural networks.

Target Network

Use one network to determine the actions to take and then use another network to determine the value of that action. That value is used to update the weights of the deep neural networks. When we use the same network to do both the things, we are chasing a rapidly changing target because the weights are getting updated. So, the valuation of similar states changes rapidly over the course of simulation, causing the learning to be unstable. So, we use two networks - one to choose actions at each timestep and the other to evaluate the effectiveness of those actions while updating the weights.

DDPG is off policy, i.e. we use a separate policy to get the data and use the data to update a different policy, performing a soft copy of the target networks. We have two target networks in DDPG because DDPG is an actor critic method. So there are two distinct networks, one for the actor and one for the critic. So, overall we will have four networks - one actor, one critic, one target actor and one target critic. The critic network evaluates state action pairs. The actor network decides what to do based on the current state. The actor network outputs action values and not probabilities. This is because DDPG is deterministic, hence the algorithm leads to the same state over and over again and we get the same action from the agent. However, the problem with this approach is that the agent has an explore-exploit dilemma. It aims to understand the environment by exploration or try to follow a chosen path by exploiting the achieved knowledge. This is implemented in DDPG by including noise into the system.

The update rule for the actor is denoted by the equation [23] below.

$$\nabla_{\theta^\mu} J \approx E_{s_t \sim \rho^\beta}\left[\nabla_{\theta^\mu} Q(s, a|\theta^Q)|_{s=s_t, a=\mu(s_t|\theta^\mu)}\right]$$

The above equation shows the expectation value or the average of the gradient of the critic network, where we input some states and actions and the action is chosen based on the current policy. So, we sample states randomly from the memory and then the actor network decides what actions to take based on those states. Then the actions and states from the actor are plugged into the critic to take its reaction on the quality of the actions. We, then take the gradient of the critic network with respect to the parameters of the actor network Updating the critic network by minimizing the loss

$$L = \frac{1}{N}\sum_i \left(y_i - Q(s_i, a_i|\theta^Q)\right)^2$$

$$\nabla_{\theta^\mu} J \approx \frac{1}{N}\sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s_i}$$

The loss function is a mean squared error of the difference between target value $y_i$ and the Q for the current state and action. We randomly sample states, new states, actions and rewards. We use the target actor to determine actions for the new states. The actions are plugged into the target critic to get the target $y$ and multiply it with the discount factor $\gamma$ and add a reward of that timestep. We then plug the states and actions into the critic and take the difference with the target.

In order to handle the updates of the target networks, the actor and critic networks are initialized with some random parameters and the same values are copied to the target actor and critic networks also. This is the only instance of a hard copy. The rest of the time, a soft copy is done to update the target.

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$

$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$

### 3.8 Proximal Policy Optimization (PPO)

PPO is an on-policy technique, so the agent learns directly from its interactions with the environment. It is also based on policy gradient. Just like DDPG, PPO also uses the actor critic approach. Here, the actor network is policy based while the critic is value based.

In PPO [25], policy gradient loss is defined as the average(expectation) of the log of policy multiplied by the advantage function.

$$L^{PG}(\theta) = \widehat{E_t}\left[\log \pi_\theta\,(a_t|s_t)\widehat{A_t}\right]$$

The advantage function is a measure of the goodness of an action. It compares the current action with the average action. The discounted set of rewards provide the baseline estimate for the advantage function.

$$\widehat{A_t} = \delta_t + (\gamma\lambda)\delta_{t+1} + \cdots + (\gamma\lambda)^{T-t+1}\delta_{T-1}$$

$$where\ \delta_t = r_t + \gamma V(s_{t+1}) - V(s_t)$$

A positive advantage function increases the probability of performing a given action when a particular state is observed. The problem with this approach is that if we keep on running gradient descent on a particular batch of states, we might go out of range and get practically incorrect results. So, the aim is to make small changes. Trust Region Policy Optimization (TRPO) [26], a predecessor to PPO, added KL constraint (Kullback-Leibler). PPO improves this technique by adding a constraint to the optimization objective directly.

$$L^{CLIP}(\theta) = \widehat{E_t}\left[\min\left(r_t(\theta)\widehat{A_t}, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\widehat{A_t}\right)\right]$$

In PPO, a value function error is added to the policy function. In order to promote exploration, an entropy bonus has been added.

$$L_t^{CLIP+VF+S}(\theta) = \widehat{E_t}[L_t^{CLIP}(\theta) - c_1 L_t^{VF}(\theta) + c_2 S[\pi_\theta](s_t)]$$

Thus, PPO manages to improve the performance achieved by reducing the policy update values to a given range. Its simple objective function further reduces performance overhead.

