

RESEARCH

Open Access



# Implementation of partially tuned PD controllers of a multirotor UAV using deep deterministic policy gradient

Emmanuel Mosweu<sup>1</sup>, Tshepo Botho Seokolo<sup>1</sup>, Theddeus Tochukwu Akano<sup>1\*</sup>  and Oboetswe Seraga Motsamai<sup>1</sup>

\*Correspondence:  
akanott@ub.ac.bw;  
manthez2016@gmail.com

<sup>1</sup> Department of Mechanical Engineering, University of Botswana, Gaborone, Botswana

## Abstract

The present methodology employed in classical control systems is characterized by high costs, significant processing requirements, and inflexibility. In conventional practice, when the controller exhibits instability after being implemented on the hardware, it is often adjusted to achieve stability. However, this approach is not suitable for mass-produced systems like drones, which possess diverse manufacturing tolerances and delicate stability thresholds. The aim of this study is to design and evaluate a controller for a multirotor unmanned aerial vehicle (UAV) system that is capable of adapting its gains in accordance with changes in the system dynamics. The controller utilized in this research employs a Simulink-constructed model that has been taught by reinforcement learning techniques, specifically employing a deep deterministic policy gradient (DDPG) network. The Simulink model of the UAV establishes the framework within which the agent engages in learning through interaction with its surroundings. The DDPG algorithm is an off-policy reinforcement learning technique that operates in continuous action spaces and does not require a model. The efficacy of the cascaded PD controllers and neural network tuner is evaluated. The results revealed that the controller exhibited stability during several flight phases, including take-off, hovering, path tracking, and landing manoeuvres.

**Keywords:** UAV, Octorotor, DDPG, ANN

## Introduction

The human capacity to generate imaginative and innovative solutions necessary for overcoming challenges has been a valuable advantage [1]. The complexity of the human brain compared to other creatures can be regarded as a significant advantage. With the advent of modern technology and an increasingly technologically driven world, there has been growing interest among scientists in transferring some degree of intelligence to machines [2]. Artificial intelligence (AI) is a discipline in science and engineering that has emerged over the past few decades and has since become pervasive in every domain. Machine learning (ML) is a subset of these intelligent systems that have the ability to enhance their performance with experience [3]. Machine learning can be seen as a

computational counterpart to human learning and has applications in a wide range of fields, including computer vision, natural language processing, robotics, and finance [4]. This study was focused on the utilization of machine learning methodologies in robots, particularly in the management of unmanned aerial vehicles (UAVs).

The octocopter is an UAV that can hover and navigate by controlling the eight lifts without the need for the complicated system of linkages and blade elements found in conventional single-rotor vehicles. Because of its adaptability, this system can accomplish vertical take-off and landing (VTOL) and any other flight manoeuvre with ease [5]. Like most design processes in engineering, the flight controller of such a system must be built and tested on software before deploying it to the hardware [6]. To achieve this, a model of the octocopter, developed using a system of equations that represent its dynamics, is required. This model will be used to simulate flight and demonstrate that the control system functions as theorized after installation on the octocopter [7].

In the past decade, a rise of techniques that apply the intelligent approach to increase the level of autonomy within UAVs have quickly emerged [8]. Researchers such as Costa et al. [9] have adopted some degree of truth in the landing of a UAV, an approach that has been made possible using mathematical models based on fuzzy logic, to achieving relatively accurate results. To achieve autonomous navigation with the limit of a closed environment, Padhy et al. [10] used a deep neural network (DNN) to filter an RGB image provided by a camera attached to the UAV. This allows it to navigate through the environment in a controlled manner. In Villanueva and Farjardo [11], the objective was to improve UAV performance using the deep Q-network (DQN) through noise injection. The authors reported promising results in terms of improved performance and reduced control effort using this approach.

In other related works, Cano et al. [12] utilized the proximal policy optimization (PPO) algorithm in combination with a stochastic policy gradient to train a quadrotor and develop a dependable control policy. By using a model-free reinforcement learning (RL) approach, they demonstrated the feasibility and benefits of training a UAV's low-level controller without human intervention. This was applied by Schulman et al. [13] to UAVs algorithm implementation. Cardenas et al. [14] created the first open-source NN-based flight controller firmware, which allows a neural network to be trained in simulation and compiled to run on hardware. The primary goal was to enhance the altitude controller of the UAV, which is traditionally achieved using a PID controller [15]. Additionally, alternative approaches have been developed for applications such as combat and reconnaissance missions.

Maciel-Pearson et al. [16] utilized deep learning (DL) and multi-task regression-based learning (MTRL) to develop a strategy for navigating through forests, even in the presence of trails and global positioning system (GPS). This strategy employed two sub-networks, each featuring a convolutional layer. Meanwhile, Xu et al. [17] focused on enhancing the UAV's decision-making autonomy for military applications. They employed a combination of deep belief network (DBN) with Q-learning and a decision-making algorithm based on genetic algorithms (GAs) to enhance the UAV's autonomy in decision-making for military applications. Their approach yielded satisfactory results. Other researchers in a similar field of study used surveillance images and a convolutional neural network (CNN) architecture to

learn human interactions and identify how the opponent’s aircraft is being controlled [18]. Bouhamed et al. [19] utilized the deep deterministic policy gradient (DDPG) approach, which has a continuous action-space, to train a UAV to navigate through or over obstacles in order to reach a target. The DDPG algorithm is an extension of the deep Q-network (DQN) [20] and combines the actor-critic approach with insights from DQN. The reward function was designed to guide the UAV through the optimal path while penalizing any collisions. The architectures reported in the literature review highlight the need for a hybrid system for control engineers, as many of these approaches are more oriented towards computer science than engineering.

According to the investigation conducted by Zulu and John [21] on the analysis of multiple control algorithms for autonomous quadrotors, it was found that no single controller provided all the desired features, such as adaptability, robustness, optimal trajectory tracking, and disturbance rejection. Instead, they found that hybrid control schemes combine multiple controllers and produced the best results. The nonlinear systems considered in their study were not effectively tuned using conventional offline PID tuning techniques [22, 23], and online tuning of PID parameters was required to achieve better performance for these systems. Several studies have explored the use of various algorithms, such as fuzzy logic and artificial neural networks (ANNs) for continuous PID gain tuning [24, 25]. ANNs have been found to have an advantage over other methods due to their ability to solve nontrivial problems [25]. Hernandez-Alvarado et al. [26] also discussed the use of online tuning of PIDs using ANNs for control of submarines. For a quadcopter, Yoon et al. [27] developed a robust control technique based on an ANN that could adjust the PID parameters in real-time to handle wind disturbance. A comparison chart for the rise time, settling time, and overshoot for pitch, roll, and yaw using the RL and PID controllers is shown in Table 1.

