applied sciences

Article

Deep Reinforcement Learning-Based Adaptive Controller for Trajectory Tracking and Altitude Control of an Aerial Robot

Ali Barzegar and Deok-Jin Lee *

School of Mechanical Engineering, Jeonbuk National University, Jeonju 54896, Korea; ali.barzegar@jbnu.ac.kr
* Correspondence: deokjlee@jbnu.ac.kr; Tel.: +82-63-270-4768

Abstract: This research study presents a new adaptive attitude and altitude controller for an aerial robot. The proposed controlling approach employs a reinforcement learning-based algorithm to actively estimate the controller parameters of the aerial robot. In dealing with highly nonlinear systems and parameter uncertainty, the proposed RL-based adaptive control algorithm has advantages over some types of standard control approaches. When compared to the conventional proportional integral derivative (PID) controllers, the results of the numerical simulation demonstrate the effectiveness of this intelligent control strategy, which can improve the control performance of the whole system, resulting in accurate trajectory tracking and altitude control of the vehicle.

Keywords: RL robot control; RL model predictive control; reinforcement learning for vehicle control; PID reinforcement learning; RL adaptive PID; reinforcement learning drone control

Citation: Barzegar, A.; Lee, D.-J. Deep Reinforcement Learning-Based Adaptive Controller for Trajectory Tracking and Altitude Control of an Aerial Robot. *Appl. Sci.* **2022**, *12*, 4764. https://doi.org/10.3390/app12094764

Academic Editors: Antoni Grau and Yolanda Bolea

Received: 14 April 2022
Accepted: 4 May 2022
Published: 9 May 2022

1. Introduction

1.1. Control Problem of Aerial Robots

Aerial robots have become increasingly popular in recent decades. Quadrotors, in particular, have piqued the attention of the scientific community, with several important discoveries and applications proposed and tested. Despite significant advancements, aerial robot control is still regarded as a very active field of research. Aerial robot controllers, on the one hand, need the ability to acquire, process, and calculate forces to apply to vehicle actuators in a very time-critical way. The flight controllers for aerial robots, on the other hand, should be able to resist failures and respond to changes in payload and disturbances. Flight control systems for aerial robots are usually implemented using proportional integral derivative (PID) control algorithms [1]. PIDs have proved their acceptable performance in some circumstances, such as racing drones, where fast control responses are crucial. A PID controller works close to optimally in stable settings and environments. Hence, most commercial aerial robot flight controllers use PIDs for both attitude and altitude control.

However, PID controllers are unable to properly control a robot when faced with unexpected dynamics disturbances (such as variable payloads) [2–12]. External disturbances (e.g., wind) can also reduce the accuracy of trajectory tracking by weakening attitude controller performance. The problem is that model-free controllers are unable to fully cover the complicated variations in nonlinear dynamics behavior of a quadrotor, causing the controller to lose stability and robustness [13]. This problem motivated some researchers to develop optimal and nonlinear model-based controllers to control aerial robots. Nonlinear Model Predictive Control (NMPC) is one of the most widely used optimal model-based control algorithms in many recent research works. The controller, as the name implies, uses a model of the system to forecast the future behavior of the robot in response to the current control input. A model predictive controller has some advantages over its model-free counterparts. Not only does the controller outperform its model-free equivalents in terms of control performance, but it also has the ability to take into account some constraints, which is a key element in some flight maneuvers such as obstacle avoidance [14]. Despite its benefits over previous model-free techniques, MPC is vulnerable to failure when the prediction model of the controller does not account for fluctuations in the dynamical system under control.

When it comes to controlling an aerial robot, it should be noted that the aerial robot controller needs to control both the altitude and the attitude of the robot. In order to implement a full optimal model-based controller for an aerial robot, some researchers proposed employing NMPC for both altitude and attitude controllers, while taking some measures to reduce its dependency on an accurate model of the robot. Some studies endeavored to mitigate the problem by using learning algorithms (e.g., a Gaussian process or neural networks) that actively estimate the dynamics parameters of the robot and update the prediction model in real time [15–20]. Although the resultant topologies improved the performance of the controller after learning the dynamical variations of the system, the methods have the potential to increase computation costs.