## 4 IMPLEMENTATION OF ENERGY EFFICIENT ATTITUDE CONTROL

In this section, we implement our Energy Efficient Attitude Control (EEAC) algorithm [27] on two deep reinforcement learning algorithms DDPG and PPO and compare the results of both the approaches. We added Ornstein Uhlenbeck noise [28] to the action space. Noise facilitates better exploration by adding uncertainty.

$$\pi'(s_t) = \pi(s_t|\theta_t^\pi) + \mathcal{N}$$

$\pi'(s_t)$denotes the exploration policy obtained when noise $\mathcal{N}$is added to the policy$\pi(s_t|\theta_t^{\pi})$. To facilitate better exploration, temporally correlated noise has been used. Parameters used are $\theta = 0.15$ and $\sigma = 0.3$ and have been kept the same in both DDPG and PPO. While working on the attitude control problem, we identified the key areas where energy was used the most during attitude control. The only moving parts of the quadcopter were the propellers and the motors driving them. Thus, if we had to reduce energy depletion, we would need to avoid running the motors at high speed. Also, the variations in speeds of motors needed to be minimized. Focussing on the energy aspect, we had written EEAC algorithm and now we expand the scope of the algorithm by comparing its performance on two of the landmark deep reinforcement learning algorithms - DDPG and PPO.

### 4.1 Proposed algorithm for Energy Efficient Attitude Control

In EEAC, we train the system to achieve a target angular velocity $\sigma^*$ (where $\sigma^*$=$\sigma_r^*, \sigma_p^*, \sigma_y^*$ corresponding to the roll, pitch and yaw axes) while reducing energy consumed in the process. At the start of each episode, this angular velocity is initialized and the algorithm aims to achieve this target value while maximizing the reward (or, minimize a negative reward). The algorithm runs a loop in ordered to minimize the negative reward so that reward $r_t \to 0$. Inside the loop, action $a_t$ is initialized with the output of the reinforcement algorithm used, with the Pulse Width Modulation values $y_i$ (where $i \to$ 0 to 3) for all the four motors. Thereafter, noise $\mathcal{N}$is added to it. This action value is sent to the environment and the state $s_{t+1}$ received from it. This state value consists of the present angular velocity and is a three-member value for each of the roll, pitch, yaw axes and the present RPM values of the four motors. The difference of the motor speed values obtained from the environment from two epochs is calculated and the average of this difference stored in $\Delta\mu$.Thereafter, the angular velocities received in $s_{t+1}$ are taken and subtracted from the target velocity values. The difference values of each of the three axes are squared, summed and the square root of the same stored in $\Delta\sigma$. The difference between the RPM values of the motors and the value of angular velocity is treated as a penalty that needs to be minimized. The algorithm multiplies a constant $\alpha$ to the motor velocity difference $\Delta\mu$ and then add
$\Delta\sigma$, the angular velocity difference. To show that it is a penalty, the algorithm assigns a negative sign to it, with the intention of minimizing it. As a last step, the current values are updated on the previous copy of RPM values for the next epoch.

```
Algorithm 1: Energy efficient attitude control (EEAC)
for episode:=1,....,N do
    Initialize $a_t$, $\sigma$, $\mu$;
    Initialize the environment and receive initial state $s_1$;
    Initialize $\sigma^*$={ $\sigma_r^*, \sigma_p^*, \sigma_y^*$};
    for timestep t:=1,....,T do
        Generate an action $a_t = (\mu_0, \mu_1, \mu_2, \mu_3)$
        $a_t \leftarrow a_t + \mathcal{N}$where $\mathcal{N}$ is Ornstein Uhlenbeck noise
        Send action $a_t$to the environment and receive states$s_{t+1}$
        $s_{t+1} = \{(\sigma_r, \sigma_p, \sigma_y), (\mu'_0, \mu'_1, \mu'_2, \mu'_3)\}$
        $\sigma = (\sigma_r, \sigma_p, \sigma_y)$
        $\mu' = (\mu'_0, \mu'_1, \mu'_2, \mu'_3)$
        $\delta\mu = (|\mu'_0 - \mu_0|, |\mu'_1 - \mu_1|, |\mu'_2 - \mu_2|, |\mu'_3 - \mu_3|)$
        $\Delta\mu = \frac{1}{4}\sum_{n=0}^{3} \delta\mu_n$
        $\delta\sigma = \left((\sigma_r^* - \sigma_r)^2, (\sigma_p^* - \sigma_p)^2, (\sigma_y^* - \sigma_y)^2\right)$
        $\Delta\sigma = \frac{1}{3}\sum_{i=0}^{2} \delta\sigma_i$
        $\Delta\sigma \leftarrow \sqrt{\Delta\sigma}$
        $r_t = -(\alpha * \Delta\mu + \Delta\sigma)$
        Set$(\mu_0, \mu_1, \mu_2, \mu_3) \leftarrow (\mu'_0, \mu'_1, \mu'_2, \mu'_3)$
        return $r_t$
    end
end
```