The classical control systems approach to solving nonlinear systems is financially, computationally, and timely expensive, and can be quite rigid in solving dynamic problems. Due to the slight changes in their physical properties, the moments of inertia of the system with respect to the three axes are also continuously changing, and therefore, the actuator effort needed archive stability at different points of the flight also varies. For example, mass changing systems that vary by over 50% behave with a completely different set of dynamics at the two extremes. Therefore, system stability cannot be archived for both using a static controller. These controllers face difficulty due to their inability to detect and respond to changes in the system’s physical attributes. This begs the question of how one designs a controller that can take in system attributes and adjust its gains in response to the changing system dynamics. Most common methods of solving this problem involve engineers analysing all possible system states and designing multiple controllers for each. This implies solving a single

**Table 1** Comparison of an RL and PID controller [28]

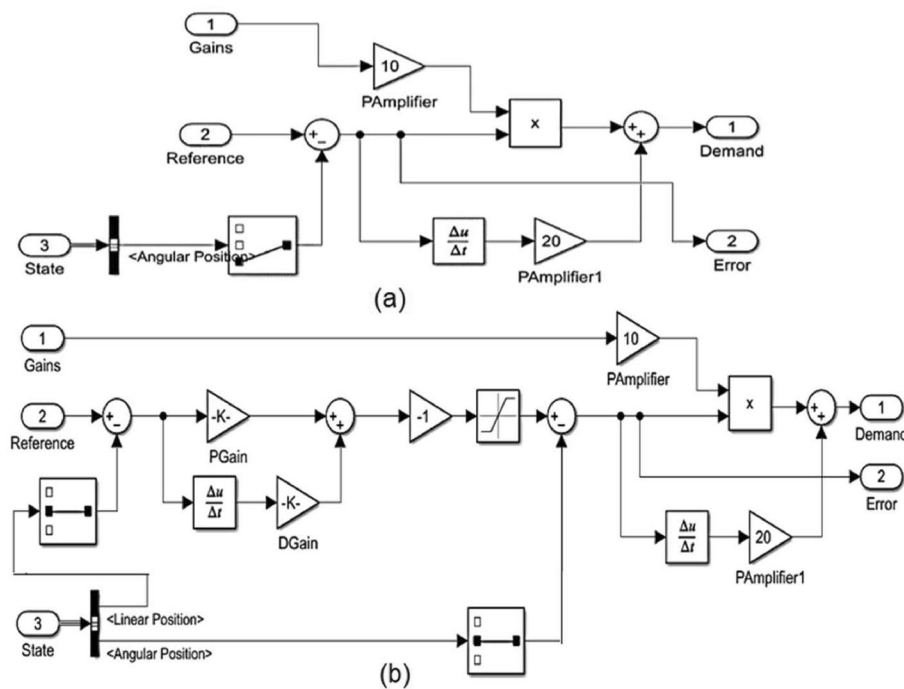
Controller	Rise time (s)			Settling time (s)			Overshoot (%)		
	Pitch	Roll	Yaw	Pitch	Roll	Yaw	Pitch	Roll	Yaw
RL	0.265	0.661	0.825	0.825	1.584	2.798	21	24	31
PID	1.344	0.228	0.962	0.962	2.05	2.198	4	17	35

problem multiple times bringing about the complexity and expensiveness of classical control methods.

Traditionally, if the controller is unstable upon deployment onto the hardware, it is further tuned to archive stability, but this method becomes unacceptable for mass produced systems such as drones with varying manufacturing tolerances and sensitive margins of stability. Controllers are often designed using soft models of the system for several reasons, mainly unavailability of hardware or to protect the hardware from possible damage of running from an imperfect controller. The challenge of designing a controller using software models is that they are often an inaccurate representation of the physical system especially for nonlinear systems. Physical properties such as sensor measurement noise and environment variations are often not included as part of the models since they are difficult to linearize. Therefore, when controller is deployed onto the actual hardware, it does not give the expected output, and this may result in an unstable system that could potentially damage the hardware.

Based on the previous discussions, it can be inferred that ANN is beneficial in enhancing the performance of control systems, particularly PID controllers. This work adopts a comparable strategy, utilizing an ANN-based cascaded PID controller to regulate various parameters of a mass-varying octocopter, such as altitude, roll, pitch, and yaw. Figure 1 shows the architecture of the proposed model deployed for the control of a quadcopter.

The remaining of the paper is organized as follows: The technique, including the octocopter model and dynamic equations, is in “[Methodology](#)” section. “[Results and discussion](#)” section shows the effectiveness of the control system and discusses its simulation



**Fig. 1** ANN-PID control system architecture for a quadcopter [27]

results. Finally, “Conclusion” section concludes, highlights the limitations, and suggests future research.

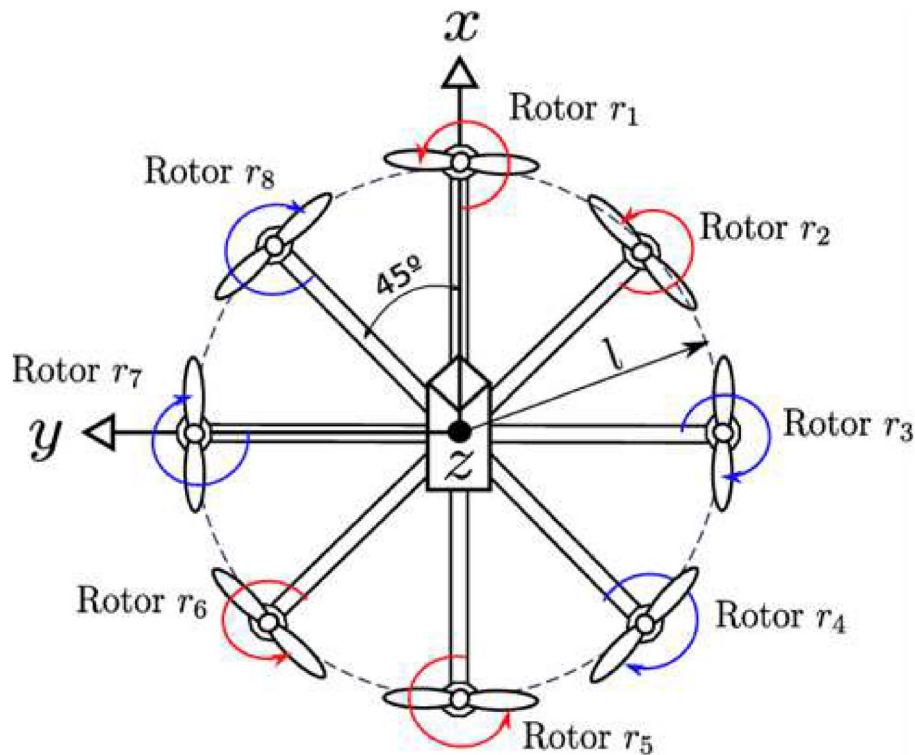
## Methodology

### Mathematical modelling of the octorotor dynamics

The kinematics and dynamics of a multirotor aircraft are typically analysed in two reference frames viz the earth inertial frame and the body-fixed frame. The earth inertial frame is oriented such that the gravity points are in the negative direction of the  $z$ -axis, while the coordinate axes of the body frame align with the rotors of the octocopter.

The octocopter comprises of eight DC motors located at the ends of the rotor arms. Each motor has a propeller mounted on its output shaft to generate the required thrust. As illustrated in Fig. 2, rotors 1, 2, 5, and 6 rotate in the counterclockwise direction with angular velocities  $\omega_1, \omega_2, \omega_5,$  and  $\omega_6,$  respectively. On the other hand, rotors 3, 4, 7, and 8 rotate in the clockwise direction with angular velocities  $\omega_3, \omega_4, \omega_7,$  and  $\omega_8,$  respectively.