Despite the fact that the aforementioned optimal control algorithms partially managed to improve the overall performance of controllers, they need powerful and fast processors to compute the online optimization problem. In order to avoid imposing high computing costs on the system, a number of research studies proposed using linear model predictive control (LMPC) [21–23]. The advantage of LMPC is its low computation cost, as compared with its nonlinear counterpart. The controller, however, does not provide a good response for attitude control when there is too much variation in the parameters of the dynamical system. The problem stems from the fact that the prediction model in the linear MPC uses a linearized model around its ideal working point. The linearized model, however, is not sufficient to be used as a prediction model for the attitude control of aerial robots that need to do challenging maneuvers. Although the LMPC is not an ideal controller for attitude control, it is sufficient for altitude control, because most aerial robots fly at low altitudes with limited variation in elevation; thus, the linearized model around that working altitude can cover the behavior of a flying robot [24,25].

Although PID controllers cannot fully cover fluctuations in system dynamics, they provide fast control response, which is a crucial ability for an aerial robot that requires agile attitude control responses to avoid obstacles and perform demanding maneuvers (e.g., delivery aerial robots in urban areas) [26]. In previous research efforts, many researchers have explored a variety of techniques to mitigate the problem in conventional standard PIDs. Several studies combined online tuning approaches with PID control to lessen the impact of changes in vehicle dynamics and disturbances [27]. A number of researchers opted for training neural networks to actively update the PID gain values of aerial drone controllers [28,29]. However, neural network training needs a large database of labeled data. Another widely used approach to actively tune the PID control is the fuzzy-logic-based auto-tuning algorithm [30–32]. The drawback of the fuzzy-logic-based tuning strategy is that the efficiency of algorithm is strongly reliant on the fuzzy rules and the inference system set by the designer. However, some fluctuations in the system dynamics may be unanticipated, resulting in the generation of inaccurate control gain coefficients by the fuzzy logic [33].

In addition to the aforementioned optimal control and active tuning approaches (for PIDs), Reinforcement Learning (RL) and Deep Reinforcement Learning (DRL) are other new approaches that have recently made their way into the field of aerial robots control. RL and DRL do not require an accurate model of the plant under control or human designed control rules, in contrast to fuzzy logic that depends on the expert's expertise. As a result, these machine learning approaches have attracted the attention of many academics working on the control of systems with uncertain dynamical models [34–36]. Unlike neural network-based control techniques, deep reinforcement learning does not require large labeled datasets. This is a major benefit, since labelling data for all critical scenarios become more and more expensive as the number of required data grows. Another important advantage of deep reinforcement learning for control is its capability to directly map image features to control states, thereby resolving the need for the implementation of complex state estimators and image processing algorithms in some special cases [37].

1.2. Related Works

Some researchers have already applied RL algorithms to control aerial robots. However, the bulk of studies focused on directly using RL algorithms to control aerial robots. As an example of these efforts, in a research study published in [38], a deep reinforcement learning algorithm was employed to control a fixed-wing aerial robot. The nonlinearities of the dynamics model, as well as the coupling effect between lateral and longitudinal control, were taken into account in the mentioned research study. The Proximal Policy Optimization (PPO) is the RL algorithm utilized in the study. Similarly, another research paper [39] employed PPO to regulate the attitude of a quadcopter. A research study presented in [40] employed a control system to achieve accurate autonomous driving of an aerial vehicle while landing on a platform. In the latter research paper, the Deep Q-learning Network (DQN) is the cornerstone of the controlling approach to mapping the images of poor quality to control states. Although the DQN method appears to be a promising approach for tackling vision-based control problems, it has significant drawbacks that limit its usage in more advanced vehicle control tasks that rely on image processing. As another example of this series of research efforts, a research study published in [41], leveraged an RL-based approach to control an aerial robot with the objective of capturing photographs of a person's front view, particularly his face. RL-based controllers also have found their ways into the world of morphing aerial robots. For instance, a work presented in [42] used a combination of the PPO algorithm with a PID controller to control a morphing aerial robot.