## 5 EXPERIMENTAL EVALUATION

We have simulated our proposed algorithm on two reinforcement learning algorithms to compare the performance of the two algos on the attitude control problem. In the coming sections, we discuss the simulation settings and the results obtained.

### 5.1 Simulation Settings

Open AI Gym is a set of reinforcement learning environments that can be used to test different deep reinforcement learning algorithms. We have deployed GymFC, a flight controller environment built using Open AI Gym. GymFC requires Ubuntu 18.04. The flight simulator required for GymFC is Gazebo and we use Gazebo v10.1.0. The model of the quadcopter is created in a *.sdf* file which the environment accesses. We use Dynamic Animation and Robotics Toolkit (DART) version 6.7.0 which is a physics engine. DART provides Gazebo the data structures and algorithms for calculating the motion dynamics. Open AI Gym has the reinforcement learning algorithms' baselines. We have forked DDPG and PPO from the GitHub page of Open AI Gym to test our algorithm.

#### 5.1.1 PPO settings

**Table 1.** Hyperparameters deployed by the PPO Agent

| Hyperparameter | Value |
|---|---|
| Horizon(T) | 500 |
| Adam Stepsize | $1 \times 10^{-4} \times \rho$ |
| Num. Epochs | 5 |
| Minibatch Size | 32 |
| Discount Factor ( $\gamma$) | 0.99 |

ITEE Journal
Information Technology & Electrical Engineering

ISSN: - 2306-708X

| | |
|---|---|
| GAE Parameter | 0.95 |

Notes: Value of $\rho$ reduces gradually from 1 to 0 during training

The agent consists of a neural network where the input layer has six nodes, the output layer has four and there are two hidden layers each having 32 nodes. For a problem of continuous domain like ours, the neural network's output is a Gaussian distribution mean and varying standard deviation (as per [25]). We have stuck to the standard hyperparameters defined in the paper, as shown in the below table 1.

### 5.1.2 DDPG settings

For DDPG, the first hidden layer has 400 nodes and the second hidden layer has 300 nodes. We use ReLU activation function for the hidden layers and Tanh for the outer layer. The weights and biases of the final layer of the actor and critic have been initialized in the range $-3 \times 10^{-3}$ to $3 \times 10^{-3}$.