The location of the octocopter’s centre of mass in the earth frame is represented by  $\varepsilon = [X \ Y \ Z]^T$ , whereas the angular position,  $\eta$ , is expressed in the inertial frame using Euler angles denoted by  $\eta = [\Phi \ \theta \ \Psi]^T$ . In addition, the absolute linear velocities of the octocopter are represented by  $V_B = [V_X \ V_Y \ V_Z]^T$ , and the angular velocities with respect to the Euler angles are represented by  $v = [P \ Q \ R]$ . Both of these velocity components are defined in the body frame. The relationship between these two frames is expressed using the rotation matrix  $R_1$ , given as follows [30],



**Fig. 2** Octorotor setup ([29])

$$R_1 = \begin{bmatrix} C_\Psi C_\theta & C_\Psi S_\theta S_\Phi - S_\Psi C_\Phi & C_\Psi S_\theta C_\Phi + S_\Psi S_\Phi \\ S_\Psi C_\theta & S_\Psi S_\theta S_\Phi + C_\Psi C_\Phi & S_\Psi S_\theta C_\Phi - C_\Psi S_\Phi \\ -S_\theta & C_\theta S_\Phi & C_\theta C_\Phi \end{bmatrix} \tag{1}$$

It should be noted that all of the motors in the octocopter are identical. However, the following derivation will only consider the operation of a single motor. According to momentum theory, the thrust generated by a single motor-propeller system on the octocopter is given by [31],

$$R_1 = T_i = C_D \rho A r^2 \omega_i^2 \tag{2}$$

In Eq. (2),  $C_D$  represents the thrust coefficient of the motor,  $\rho$  represents the density of air,  $A$  is the cross-sectional area of the propeller's rotation,  $r$  is the radius of the rotor, and  $\omega_i$  is the angular speed of the rotor. In the case of simple flight motion, a lumped parameter approach can be used to simplify Eqs. (2)–(3) as follows,

$$T_i = K \omega_i^2 \tag{3}$$

Combining the thrust from all the eight motor-propeller systems, the net thrust in the body frame  $z$ -direction is given in Eq. (4) as follows,

$$T = K \sum \omega_i^2 \tag{4}$$

giving the net thrust acting on the octorotor in the body frame as follows,

$$F^B = [0 \quad 0 \quad T]^T \tag{5}$$

In addition to thrust, a drag force also acts on the octocopter, which is a resisting force. This force has components along the coordinate axes in the inertial frame that is directly proportional to the corresponding velocities. The drag force can be expressed in component form of Eq. (6) as follows,

$$F^D = \begin{bmatrix} A_x & 0 & 0 \\ 0 & A_y & 0 \\ 0 & 0 & A_z \end{bmatrix} \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix} \tag{6}$$

where  $A_x$ ,  $A_y$ , and  $A_z$  are the drag coefficients in the  $x$ ,  $y$ , and  $z$  directions. If all the rotor velocities are equal, the octocopter will experience a force in the  $z$ -direction and will move up, hover, or fall depending on the magnitude of the force relative to gravity. The moments acting on the octocopter cause pitch, roll, and yaw motion. The pitching moment  $M_\Phi$  occurs due to the difference in thrust produced by motors 8 and 2, and 4 and 6. The rolling moment  $M_\Theta$  occurs due to the difference in thrust produced by motors 2 and 4, and 8 and 6.

$$\begin{aligned} M_\Phi &= L((T_2 + T_4) - (T_8 + T_6)) \\ M_\Theta &= L((T_8 + T_2) - (T_4 + T_6)) \end{aligned} \tag{7}$$

The yawing moment  $M$  is caused by the drag force acting on all the propellers and opposing their rotation. From the lumped parameter approach, it can be expressed as follows [31]:

$$\tau_{M_i} = B\omega_i^2 + I_R\dot{\omega}_i \tag{8}$$

where  $\tau_{M_i}$  is the torque produced by motor 1,  $B$  is the torque constant, and  $I_R$  is the inertia moment of rotor. The effect of  $\omega_i$  is very small and could be neglected.

$$M_\Psi = B(-\omega_1^2 - \omega_2^2 + \omega_3^2 + \omega_4^2 - \omega_5^2 - \omega_6^2 + \omega_7^2 + \omega_8^2) \tag{9}$$

The rotational moment acting on the octorotor in the body frame is given as follows [31],

$$F^B = [M_\Phi \quad M_\theta \quad M_\Psi]^T \tag{10}$$

The body frame experiences a resistive torque known as rotational drag, which is directly proportional to the body’s angular velocities. The equation for rotational drag is as follows:

$$M^R = [A_rP \quad A_rQ \quad A_rR]^T \tag{11}$$

where  $A_r$  is the rotational drag coefficient. The presented model has been simplified by neglecting various intricate phenomena such as blade flapping, which involves the deformation of blades at high velocities with flexible materials, and surrounding wind velocities. The dynamic equations of motion for the octorotor have been derived using the Newton–Euler formulation. It is assumed that the octorotor has a symmetrical structure, and therefore, the inertia matrix is both diagonal and time-invariant, with  $I_{XX} = I_{YY}$ .

$$I = \begin{bmatrix} I_{XX} & 0 & 0 \\ 0 & I_{YY} & 0 \\ 0 & 0 & I_{ZZ} \end{bmatrix} \tag{12}$$

The force that results in the acceleration of the mass  $m\dot{V}_B$ , as well as the centrifugal force  $v \times (mV_B)$ , is equivalent to the gravity force  $R^T G$  and the total external thrust force  $F^B$ , and the force of aerodynamic drag  $R^T F^D$ , all measured in the body frame and are related according to Eq. (13) [31].

$$m\dot{V}_B + v \times (mV_B) = R^T G + F^B - R^T F^D \tag{13}$$

When dealing with an octorotor, it is more practical to describe the dynamics using a mixed frame  $\{M\}$  that considers translational dynamics with respect to the inertial frame  $\{O\}$  and rotational dynamics with respect to the body frame  $\{B\}$ . In the inertial frame, centrifugal effects can be disregarded, and the only forces that need to be taken into account are the gravitational force, thrust, drag, and the mass acceleration of the octorotor. The dynamic equation could now be written in the form of Eq. (14).

$$\begin{bmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{z} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ -g \end{bmatrix} + R \frac{F^B}{m} - \frac{F^D}{m} \tag{14}$$

After some substitutions and taking the component form gives the dynamic equation for translational motion in the form of Eqs. (15) and (16).

$$\begin{bmatrix} \dot{X} \\ \dot{Y} \\ \dot{Z} \end{bmatrix} = \begin{bmatrix} U \\ V \\ W \end{bmatrix} \quad (15)$$

$$\begin{aligned} \dot{U} &= (\sin\Phi\sin\Psi + \cos\Phi\sin\theta\cos\Psi)\frac{T}{m} - \frac{A_x}{m}U \\ \dot{V} &= (-\sin\Phi\cos\Psi + \cos\Phi\sin\theta\sin\Psi)\frac{T}{m} - \frac{A_y}{m}V \\ \dot{W} &= -g + (\cos\Phi\cos\theta)\frac{T}{m} - \frac{A_z}{m}W \end{aligned} \quad (16)$$

Focusing on the rotational dynamics in the body frame, the angular acceleration of the inertia,  $I$  represented by  $\dot{\nu}$ , is equivalent to the centripetal forces  $\nu \times (I\nu)$  and the gyroscopic forces  $\tau$ . These forces are equal to the external torque  $M_B$  and the torque generated by aerodynamic drag. Hence, the rotational equation becomes,

$$I\dot{\nu} + \nu \times (I\nu) + \tau = M^B - M^D \quad (17)$$

giving

$$\dot{\nu} = I^{-1} \left( - \begin{bmatrix} P \\ Q \\ R \end{bmatrix} \times \begin{bmatrix} I_{xx}P \\ I_{yy}Q \\ I_{zz}R \end{bmatrix} - I_R \begin{bmatrix} P \\ Q \\ R \end{bmatrix} \times \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \Omega + M_B - M_D \right) \quad (18)$$

where  $\Omega = -\omega_1^2 - \omega_2^2 + \omega_3^2 + \omega_4^2 - \omega_5^2 - \omega_6^2 + \omega_7^2 + \omega_8^2$  and  $I_R$  is the rotational inertia of each motor written in component form as follows,