In contrast to the majority of the mentioned works that used RL algorithms to directly control aerial robots, some researchers opted not to use RL for that purpose. Instead, some attempts have been made to use the RL-based algorithm as a foundation for active tuning and state estimation mechanisms for other classical controllers. The drawback of a direct RL-based control algorithm is its slow response, compared with that of conventional PID controllers that provide very fast controlling responses [43]. In addition, direct deep reinforcement learning-based controlling approaches does not provide any analytical guarantees for the stable response and robustness in the control process, as unexplainable neural networks underpin its structure. However, RL has the potential to be used along with traditional control algorithms to provide adaptive and robust controllers.

As an example of the latter approach, the research study reported in [44] employed a fault-tolerant RL-based adaptive controller that combined an RL-based adaptive algorithm (in the study PPO) with a PID controller and an Unscented Kalman Filter (UKF) to develop a fault-tolerant RL-based adaptive controller. The proposed controlling strategy has employed a hybrid of parameter estimation and a deep reinforcement learning method. When the value of the parameters associated with faults affected the controller performance, the algorithm updated the PID controller. Although the findings of the study revealed a satisfactory control response for altitude control, the attitude control response did not provide clear superiority over earlier conventional controllers. Another research study [45] developed an adaptive neuro-fuzzy PID controller for nonlinear systems based on the Twin Delayed Deep Deterministic Policy Gradient (TD3) method. The observation of the environment is integrated with information from a multiple-input single-output (MISO) fuzzy inference system (FIS) and has a specifically defined fuzzy PID controller functioning as the actor in the TD3 method, which provides automated tuning of fuzzy PID controller gains.

1.3. Research Objectives

The majority of the aforementioned research studies focused solely on improving either attitude control performance or altitude controller response. In addition, it must be noted that some DRL algorithms are more efficient for attitude control (e.g., PPO), while another group of DRL algorithms (e.g., DDPG) shows better performance in improving trajectory tracking [46]. In order to compensate for fluctuations in the dynamics of the aerial robot, in the proposed controlling architecture, a reinforcement learning algorithm interacts with the system and learns adaptation policies for actively updating the gains of the controllers. To adjust the parameters of the attitude PID controller, the trained policy actively creates appropriate control gain values. Similarly, the scaling factor of the compensator is updated using actions generated by the RL agent. The aerial RL-based adaption algorithm is trained in a simulated environment in MATLAB software.

The rest of this paper is organized as follows: In Section 2, the dynamic model of a quadrotor is discussed. In Section 3, the control problem of the aerial robot is addressed, where the altitude and attitude control of the robot is discussed before introducing the proposed RL-based adaptive control framework. The applicability and efficacy of the proposed control strategy are evaluated in a simulated environment in Section 4. Finally, Section 5 summarizes the research findings.

2. Aerial Robot Dynamics

The aerial robot used in this research study is a quadcopter. Quadcopters are substantially underactuated, with six degrees of freedom (three translational and three rotational) and only four distinct inputs (rotor speeds). Rotational and translational motions are coupled to achieve six degrees of freedom. After accounting for the intricate aerodynamic effects, the resulting dynamics is highly nonlinear. As another property of quadcopters, it must be noted that, unlike conventional helicopters, the rotor blade pitch angle in a quadcopter does not need to be varied.