**Table 2.** Hyperparameters used by the DDPG Agent

| Hyperparameter | Value |
|---|---|
| Adam Stepsize (actor) | $1 * 10^{-4}$ |
| Adam Stepsize (critic) | $1 * 10^{-3}$ |
| Size of Minibatch | 64 |
| L2 weight decay for Q(critic) | $10^{-2}$ |
| Smoothing constant ($\tau$) | 0.001 |
| Size of replay buffer | $10^{6}$ |
| Discount Factor ($\gamma$) | 0.99 |

### 5.1.3 Other settings and evaluation metrics

For Ornstein Uhlenbeck Noise [28], we have taken the following values: $\mu=0$, $\sigma=0.3$ and $\theta=0.15$. The training of the model for the two reinforcement learning algorithms was performed for 10 million steps each. The simulation was performed on a four core i5-8250U processor system running Ubuntu 18.04 operating system. Our code creates checkpoints every 100,000 steps. Thus, altogether, we created 100 checkpoints for each of the runs and used these checkpoints to measure the performances of the PPO algorithm against the DDPG algorithm. In order to create graphs, we used Tensor Flow 1.14.

The evaluation and comparison of DDPG and PPO using our EEAC algorithm to test attitude control of quadcopters was done on the following four parameters:

*Reward*

The reward function, the heart of any reinforcement learning algorithm, is the change in quadcopter motor speeds and the difference in angular velocity values. We compare DDPG and PPO using the reward function and evaluating on the following metrics:

- Mean Absolute Error (MAE) – It is the difference between the current and target angular velocities.
- Average Motor Velocities (RPM values) - The mean value of the quadcopter motor velocities.
- Average Change in Motor Velocities - Mean difference between the motor velocity of the previous and current time step.
- Average Reward - The output given by EEAC at the end of each episode.

*Energy efficiency*

We calculate energy using the formula [29]:

$$EnergyConsumption(E) = C_P \rho n^3 d^5$$

where $C_P$(Energy consumption co-efficient) = $2\pi C_Q$ and $C_Q$ represents torque co-efficient and $C_Q = 1.38 \times 10^{-3}$ in GymFC.$\rho$(Atmospheric density) = $1.275 kg/m^3$ at zero celsius temperature and at sea level [30][31],
n represents propeller speed, d (Propeller diameter) =$10 cms$
(This simulation uses PX4 Gazebo simulation plugins whose diameter$(d) = 10cms$)

### 5.2 Results and analysis

We describe the results obtained through graphs. The Mean Absolute Error (MEA) (shown in Fig. 4 (a)) is the angular velocity difference.

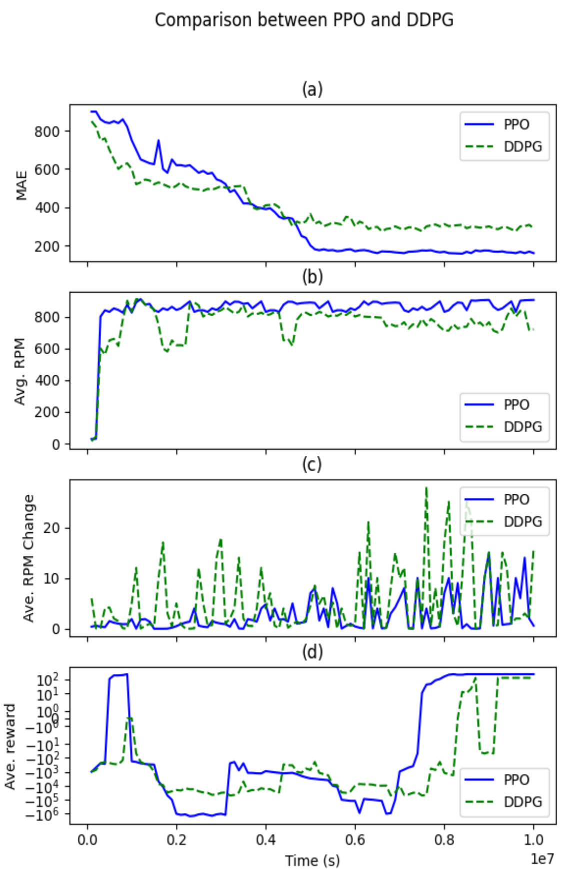

Fig. 4 Reward comparison between PPO and DDPG

MEA values at the start of the simulation are more for PPO than for DDPG. Till about 4 million timesteps, the MAE values are more for PPO, thereafter the values are less and continue so till the end of our simulation.