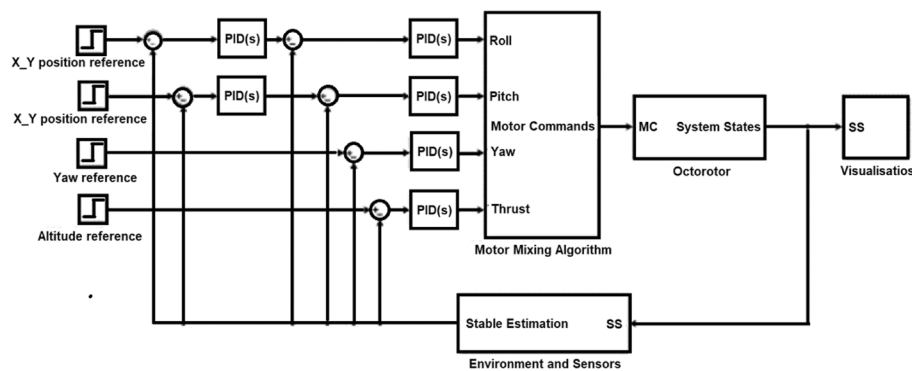
$$\begin{aligned} \dot{P} &= \frac{I_{YY} - I_{XX}}{I_{XX}}PQ - \frac{I_R}{I_{XX}}P\phi + \frac{M_\theta}{I_{XX}} \\ \dot{Q} &= \frac{I_{ZZ} - I_{XX}}{I_{YY}}PQ - \frac{I_R}{I_{YY}}Q\Omega + \frac{M_\theta}{I_{YY}} \\ \dot{R} &= \frac{I_{XX} - I_{YY}}{I_{ZZ}}QP - \frac{I_R}{I_{ZZ}}Q\Omega + \frac{M_\phi}{I_{ZZ}} \end{aligned} \quad (19)$$

### PID flight controller

The octorotor system consists of six control variables of interest. These are  $(\Phi, \theta, \Psi, X, Y, Z)$  and take in only eight inputs from the controller. The flight controller will take in two sets of inputs viz the octorotor system states and the reference command signal. The set of system state signals will comprise of 12 values, and they are as follows: six linear and angular positions  $(x, y, z, \Phi, \theta, \Psi)$ , and six velocities  $(dx, dy, dz, d\Phi, d\theta, d\Psi)$ . The reference command signal set will consist of the following signals: live time ticks, orientation reference, positions reference, take-off flag, and control mode vs orientation flag. There is a single output from the flight controller, and it comprises of the eight motor commands. The flight controller consists of six different PID controllers, connected in a cascaded manner; two inner loop and a single outer loop as shown in Fig. 3. The simulation ran for 300 s with MATLAB default rate of 0.005 s.

The angular motion of the UAV is not influenced by the translational components, whereas the translational motion is dependent on the Euler angles. Hence, the primary





**Fig. 3** Cascaded PID flight controller

objective is to regulate the rotational behaviour of the vehicle due to its independence from translational motion, followed by controlling the translational behaviour. To control the attitude of the vehicle, it is necessary to have sensors that can measure the orientation of the vehicle, actuators that can apply the required torques to re-orient the vehicle to the desired position, and algorithms that can command the actuators based on (1) sensor measurements of the current attitude and (2) specification of a desired position. Once modelling is completed, the controllers were each tuned using the Simulink auto-tuner. The altitude controller is tuned first since it has no state behaviour dependent inputs and is essential for other controllers to work. The same process is taken for the yaw controller. The gains obtained from the yaw tuning process were passed on to the pitch and roll controllers. Lastly, the  $X$  and  $Y$  position controllers were tuned since they require a functioning pitch and roll controller in order to work as expected.

*Altitude controller* This PD controller has three inputs; system altitude reference and current system altitude and, lastly, the gain multiplier for the proportional part and outputs thrust.

*Yaw controller* This is a PD controller with only three inputs; the reference yaw angle, the current system yaw angle, and the proportional gain multiplier. It has a single output that is the motor torque difference for the motor configurations.

*Pitch and roll controllers* These two have the same architecture and shall be full PD controllers with only three inputs; their respective reference angles to be computed by the position controller, their respective current system angles, and lastly their respective proportional gain multiplier. They have a single output that is the motor torque difference for the motor configurations.

*Pitch–roll position controllers* These are PD controller with four inputs each; the current yaw angle, the reference system  $x$ ,  $y$  positions, and lastly the current system  $x$ ,  $y$  positions. They have a single output that is the pitch and roll reference angles for the pitch and roll controllers, respectively

### Self-tuning neural network

Artificial neural networks are designed based on the structure and function of biological neurons. In a neural network, each neuron is represented by a weight function, similar to the synaptic connections in natural neurons. The neural network processor is composed of two components: The first component calculates the weighted inputs,

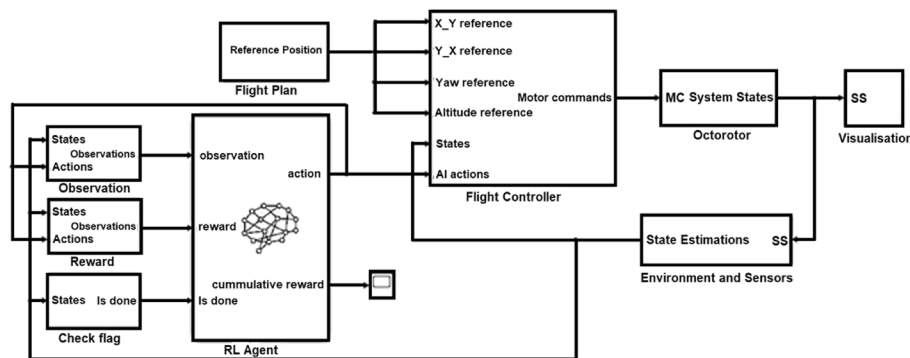
while the second component is an activation function that acts as a filter for the neuron. The input for each neuron is multiplied by the corresponding weight and then passed on to the adjacent neurons through an activation function. The neural network developed in this study has 16 inputs and four outputs, where the inputs are reference signal, error, and plant output, while the outputs are Kp values for each PID controller. As the input parameters of the neural network change, the output values also change accordingly. The tuning algorithm proposed in this study consists of two hidden layers, with a leaky RELU activation function used for the hidden layers and tanh for the last layer. The structure of the complete Simulink model of the entire system is shown in Fig. 4.

A neural network can enhance a PID controller by dynamically adjusting its proportional, integral, and derivative gains based on real-time system conditions. This process involves collecting data on system states, errors, and optimal PID gains under various conditions. A neural network is then designed and trained using this data to predict the optimal PID gains. Once trained, the neural network inputs current system states and external conditions, outputs the tuned PID gains, and feeds these values into the PID controller. This integration allows the PID controller to adapt to varying system dynamics, improving its performance and responsiveness.

**Development**

The first step in training a Simulink-built model with reinforcement learning is to define the environment, which is the simulation or real-world system in which the agent interacts to learn. In this case, the environment is the Simulink model of the octorotor that has been defined in the previous sections through equations. With the state, action, and reward spaces of the environment defined, as well as any other relevant parameters that training procedure may follow as shown in Table 2. The “rlSimulinkEnv” function in MATLAB [32] is used to create the environment object. The function takes the name of the Simulink model as an argument and returns an environment object. The reset function is defined for the environment to specify how the environment should be reset after each episode. The reset function takes an input argument and returns an output argument that specifies the initial state of the environment.