2.1. Quadcopter Coordinate Frames, Forces, and Torques

The reference coordinate frame and the coordinate frame of the vehicle body must be determined before building a mathematical model of the quadrotor, as shown in Figure 1. The ground and the reference coordinate frames are both tied to $\Re_E\left(O, \vec{I}, \vec{J}, \vec{K}\right)$. The $\Re_B\left(o, \vec{i}, \vec{j}, \vec{k}\right)$ is a coordinate frame that is attached to the body of the vehicle and has its center aligned with the center of mass of the robot.

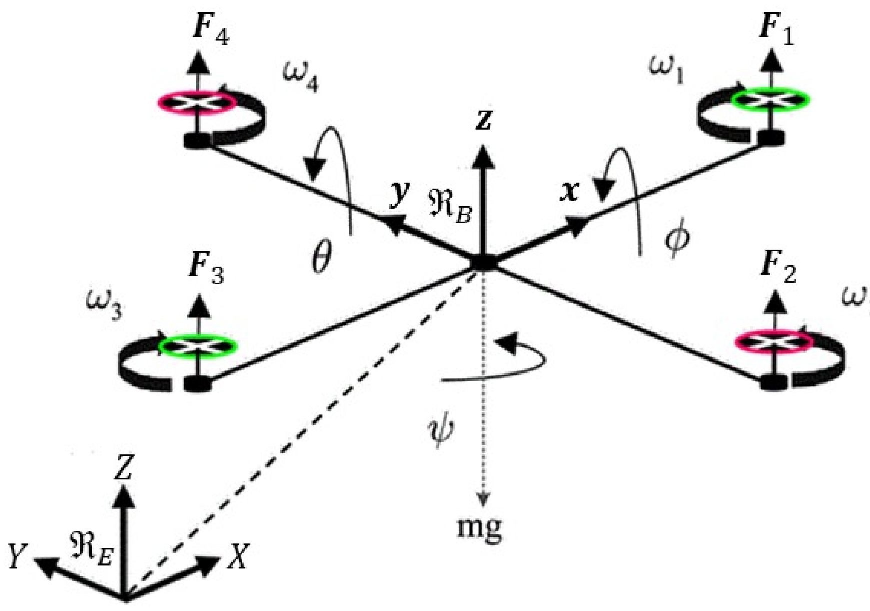

Figure 1. The coordinate frame of the quadcopter.

In this research study, the dynamical equations governing the quadcopter were derived from the text published in [47]. The following assumptions were taken into consideration in order to determine the examined equations of the motion of the system:

- The aerial robot consists of a stiff body with a symmetrical structure.
- The geometrical center of the robot is the same as its center of gravity and mass.
- The moment of inertia of the propellers has been overlooked.

The dynamical model of the system could be constructed by taking into account both the translational dynamic (Newton's second law) and the rotational dynamic (Euler's rotation equations).

2.2. Translational Dynamics

The following forces acted on the system being studied:

- The total weight of the vehicle, as expressed in Equation (1).
- The generated thrust of rotors, which can be calculated using Equation (2).
- As indicated in Equation (3), the drag force and air friction.

$$\boldsymbol{w} = \begin{bmatrix} 0 \\ 0 \\ -mg \end{bmatrix} \tag{1}$$

$$\boldsymbol{F}_t = \boldsymbol{R}\sum_{i=1}^4 F_i = b\sum_{i=1}^4 (\omega_i)^2 \begin{bmatrix} \sin\varphi\sin\psi + \sin\theta\cos\psi\cos\varphi \\ -\sin\varphi\cos\psi + \sin\psi\cos\varphi\sin\theta \\ \cos\theta\cos\varphi \end{bmatrix} \tag{2}$$

$$\boldsymbol{F}_d = \boldsymbol{C}_d\dot{\boldsymbol{\xi}} = -\begin{bmatrix} C_{dx} & 0 & 0 \\ 0 & C_{dy} & 0 \\ 0 & 0 & C_{dz} \end{bmatrix}\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix} = -\begin{bmatrix} C_{dx}\dot{x} \\ C_{dy}\dot{y} \\ C_{dz}\dot{z} \end{bmatrix} \tag{3}$$

$$\boldsymbol{F} = m\ddot{\boldsymbol{\xi}} = \boldsymbol{w} + \boldsymbol{F}_t + \boldsymbol{F}_d \tag{4}$$