In Fig. 4(b), the average velocity of the four motors is compared for both the algorithms. The values remain similar till around 4 million time steps, after which the average motor velocities for DDPG are less as compared to PPO. So, in this metric, DDPG scores better.

Figure 4(c) displays the graph of the average changes in RPM values of the motors by taking the average value of the four motors and then finding the change in the current average and the previous average. The graph shows frequent changes in average RPM values for DDPG. The changes are much less in PPO. As the algorithms start converging, the spurts in graph become more pronounced.

In Fig. 4(d), we display the average reward values at each timestep. The y-axis is represented as a log scale. During the start of the simulation, as MEA values of both the algorithms reduce, the reward values increase. As the algorithms try reducing energy consumption, the reward values go negative. PPO algorithm starts converging sooner than DDPG. We observe from the graph that around 7.5 million steps, PPO algorithm starts converging. DDPG starts converging later at around 9 million steps. Being off policy, we expected that DDPG would start converging later and our results proved this.

A comparison of the energy consumed by both the algorithms has been shown in Fig. 5.

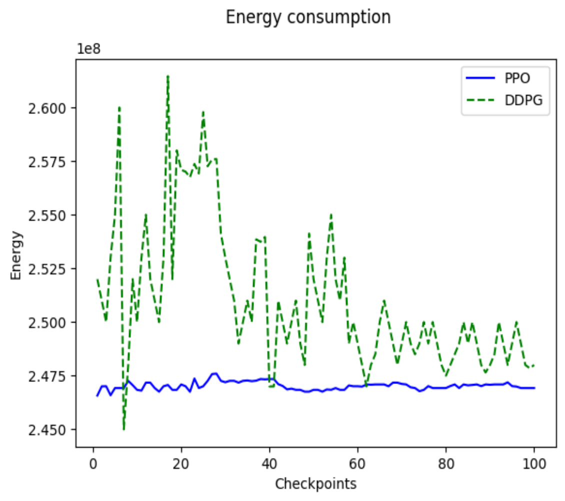

Fig. 5 Energy efficiency comparison between PPO and DDPG

During the training process, we had created 100 checkpoints and the graph plots the energy consumption values at each of these checkpoints. As can be observed from the graph, the energy consumed by PPO remains almost constant throughout the run. The graph shows energy consumption in DDPG to be more than that obtained in PPO. The changes in energy consumption values across checkpoints is also highly varied for DDPG and is much lesser in PPO. Energy consumption values are around 2.475e8 in PPO and are in the range of 2.475e8 and 2.575e8 in DDPG. Looking at the graph, we can safely claim PPO to give better results than DDPG in the attitude control problem.

## 6 CONCLUSION

In this paper, we have attempted handling the attitude control problem in quadcopters. Attitude control consumes a lot of energy and we have used reinforcement learning algorithms to minimize this consumption. We had created an EEAC algorithm and we used that algorithm in this paper to test the performance of PPO and DDPG, two of the state of the art reinforcement learning algorithms. We compared mean absolute error, change in motor velocities and energy consumption. We found PPO performing better than DDPG in the above metrics. The on-policy technique adopted by PPO was the major cause that tilted the results in its favour. Also, the clipping function, a key differentiator between PPO and the other reinforcement learning algorithms was also critical in giving us an output better than the one given by DDPG.

We feel that this research work can further be used to test the navigational control problem. Navigation control is another field where quadcopter energy depletes fast. We would be keen to test the performances of PPO and DDPG to handle navigation control. If we can come up with an algorithm just like EEAC that integrates navigation control and attitude control, we can have a complete reinforcement learning solution for quadcopter movements.

While simulating our work, we had disabled the feature of gravity on Gazebo, the simulator we had used. Incorporating gravity presents a whole lot of issues that can be explored independently or in sync with the attitude control and navigation control problems.

## REFERENCES

[1] H. Bou-Ammar, H. Voos and W. Ertel, "Controller design for quadrotor UAVs using reinforcement learning," *2010 IEEE International Conference on Control Applications*, Yokohama, 2010, pp. 2130-2135, doi: 10.1109/CCA.2010.5611206

[2] R. S. Sutton and A. G. Barto, Reinforcement learning: an introduction. Cambridge (Mass.): The MIT Press., 2018.