The second step is to define the agent, which is an algorithm that takes the state of the environment as input and outputs an action. In this case, a DDPG network is used with a leaky ReLU activation function. The DDPG algorithm is a model-free,



**Fig. 4** Structure of the complete Simulink model of the system

**Table 2** Network architecture parameters

Variable name	Value
Discount factor	0.99
Mini batch size	128
Experience buffer length	1e6
Target smooth factor	1e−3
Noise options mean attraction constant	0.15
Noise options variable	0.1

off-policy reinforcement learning algorithm that can learn policies in continuous action spaces. The structure of the DDPG network is defined using the MATLAB's deep learning toolbox. In it, it is important to specify the number of hidden layers and nodes in each layer, the activation function for each layer, and any other relevant parameters shown in Table 3. In addition, other training parameters such as the learning rate, exploration strategy, and other hyperparameters for the agent were defined in the MATLAB code.

The third step is to define the training parameters using the options shown in Table 4, such as the number of episodes or iterations of training, the maximum number of steps per episode, and the exploration strategy (e.g. how the agent should explore the state-action space). These were defined using the “*rlTrainingOptions*” function.

The fourth step is to implement the training loop, which consists of repeatedly interacting with the environment using the agent to learn from the experience. In each episode, the agent takes an action in the current state of the environment, receives a reward, and observes the next state. The agent then uses this experience to

**Table 3** Network parameters

Variable name	Actor value	Critic value
Algorithm	Adam	Adam
Learn rate	1e−4	1e−3
Gradient threshold	1	1
L2 regularization factor	1e−5	2e−4

**Table 4** Training options

Parameter	Value
Max episodes	1000
Max steps	Floor (Tf/Ts)
Train opts	RL training options
Max steps per episode	Max steps
Score averaging window length	250
Plots	Training progress
Stop training criteria	Average reward
Stop training value	190
Save agent criteria	Episode reward
Save agent value	−700

update its network parameters. The “train” function is used to run the training loop. The function takes the environment object, agent object, and training options object as inputs and returns the trained agent object.

### **Reward**

During training, the agent will receive a reward at each time step. The reward function is designed to motivate the agent to move towards the desired position, which is achieved by providing a positive reward for positive forward velocity. Additionally, the reward function is intended to encourage the agent to approach the target position by providing positive rewards for minimizing the errors between the reference and current positions. Furthermore, the agent will be incentivized to avoid early termination by receiving a constant reward at each time step. The other components of the reward function are penalties that discourage undesirable states, such as high angular velocities and orientation angles beyond 45 degrees. The DDPG agents discussed above were developed as shown in Table 5.

### **Complete model test**

Once all the system were modelled, all the PD controllers manually tuned and the neural networks trained, the system is tested for a typical path following model. The path to be followed is as shown in Fig. 5.

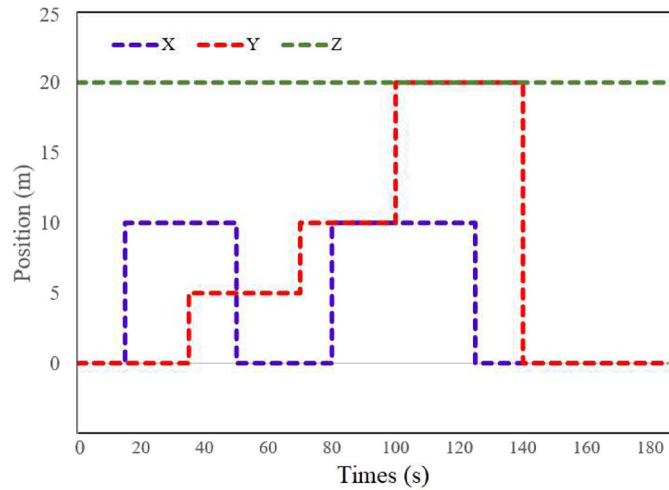
## **Results and discussion**

### **Complete model test**

The cascaded PD control architecture stated in “Methodology” section is developed with the proposed  $X$  and  $Y$  PD controllers imbedded within the pitch and roll blocks, respectively. The utilization of a PD is primarily motivated by several key factors. Firstly,

**Table 5** Summary of the DDPG agents

Network	Observations	Reward variables	Simulation boundaries	No. of hidden layers	Neurons per layer
Yaw agent	Angular position ( $Z$ -axis), angular velocity ( $Z$ -axis), yaw error, previous error	Yaw error, constant	$> 1.0472, > 1.0472$	2	400:300
Pitch agent	Angular position ( $X$ -axis), angular velocity ( $X$ -axis), pitch error, previous error	Pitch error, constant	$> 0.7854, < -0.7854$	2	400:300
Roll agent	Angular position ( $Y$ -axis), angular velocity ( $Y$ -axis), roll error, previous error	Roll error, constant	$> 1.0472, < -1.0472$	2	400:300
Altitude agent	Angular position ( $Z$ -axis), angular velocity ( $Z$ -axis), altitude error, previous error	Altitude error, constant	$> 50, < -50$	2	400:300:150



**Fig. 5** Flight path

**Table 6** PD tuning gain results

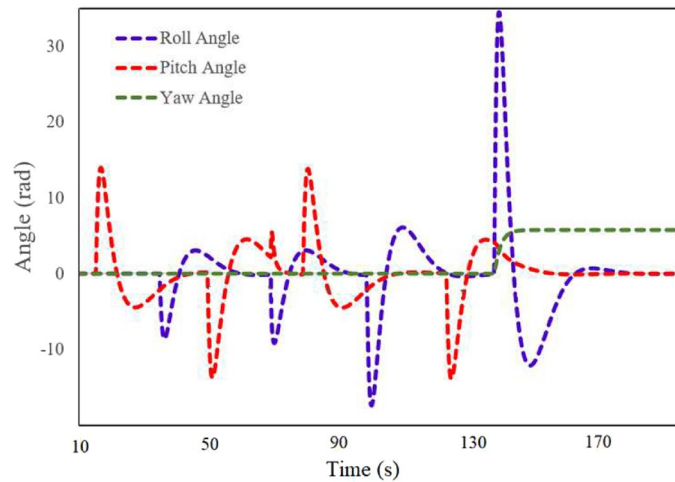
Controller name	Proportional gain	Derivative gain	Input	Output
PD controller yaw	10	20	Yaw angle	Yaw Torque
PD controller P (X-axis)	0.008	0.05	Linear X position	Pitch angle
PD controller pitch	10	20	Pitch angle	Pitch torque
PD controller R (Y-axis)	0.01	0.05	Linear Y position	Roll angle
PD controller roll	10	20	Roll angle	Roll torque
PD controller altitude	40	25	Linear Z position	Thrust

PD controllers effectively avoid the issue of integral windup, a common problem in PID controllers where the integral term can accumulate error over time, leading to excessive overshoot and potential instability. Secondly, the process of tuning a PD controller is more straightforward due to the absence of the integral term, resulting in fewer parameters to adjust and facilitating quicker and more efficient optimization. Moreover, for many UAV applications, the proportional and derivative actions provided by a PD controller are sufficient to achieve the desired stability and performance, as the primary function of integral term is to eliminate steady-state error that is less critical in dynamic systems like UAVs, where maintaining rapid response and stability is paramount. In the context of this study, these translates to fewer parameters, which simplifies the training process and reduces the training time for the neural network agents.