In the equations, the gravity acceleration is denoted by g. In Equation (2), the Euler angles are represented by (φ, θ, ψ). The rotation transform matrix, the angular velocity of the ith propeller, and the thrust constant are represented by $\boldsymbol{R}$, ω_i, and b, respectively. In Equation (3), $\boldsymbol{C}_d$ is the matrix of translational drag coefficients. The position of the center of mass ($\boldsymbol{\xi}$) in the flat earth coordinate is defined as a 3 by 1 vector. The equation of motion that describes the translational motion of a quadcopter can be stated as follows, using Newton's second law:

$$\ddot{x} = \frac{1}{m}\left(b\sum_{i=1}^4 (\omega_i)^2(\sin\varphi\sin\psi + \cos\varphi\sin\theta\cos\psi) - C_{dx}\dot{x}\right) \tag{5}$$

$$\ddot{y} = \frac{1}{m}\left(b\sum_{i=1}^4 (\omega_i)^2(-\sin\varphi\cos\psi + \sin\psi\cos\varphi\sin\theta) - C_{dy}\dot{y}\right) \tag{6}$$

$$\ddot{z} = \frac{1}{m}\left(b\sum_{i=1}^4 (\omega_i)^2(\cos\theta\cos\varphi) - C_{dz}\dot{z} - gm\right) \tag{7}$$

2.3. Rotational Dynamics

A quadrotor is affected by roll, pitch, and yaw torques, as well as by an aerodynamic friction torque and the gyroscopic effect of the propeller. The torques are expressed as follows, in Equations (8)–(12):

$$\boldsymbol{\tau}_x = \begin{bmatrix} 0 \\ -l \\ 0 \end{bmatrix} \times \begin{bmatrix} 0 \\ 0 \\ F_2 \end{bmatrix} + \begin{bmatrix} 0 \\ l \\ 0 \end{bmatrix} \times \begin{bmatrix} 0 \\ 0 \\ F_4 \end{bmatrix} = \begin{bmatrix} lb\left({\omega_4}^2 - {\omega_2}^2\right) \\ 0 \\ 0 \end{bmatrix} \tag{8}$$

$$\boldsymbol{\tau}_y = \begin{bmatrix} l \\ 0 \\ 0 \end{bmatrix} \times \begin{bmatrix} 0 \\ 0 \\ F_1 \end{bmatrix} + \begin{bmatrix} -l \\ 0 \\ 0 \end{bmatrix} \times \begin{bmatrix} 0 \\ 0 \\ F_3 \end{bmatrix} = \begin{bmatrix} 0 \\ lb\left(\omega_3{}^2 - \omega_1{}^2\right) \\ 0 \end{bmatrix} \tag{9}$$

$$\boldsymbol{\tau}_z = \begin{bmatrix} 0 \\ 0 \\ d\sum_{i=1}^{4}(\omega_i)^2 \end{bmatrix} \tag{10}$$

$$\boldsymbol{\tau}_a = \boldsymbol{C}_a \begin{bmatrix} \dot{\psi}^2 \\ \dot{\varphi}^2 \\ \dot{\theta}^2 \end{bmatrix} = \begin{bmatrix} C_{az}\dot{\psi}^2 \\ C_{ax}\dot{\varphi}^2 \\ C_{ay}\dot{\theta}^2 \end{bmatrix} \tag{11}$$

$$\boldsymbol{\tau}_{gp} = J_r\Omega_r \begin{bmatrix} 0 \\ \dot{\theta} \\ -\dot{\varphi} \end{bmatrix} \tag{12}$$