[3] J. Jiang and M. S. Kamel, "Pitch Control of an Aircraft with Aggregated Reinforcement Learning Algorithms," *2007 International Joint Conference on Neural Networks*, Orlando, FL, 2007, pp. 41-46, doi: 10.1109/IJCNN.2007.4370928.

[4] K. Alexis, G. Nikolakopoulos and A. Tzes, "Constrained optimal attitude control of a quadrotor

helicopter subject to wind-gusts: Experimental studies," *Proceedings of the 2010 American Control Conference*, Baltimore, MD, 2010, pp. 4451-4455, doi: 10.1109/ACC.2010.5531005.

[5] J. Hwangbo, I. Sa, R. Siegwart and M. Hutter, "Control of a Quadrotor With Reinforcement Learning," in *IEEE Robotics and Automation Letters*, vol. 2, no. 4, pp. 2096-2103, Oct. 2017, doi: 10.1109/LRA.2017.2720851.

[6] N. O. Lambert, D. S. Drew, J. Yaconelli, S. Levine, R. Calandra and K. S. J. Pister, "Low-Level Control of a Quadrotor With Deep Model-Based Reinforcement Learning," in *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 4224-4230, Oct. 2019, doi: 10.1109/LRA.2019.2930489.

[7] P. Abbeel, M. Quigley, & A. Y. Ng. 2006. Using inaccurate models in reinforcement learning. In Proceedings of the 23rd international conference on Machine learning (ICML '06). Association for Computing Machinery, New York, NY, USA, 1–8. DOI:https://doi.org/10.1145/1143844.1143845

[8] S. R. B. dos Santos, C. L. Nascimento and S. N. Givigi, "Design of attitude and path tracking controllers for quad-rotor robots using reinforcement learning," *2012 IEEE Aerospace Conference*, Big Sky, MT, 2012, pp. 1-16, doi: 10.1109/AERO.2012.6187314.

[9] W. Lou and X. Guo, "Adaptive Trajectory Tracking Control using Reinforcement Learning for Quadrotor", International Journal of Advanced Robotic Systems, vol. 13, no. 1, p. 38, 2016. Available: 10.5772/62128 [Accessed 25 January 2021].

[10] N. Imanberdiyev, C. Fu, E. Kayacan and I. Chen, "Autonomous navigation of UAV by using real-time model-based reinforcement learning," 2016 14th International Conference on Control, Automation, Robotics and Vision (ICARCV), Phuket, 2016, pp. 1-6, doi: 10.1109/ICARCV.2016.7838739.

[11] A. Rodriguez-Ramos, C. Sampedro, H. Bavle, I. G. Moreno and P. Campoy, "A Deep Reinforcement Learning Technique for Vision-Based Autonomous Multirotor Landing on a Moving Platform," 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Madrid, 2018, pp. 1010-1017, doi: 10.1109/IROS.2018.8594472.

[12] S. Li, P. Durdevic, and Z. Yang, "Optimal Tracking Control Based on Integral Reinforcement Learning for An Underactuated Drone", IFAC-PapersOnLine, 52(8), 2019, pp. 55-60, doi.org/10.1016/j.ifacol.2019.08.048.

[13] C. Nie, Z. Zheng, and M. Zhu, "Three-Dimensional Path-Following Control of a Robotic Airship with Reinforcement Learning," International Journal of Aerospace Engineering, 25-Mar-2019. [Online]. Available: https://www.hindawi.com/journals/ijae/2019/7854173/abs/. [Accessed: 25-Jan-2021].

[14] S.-Y. Shin, Y.-W. Kang, and Y.-G. Kim, "Obstacle Avoidance Drone by Deep Reinforcement Learning and Its Racing with Human Pilot," Applied Sciences, vol. 9, no. 24, p. 5571, 2019.

[15] T. Koning, "Low level quadcopter control using Reinforcement Learning: Developing a self-learning drone," TU Delft Repositories, 01-Jan-1970. [Online]. Available: https://repository.tudelft.nl/islandora/object/uuid:0b9e0796-13b5-42ba-b231-fbb6aadd5233. [Accessed: 25-Jan-2021].