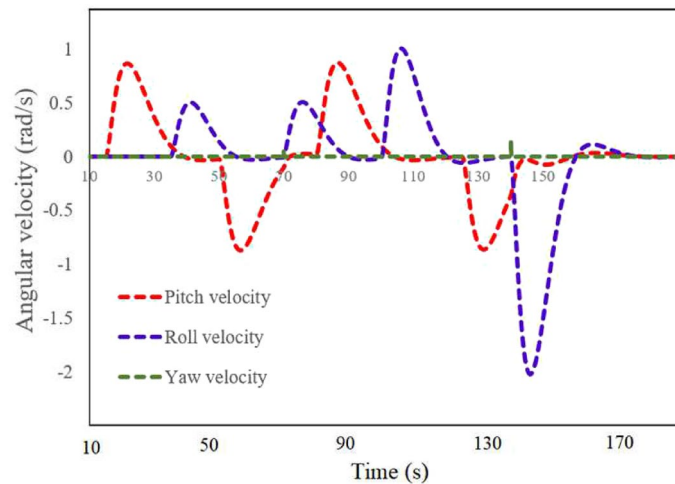
As previously discussed, the proportional gain of the PD controllers is taken from an external source and multiplied with the state error of the respective controller. Using the process described in the methodology, the individual controllers were tuned using the Simulink auto-tuner, and the results of this progress are as shown in Table 6.

**Flight simulation results**

Using the flight path defined in Fig. 5, the simulation is run to test the stability of the cascaded PD controllers and neural network tuner, and the results are as shown in



**Fig. 6** Angular position simulation results

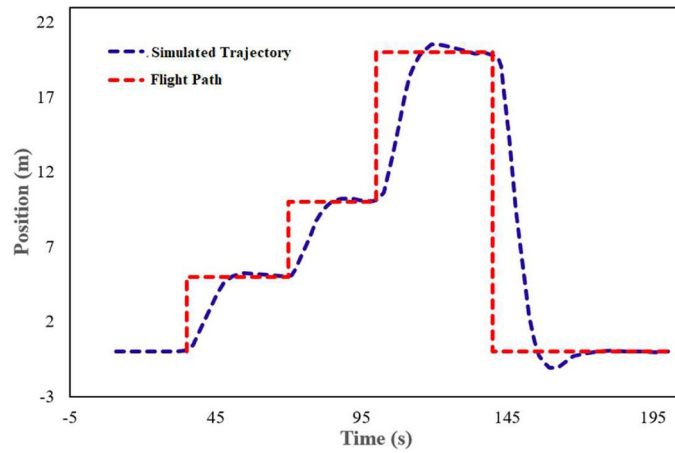


**Fig. 7** Angular velocity simulation results

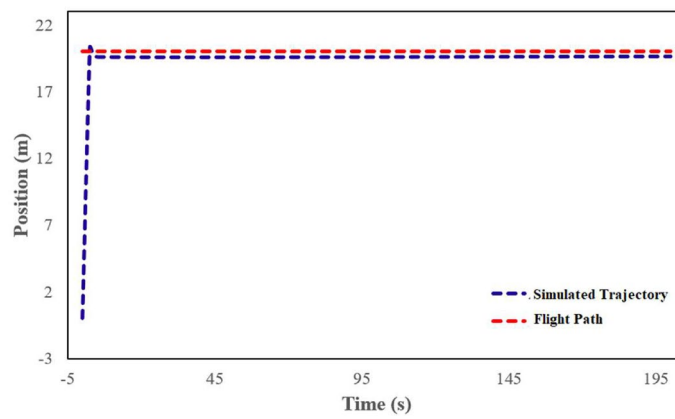
Figs. 6, 7, 8, 9, 10, and 11. In the flight data, there were five major positions of interest in the time scale, and these were at 0 s, 15 s, 35 s, 70 s, and lastly 140 s.

The designed position trajectory and the simulated trajectory have been plotted on the same axes as shown in Figs. 8, 9, and 10. The results show a high level of consistency of the simulated trajectory with the flight path. At the 0-s mark, the system had to take off from the ground and rise to a height of 20 m, which is to be maintained for the remainder of the system. The altitude increases from 0 to about 19 m, and this height was maintained for the remainder of the flight with a steady-state error of 1 m.

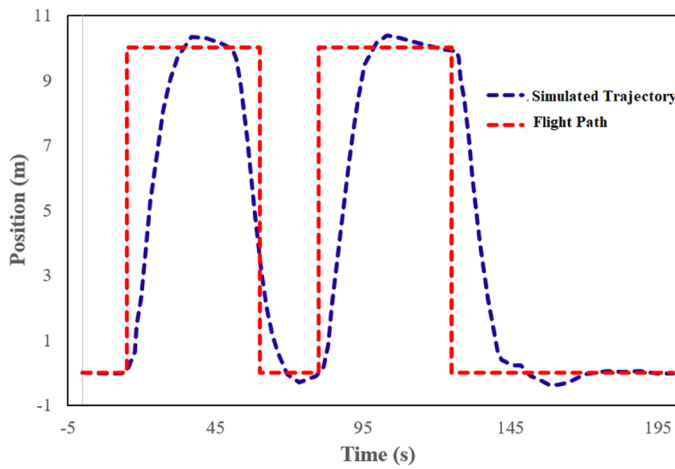
At the 15-s mark, the second major event took place where the UAV is expected to move from 0 to 10 m along the  $x$ -axis. In order to execute this manoeuvre, the  $X$  controller must detect a position error and command the pitch controller to take action in the right direction. As shown in Fig. 7, the angular position of the UAV increased along with the speed shown in Fig. 6. This caused a distribution of the force along



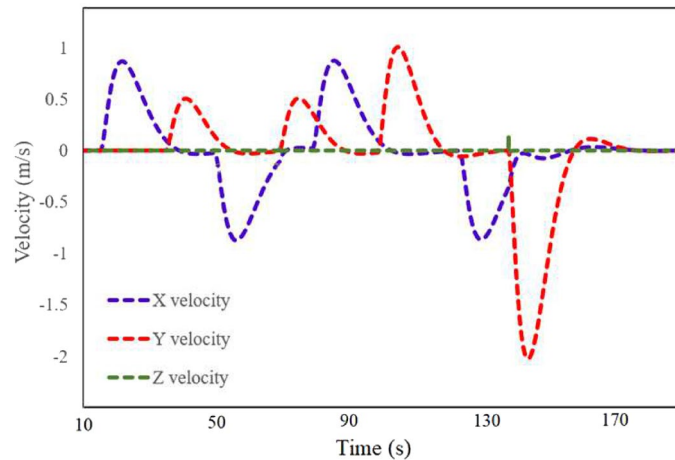
**Fig. 8** Linear X position simulation results



**Fig. 9** Linear Y position simulation results



**Fig. 10** Linear Z position simulation results



**Fig. 11** Linear velocity simulation results

**Table 7** Control behaviour of the controllers

Controller	Rise time (s)			Settling time (s)			Overshoot (%)		
	Pitch	Roll	Yaw	Pitch	Roll	Yaw	Pitch	Roll	Yaw
DDPG	0.913	2.417	4.8395	1.493	5.778	12.977	12.077	13.667	11.35

the  $z$ - and  $x$ -axis. This distribution initiated velocity in the  $x$ -axis and as expected the UAV moved to the 10-m mark. The distribution also decreased the  $z$  component of trust, and therefore, the altitude controller had to react to this change in order to maintain the desired position, and this was successfully done as shown in Fig. 10. The pitch controller was stable, and the 10-m command was maintained for the desired period of time.

Similarly at the 35-s mark, the  $Y$  and roll controllers were engaged in order to change the  $y$  position of the UAV through the same concept discussed above. The  $y$  position is to be changed by 5 m, and the two controllers like the previous reacted with adequate stability.