where l is the distance between the motor axis and the center of mass of the quadcopter. In Equation (11), $\boldsymbol{C}_a$ is a 3 by 3 matrix of aerodynamic friction coefficients. In Equation (12), J_r and Ω_r, respectively, are the inertia and rotation velocity of rotors. Applying Euler's rotation equations yields the equations of motion (Equations (13)–(15)) that govern the rotating motion of the quadrotor. In the equations, $I_x,\ I_y,\ and\ I_z$ are moments of inertia along the $x, y,\ and\ z$ directions respectively:

$$\ddot{\varphi} = \frac{1}{I_x}\left(-C_{ax}\dot{\varphi}^2 - J_r\Omega_r\dot{\theta} - \left(I_z - I_y\right)\dot{\theta}\,\dot{\psi} + lb\left(\omega_4{}^2 - \omega_2{}^2\right)\right) \tag{13}$$

$$\ddot{\theta} = \frac{1}{I_y}\left(-C_{ay}\dot{\theta}^2 - J_r\Omega_r\dot{\varphi} - (I_x - I_z)\dot{\varphi}\dot{\psi} + lb\left(\omega_3{}^2 - \omega_1{}^2\right)\right) \tag{14}$$

$$\ddot{\psi} = \frac{1}{I_z}\left(-C_{ay}\dot{\theta}^2 - J_r\Omega_r\dot{\varphi} - (I_x - I_z)\dot{\varphi}\dot{\psi} + \sum_{i=1}^{4}(-1)^{i+1}\omega_i{}^2\right) \tag{15}$$

2.4. Dynamics Model of the Quadcopter

Having considered both translational and rotational dynamics, the entire dynamic model of the quadcopter could be stated as follows:

$$\ddot{x} = \frac{1}{m}\left(u_1u_x - C_{dx}\dot{x}\right) \tag{16}$$

$$\ddot{y} = \frac{1}{m}\left(u_1u_y - C_{dy}\dot{y}\right) \tag{17}$$

$$\ddot{z} = \frac{1}{m}\left(u_1(\cos\theta\cos\varphi) - C_{dz}\dot{z} - gm\right) \tag{18}$$

$$\ddot{\varphi} = \frac{1}{I_x}\left(-C_{ax}\dot{\varphi}^2 - J_r\Omega_r\dot{\theta} - \left(I_z - I_y\right)\dot{\theta}\,\dot{\psi} + u_2\right) \tag{19}$$

$$\ddot{\theta} = \frac{1}{I_y}\left(-C_{ay}\dot{\theta}^2 - J_r\Omega_r\dot{\varphi} - (I_x - I_z)\dot{\varphi}\dot{\psi} + u_3\right) \tag{20}$$

$$\ddot{\psi} = \frac{1}{I_z}\left(-C_{ay}\dot{\theta}^2 - J_r\Omega_r\dot{\varphi} - (I_x - I_z)\dot{\varphi}\dot{\psi} + u_4\right) \tag{21}$$

where u is a vector expressed as follows (in the equations, d is the drag coefficient):

$$u_x = (\sin\varphi\sin\psi + \cos\varphi\sin\theta\cos\psi) \tag{22}$$

$$u_y = (-\sin\varphi\cos\psi + \sin\psi\cos\varphi\sin\theta) \tag{23}$$

$$\begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix} = \begin{bmatrix} b & b & b & b \\ 0 & -lb & 0 & lb \\ -lb & 0 & lb & 0 \\ d & -d & d & -d \end{bmatrix} \begin{bmatrix} {\omega_1}^2 \\ {\omega_2}^2 \\ {\omega_3}^2 \\ {\omega_4}^2 \end{bmatrix} \tag{24}$$