[16] C. Bekar, B. Yuksek, G. Inalhan, "High Fidelity Progressive Reinforcement Learning for Agile Maneuvering UAVs, AIAA SciTech Forum, 05-Jan-2020. [Online]. Available: https://arc.aiaa.org/doi/10.2514/6.2020-0898. [Accessed: 25-Jan-2021].

[17] E. Bohn, E. M. Coates, S. Moe, and T. A. Johansen, "Deep Reinforcement Learning Attitude Control of Fixed-Wing UAVs Using Proximal Policy optimization," 2019 International Conference on Unmanned Aircraft Systems (ICUAS), 2019.

[18] W. Zhou, K. Yin, R. Wang, and Y.-E. Wang, "Design of Attitude Control System for UAV Based on Feedback Linearization and Adaptive Control," Mathematical Problems in Engineering, vol. 2014, pp. 1–8, 2014.

[19] R. Amiri, H. Mehrpouyan, L. Fridman, R. K. Mallik, A. Nallanathan, and D. Matolak, "A Machine Learning Approach for Power Allocation in HetNets Considering QoS," 2018 IEEE International Conference on Communications (ICC), 2018.

[20] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, and S. Petersen, "Human-level control through deep reinforcement learning. Nature", 518(7540), 529-533, 2014. http://dx.doi.org/10.1038/nature14236

[21] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller,

"Playing Atari with Deep Reinforcement Learning," arXiv.org, 19-Dec-2013. [Online]. Available: https://arxiv.org/abs/1312.5602. [Accessed: 25-Jan-2021].

[22] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. V. D. Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of Go with deep neural networks and tree search," Nature, vol. 529, no. 7587, pp. 484–489, 2016.

[23] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," arXiv.org, 05-Jul-2019. [Online]. Available:https://arxiv.org/abs/1509.02971. [Accessed: 25-Jan-2021].

[24] S. Vieira, W. H. Pinaya, and A. Mechelli, "Using deep learning to investigate the neuroimaging correlates of psychiatric and neurological disorders: Methods and applications," Neuroscience & Biobehavioral Reviews, vol. 74, pp. 58–75, 2017.

[25] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal Policy Optimization Algorithms," arXiv.org, 28-Aug-2017. [Online]. Available: https://arxiv.org/abs/1707.06347. [Accessed: 25-Jan-2021].

[26] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust Region Policy Optimization," PMLR, 01-Jun-2015. [Online]. Available: http://proceedings.mlr.press/v37/schulman15. [Accessed: 25-Jan-2021].

[27] V. Agarwal, R. R. Tewari, "Improving Energy Efficiency in UAV Attitude Control using Deep Reinforcement Learning", 2020. Manuscript submitted for publication.

[28] G. E. Uhlenbeck and L. S. Ornstein, "On the Theory of the Brownian Motion," Physical Review Journals Archive, 01-Sep-1930. [Online]. Available: https://journals.aps.org/pr/abstract/10.1103/PhysRev.36.823. [Accessed: 25-Jan-2021].

[29] A. Oosedo et al., "Design and simulation of a quad rotor tail-sitter unmanned aerial vehicle," 2010 IEEE/SICE International Symposium on System Integration, Sendai, 2010, pp. 254-259, doi: 10.1109/SII.2010.5708334.

[30] "US5904323A - Constrained store release system," Google Patents. [Online]. Available: https://patents.google.com/patent/US5904323A/en. [Accessed: 25-Jan-2021].

[31] F. E. Kidder and T. Nolan, The architects and builders hand-book. New York: John Wiley & Sons, Inc., 1921.

## AUTHOR PROFILES

**Varun Agarwal** has completed his B. Tech and M. Tech in Computer Science, has worked with Infosys Technologies Ltd. and is currently pursuing his PhD from University of Allahabad, Allahabad, India.

**Prof. R. R. Tewari** has been the former Head of Department, Department of Electronics and Communication, University of Allahabad. He has also been the former Dean Faculty of Science and former Vice Chancellor of the University of Allahabad. He has numerous journals and books to his name. He has supervised many PhD students and been the principal investigator in a lot of government sponsored research projects.