The 140-s mark is the most critical since it tested the stability of all six controllers simultaneously. The  $X$  controller is commanded to maintain the 0-m position and, therefore, creates no component of force along the  $x$ -axis, the same is done for the altitude controller which is to maintain a height of 20 m by creating zero force difference in the  $z$ -axis, the  $Y$  controller is instructed to change the  $Y$  position from 20 m to 0, and the yaw controller is instructed to change orientation to 0.1 radians. The combination of all these controllers is most likely going to create a backlog in the motor mixing algorithm and thus negatively affecting the results, but because of the NN auto-tuner, this is averted. It could be seen that the yaw controller successfully changed the orientation from and maintained stability while the  $Z$  velocity spike in Fig. 7 is evidence that the altitude controller also responded to the situation. The introduction of an altered orientation meant that the  $Y$  controller’s efforts to subsidize error introduced error in the  $X$  controller that had since archived stability, and as a result, the pitch controller reacted to this change. As shown in Table 7, the DDPG network improved the overshoot



of the system, and this is due to the maintained derivative gain. The results of the present study are consistent with the results presented by Bohn et al. [33].

## Conclusion

The octorotor model created using Simulink, takes into account motor actuation, air resistance, gravity, climatic conditions, and sensor variations. By conducting level and roll tests, both single motor and complete actuation designs exhibited outcomes that were consistent with the anticipated performance for the specified motor speed. This alignment necessitates precise modelling of air resistance, gravitational forces, ambient conditions, and sensor performance. The airframe model, when exposed to a level test, verifies the accurate representation of the UAV system. In addition, a cascaded proportional-derivative control architecture has been successfully constructed, calibrated, evaluated, and integrated into the unmanned aerial vehicle (UAV) system.

In order to enhance the performance of UAVs, a deep deterministic policy gradient network is built and trained using reinforcement learning techniques in MATLAB. The findings, assessed for stability using simulated flight routes, demonstrate the UAV's capacity to accurately track the assigned trajectory. Significantly, each of the four principal controllers operates autonomously without any influence with the aims of the others.

The integration of DDPG with PD controllers for multirotor UAVs reduces the significant manual effort traditionally required, ensuring consistent and optimal performance across various UAVs and flights. This automation not only saves time but also minimizes the dependency on expert knowledge, making advanced UAV control more accessible.

Moreover, this work is a pivotal step towards achieving fully autonomous UAV operations. By enabling UAVs to self-tune and optimize their control strategies in real-time, it sets the foundation for more sophisticated autonomous systems capable of adapting to diverse and unpredictable environments. This enhanced autonomy can revolutionize a myriad of applications, from delivery services to emergency response, by providing more reliable and efficient UAV performance.

In addition to practical benefits, this paper contributes significantly to the academic and industrial fields. It showcases the innovative application of reinforcement learning in control systems, offering new methodologies and insights that can spur further research and development. This work not only demonstrates the potential of combining machine learning with traditional control techniques but also paves the way for future advancements, ultimately driving innovation in UAV technology and control systems.

The implementation of DDPG with PD controllers for multirotor UAVs offers significant potential for enhancing autonomous operations. However, the research faces notable challenges related to the complexity and time required for training. Overcoming these hurdles will be critical in realizing the full benefits of this approach.

Moving forward, focusing on extending the DDPG approach to multi-agent systems will be pivotal. This evolution promises to revolutionize sectors such as search and rescue, surveillance, and environmental monitoring by leveraging cooperative behaviour among UAVs. By harnessing the power of reinforcement learning in this context, we can expect significant advancements in efficiency, scalability, and

adaptability, paving the way for a future where autonomous UAV fleets operate seamlessly and effectively in complex dynamic environments.

#### Abbreviations

AEF	Aerodynamic and external forces
AFC	Adaptive filter controller
AI	Artificial intelligence
ANN	Artificial neural networks
ARC	Aspect ratio change
BPNN	Back propagation neural network
CNN	Convolutional neural network
CPU	Central processing unit
CVM	Custom varying mass
DDPG	Deep deterministic policy gradient
DL	Deep learning
DQN	Deep Q-network
EKF	Extended Kalman filter
FL	Feedback linearization
GA	Genetic algorithms
IMC	Internal model control
IMU	Inertial measurement unit
KF	Kalman filter
MIMO	Multi-input multi-output
ML	Machine learning
MLP	Multiple layer perceptron
MPC	Model predictive control
MTRBL	Multi-task regression-based learning
NN	Neural network
PID	Proportional, integral, and derivative
PPO	Proximal policy optimization
RL	Reinforcement learning
RLS	Recursive least squares
RNN	Recurrent neural network
SISO	Single-input single-output
SLC	Successive loop closure
SMC	Sliding mode control
SVSF	Smooth variable structure filter
TRPO	Trust-region policy optimization
UAV	Unmanned aerial vehicles
VTOL	Vertical take-off and landing
$\ddot{\theta}$	Euler pitch angular acceleration (rad/s <sup>2</sup> )
$\dot{\theta}$	Euler pitch angular velocity (rad/s)
$\theta$	Euler pitch angle (rad)
$\phi$	Euler roll angle (rad)
$\ddot{\phi}$	Euler roll angular acceleration (rad/s <sup>2</sup> )
$\dot{\phi}$	Euler roll angular velocity (rad/s)
$\psi$	Euler yaw angle (rad)
$\ddot{\psi}$	Euler yaw angular acceleration (rad/s <sup>2</sup> )
$\dot{\psi}$	Euler yaw angular velocity (rad/s)
$X$	Position along the x-axis (m)
$\dot{X}$	Linear velocity along the x-axis (m/s)
$\ddot{X}$	Linear acceleration along the x-axis (m/s <sup>2</sup> )
$Y$	Position along the y-axis (m)
$\dot{Y}$	Linear velocity along the y-axis (m/s)
$\ddot{Y}$	Linear acceleration along the y-axis (m/s <sup>2</sup> )
$Z$	Position along the z-axis (m)
$\dot{Z}$	Linear velocity along the z-axis (m/s)
$\ddot{Z}$	Linear acceleration along the z-axis (m/s <sup>2</sup> )
$\Omega/\omega$	Motor angular velocity (rad/s)
$I_{xx}$	Moment of inertia in the x-axis (Kg m <sup>2</sup> )
$I_{yy}$	Moment of inertia in the y-axis (Kg m <sup>2</sup> )
$I_{zz}$	Moment of inertia in the z-axis (Kg m <sup>2</sup> )
$M_{\theta}$	Pitch moment (Nm)
$M_{\phi}$	Roll moment (Nm)
$M_{\psi}$	Yaw moment (Nm)
$\rho$	Density (Kg/m <sup>3</sup> )

**Author contributions**

E. Mosweu helped in conceptualization and writing of the original draught, T. B. Seokolo contributed to writing of the original draught, T. T. Akano worked in supervision and review and editing, and O.S. Motsamai worked in supervision and review.

**Funding**

Not applicable.

**Availability of data and materials**

Not applicable.

**Declarations****Ethics approval and consent to participate**

Not applicable. Consent granted is granted by authors.

**Consent for publication**

Consent for publication is granted by authors.

**Competing interests**

The authors declare that they have no competing interests.