3. Quadcopter Control

3.1. Controller Framework

A quadcopter is an underactuated system, which means that six degrees of freedom in space are controlled by just four motors. Hence, controllers in such vehicles must be designed for a subset of four degrees of freedom. Furthermore, the fact must be taken into account that the control of the x and y positions in space is influenced by changes in the pitch and roll angles. Having considered the aforementioned relationships, the control of a quadrotor is normally designed for two independent subsets of coordinates. The necessity for a swashplate mechanism is eliminated with four separate rotors. The swashplate mechanism was necessary to give the helicopter more degrees of freedom, but the same level of control can be achieved by simply adding two more rotors, as implemented in the structure of quadcopters. Despite the fact that the command is for three position coordinates (x, y, z) plus yaw angle, the control algorithm employs both roll and pitch orientation controllers. In the inertial coordinate system, the control signals of three position controllers define a force vector (thrust). The setpoints $(u1_x, u1_y)$ transmitted to the roll and pitch controls are considered as the orientation of the vector. The stated architecture, as well as the elements of our proposed controllers, are depicted in Figure 2, which will be explored in the next sections.

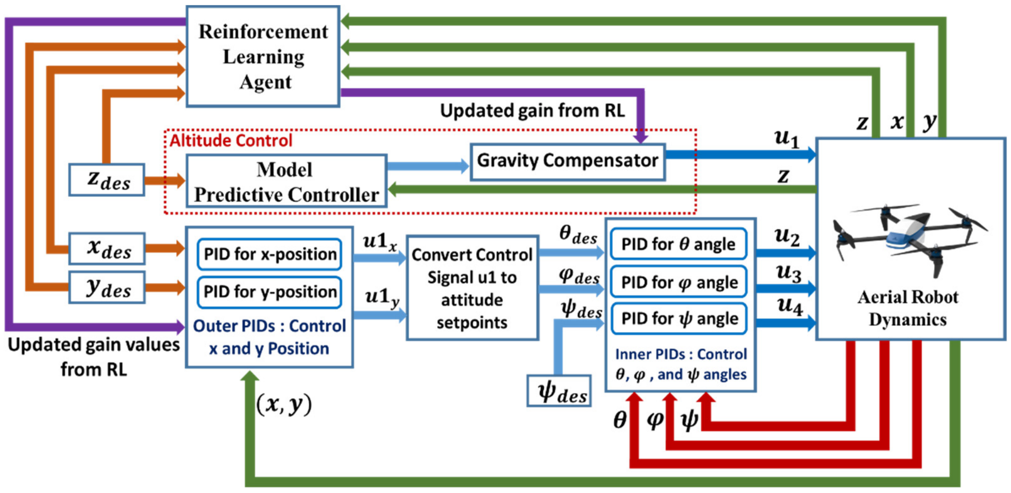

Figure 2. The elements of the proposed control architecture of the quadcopter.

The altitude controller and the attitude controller are the two main parts of the control architecture. The altitude controller, as shown in Figure 2, maintains the altitude of the aerial robot at the required level. In most commercial aerial robots, the altitude controller is a PID controller with fixed control gain values. In this research, a proposed control architecture consisting of a MPC controller and a gravity compensator is proposed as a replacement for conventional PID controllers. The scaling factor of the compensator is adaptively adjusted during the operation of the robot, using actions generated by the reinforcement learning agent. The robot attitude controller is the second major controlling component of the system. The controller is made up of two distinct PID controller blocks. The difference between the desired x and y position and the actual x and y location in space is measured and defined as the position error in 2D. The x-position and y-position PID controllers in the outer loop were designed to minimize the error. The control commands of the position controllers are transformed to appropriate roll and pitch setpoints. The inner loop PID controllers use the resulting roll and pitch setpoints as reference inputs.

The control gain values of the inner loop PID controllers are constant in our proposed control architecture, whereas the control gain values of the outer loop PID controllers are adaptively adjusted using trained policy from the RL-based adaptation algorithm.