Received: 11 December 2023 Accepted: 9 July 2024

Published online: 19 July 2024

**References**

1. ClimateWire NM Humans may be the most adaptive species. <https://www.scientificamerican.com/article/humans-may-be-most-adaptive-species/>. Accessed 12 Feb 2023
2. Badawy M, Ramadan N, Hefny HA (2023) Healthcare predictive analytics using machine learning and deep learning techniques: a survey. *J Electr Syst Inf Technol*. <https://doi.org/10.1186/s43067-023-00108-y>
3. Mitchell TM (1988) (ed.): Machine Learning: a Guide to Current Research, 3. print edn. In: Kluwer international series in engineering and computer science Knowledge representation, learning and expert systems, vol. 12. Kluwer, Boston
4. Alpaydin E (2020) Introduction to machine learning. In: Adaptive computation and machine learning series. MIT Press, Cambridge
5. Illman PE (2000) The pilot's handbook of aeronautical knowledge. In: United States Department of Transportation, Federal Aviation Administration, Airman Testing Standards Branch, p 471
6. Chapman WL, Bahill AT, Wymore AW (2018) Engineering modeling and design, 1st edn. CRC Press. <https://doi.org/10.1201/9780203757314>
7. Burns RS (2001) Advanced control engineering. Butterworth-Heinemann, Oxford, Boston OCLC: ocm47823330
8. Malik W, Hussain S (2019) Developing of the smart quadcopter with improved flight dynamics and stability. *J Electr Syst Inf Technol*. <https://doi.org/10.1186/s43067-019-0005-0>
9. Sielly Jales Costa B, Greati VR, Campos Tinoco Ribeiro V, Da Silva CS, Vieira IF (2015) A visual protocol for autonomous landing of unmanned aerial vehicles based on fuzzy matching and evolving clustering. In: 2015 IEEE international conference on fuzzy systems (FUZZ-IEEE), pp 1–6. IEEE, Istanbul. <https://doi.org/10.1109/FUZZ-IEEE.2015.7337907>
10. Paddy RP, Ahmad S, Verma S, Sa PK, Bakshi S (2019) Localization of unmanned aerial vehicles in corridor environments using deep learning. <https://doi.org/10.48550/ARXIV.1903.09021>. Publisher: arXiv Version Number: 1
11. Villanueva A, Fajardo A (2019) UAV navigation system with obstacle detection using deep reinforcement learning with noise injection. In: 2019 International conference on ICT for smart society (ICISS), pp. 1–6. IEEE, Bandung, Indonesia. <https://doi.org/10.1109/ICISS48059.2019.8969798>
12. Cano Lopes G, Ferreira M, Da Silva Simoes A, Luna Colombini E (2018) Intelligent control of a quadrotor with proximal policy optimization reinforcement learning. In: 2018 Latin American robotic symposium, 2018 Brazilian symposium on robotics (SBR) and 2018 workshop on robotics in education (WRE), pp 503–508. IEEE, Joao Pessoa. <https://doi.org/10.1109/LARS/SBR/WRE.2018.00094>
13. Schulman J, Wolski F, Dhariwal P, Radford A, Klimov O (2017) Proximal policy optimization algorithms. <https://doi.org/10.48550/ARXIV.1707.06347>. Publisher: arXiv Version Number: 2
14. Cardenas JA, Carrero UE, Camacho EC, Calderon JM (2023) Intelligent position controller for unmanned aerial vehicles (UAV) based on supervised deep learning. *Machines* 11(6):606. <https://doi.org/10.3390/machines11060606>
15. Mohammed FA, Bahgat ME, Elmasry SS, Sharaf SM (2022) Design of a maximum power point tracking-based PID controller for DC converter of stand-alone PV system. *J Electr Syst Inf Technol*. <https://doi.org/10.1186/s43067-022-00050-5>
16. Maciel-Pearson BG, Akcay S, Atapour-Abarghouei A, Holder C, Breckon TP (2019) Multi-task regression-based learning for autonomous unmanned aerial vehicle flight control within unstructured outdoor environments. *IEEE Robot Autom Lett* 4(4):4116–4123. <https://doi.org/10.1109/LRA.2019.2930496>
17. Xu J, Guo Q, Xiao L, Li Z, Zhang G (2019) Autonomous decision-making method for combat mission of UAV based on deep reinforcement learning. In: 2019 IEEE 4th advanced information technology, electronic and automation control conference (IAEAC), pp 538–544. IEEE, Chengdu, China. <https://doi.org/10.1109/IAEAC47372.2019.8998066>

18. Cho S, Kim DH, Park YW (2017) Learning drone-control actions in surveillance videos. In: 2017 17th International conference on control, automation and systems (ICCAS), pp 700–703. IEEE, Jeju. <https://doi.org/10.23919/ICCAS.2017.8204319>
19. Bouhamed O, Ghazzai H, Besbes H, Massoud Y (2020) Autonomous UAV navigation: a DDPG-based deep reinforcement learning approach. In: 2020 IEEE international symposium on circuits and systems (ISCAS), pp 1–5. IEEE, Seville, Spain. <https://doi.org/10.1109/ISCAS45731.2020.9181245>
20. Sewak M (2019) Deep Q network (DQN), double DQN, and dueling DQN: a step towards general artificial intelligence. In: Deep reinforcement learning, pp 95–108. Springer, Singapore. [https://doi.org/10.1007/978-981-13-8285-7\\_8](https://doi.org/10.1007/978-981-13-8285-7_8)
21. Zulu A, John S (2014) A review of control algorithms for autonomous quadrotors. *OJAppS* 04(14):547–556. <https://doi.org/10.4236/ojapps.2014.414053>
22. Shao-yuan L (2009) Adaptive PID control for nonlinear systems based on lazy learning. *Control Theory Appl*
23. Nuella I, Cheng C, Chiu M-S (2009) Adaptive PID controller design for nonlinear systems. *Ind Eng Chem Res* 48(10):4877–4883. <https://doi.org/10.1021/ie801227d>
24. Malekabadi M, Haghparast M, Nasiri F (2018) Air condition's PID controller fine-tuning using artificial neural networks and genetic algorithms. *Computers* 7(2):32. <https://doi.org/10.3390/computers7020032>
25. Essalmi A, Mahmoudi H, Abbou A, Bennassar A, Zahraoui Y (2014) DTC of PMSM based on artificial neural networks with regulation speed using the fuzzy logic controller. In: 2014 International renewable and sustainable energy conference (IRSEC), pp 879–883. IEEE, Ouarzazate, Morocco. <https://doi.org/10.1109/IRSEC.2014.7059801>
26. Hernández-Alvarado R, García-Valdovinos L, Salgado-Jiménez T, Gómez-Espinosa A, Fonseca-Navarro F (2016) Neural network-based self-tuning PID control for underwater vehicles. *Sensors* 16(9):1429. <https://doi.org/10.3390/s16091429>
27. Yoon G-Y, Yamamoto A, Lim H-O (2016) Mechanism and neural network based on PID control of quadcopter. In: 2016 16th International conference on control, automation and systems (ICCAS), pp 19–24. IEEE, Gyeongju, South Korea. <https://doi.org/10.1109/ICCAS.2016.7832294>
28. Bohn E, Coates EM, Moe S, Johansen TA (2019) Deep reinforcement learning attitude control of fixed-wing UAVs using proximal policy optimization. In: 2019 International conference on unmanned aircraft systems (ICUAS). IEEE. <https://doi.org/10.1109/icuas.2019.8798254>
29. Salazar JC, Sanjuan A, Nejjarri F, Sarrate R (2017) Health-aware control of an octorotor UAV system based on actuator reliability. In: 2017 4th International conference on control, decision and information technologies (CoDIT), pp 0815–0820. IEEE
30. Artale V, Milazzo C, Ricciardello A (2013) Mathematical modeling of hexacopter. *Appl Math Sci* 7(97):4805–4811. <https://doi.org/10.12988/ams.2013.37385>
31. Artale V, Milazzo CLR, Ricciardello A (2013) Mathematical modeling of hexacopter. *Appl Math Sci* 7:4805–4811. <https://doi.org/10.12988/ams.2013.37385>
32. MathWorks: MATLAB version: 9.12.0. The MathWorks Inc., Natick, Massachusetts, United States (2022). <https://www.mathworks.com>
33. Bohn E, Coates EM, Moe S, Johansen TA (2019) Deep reinforcement learning attitude control of fixed-wing UAVs using proximal policy optimization. In: 2019 International conference on unmanned aircraft systems (ICUAS). IEEE. <https://doi.org/10.1109/icuas.2019.8798254>

## Publisher's Note

Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.