3.2. Attitude Control

To control the attitude of the robot, an architecture comprising of RL-based adaptive controllers is proposed in this study. The outer loop PID controllers were designed to generate the $u1_x$ and $u1_y$ virtual control signals as described in Equations (25)–(28). Equation (29) is used to convert the control commands from outer loop PIDs to the necessary roll and pitch reference values for the inner-loop PID controllers.

$$e_x = x_{des} - x \tag{25}$$

$$e_y = y_{des} - y \tag{26}$$

$$u1_x = (Kp_x \cdot e_x) + (Kd_x \cdot \dot{e}_x) + \left(Ki_x \int e_x \, dt\right) \tag{27}$$

$$u1_y = (Kp_y \cdot e_y) + (Kd_y \cdot \dot{e}_y) + \left(Ki_y \int e_y \, dt\right) \tag{28}$$

$$\begin{bmatrix} \varphi_d \\ \theta_d \end{bmatrix} = \frac{1}{g} \begin{bmatrix} \sin(\psi_{des}) - \cos(\psi_{des}) \\ \cos(\psi_{des}) - \sin(\psi_{des}) \end{bmatrix} \begin{bmatrix} u1_x \\ u1_y \end{bmatrix} \tag{29}$$

As indicated in Equations (30)–(32), three PID controllers were implemented in the inner loop PID control block to provide manipulated variables for robot attitude control.

$$u_2 = Kp_2(\varphi_d - \varphi) + Kd_2(\dot{\varphi}_d - \dot{\varphi}) + Ki_2 \int (\varphi_d - \varphi) \, dt \tag{30}$$

$$u_3 = Kp_3(\theta_d - \theta) + Kd_3\left(\dot{\theta}_d - \dot{\theta}\right) + Ki_3 \int (\theta_d - \theta) \, dt \tag{31}$$

$$u_4 = Kp_4(\psi_d - \psi) + Kd_4\left(\dot{\psi}_d - \dot{\psi}\right) + Ki_4 \int (\psi_d - \psi) \, dt \tag{32}$$

The optimal control gains for the inner loop PID controllers were obtained, based on several trials and errors. The obtained gain coefficients are listed in Table 1. The control gains of outer loop PID controllers are actively estimated and adjusted by the RL agent.

Table 1. Control gain values of the inner-loop PID controllers.

PID Control	***Kp***	***Ki***	***Kd***
Roll (φ)	0.021	0.011	0.003
Pitch (θ)	0.014	0.03	0.001
Yaw (ψ)	0.002	0.07	0.013

3.3. Altitude Control

In this study, the proposed altitude controller utilizes a linear model predictive controller and a gravity compensator in its controlling architecture. The gravity compensator is responsible for alleviating the effect of forces that arise from fluctuations in the weight of the robot. The scaling factor of the compensator is adaptively updated using the RL policy. The proposed algorithm aimed to mitigate the impact of disturbances arising from changes in the weight of the robot on the performance of the aerial robot in trajectory tracking and altitude stabilization. The MPC is based on an iterative, finite-horizon robot model optimization. The present states of the quadcopter are sampled at time t, and a cost-minimizing control strategy for a relatively short time horizon in the future $[\mathrm{t}, \mathrm{t} + \mathrm{T}]$ is computed (using a numerical minimization technique). At each control interval, model predictive control solves an optimization problem, a quadratic program (QP). Until the next control interval, the solution generated a sequence of manipulated variables to be applied to the robot. A series of online optimizations are run to estimate possible state trajectories that would arise from the present states. Furthermore, the solution identifies a cost-minimizing control strategy (by the solution of Euler–Lagrange equations) from t until time ran out at t + T. Although the MPC computes a series of manipulated variables, only the first step of the computed control strategy is applied to the quadcopter, after which the updated states of the robot are sampled again and the computations are repeated using the updated states, resulting in computation of fresh control inputs and a new anticipated state route. Figure 3 shows the relationship between the prediction horizon and generated control inputs.

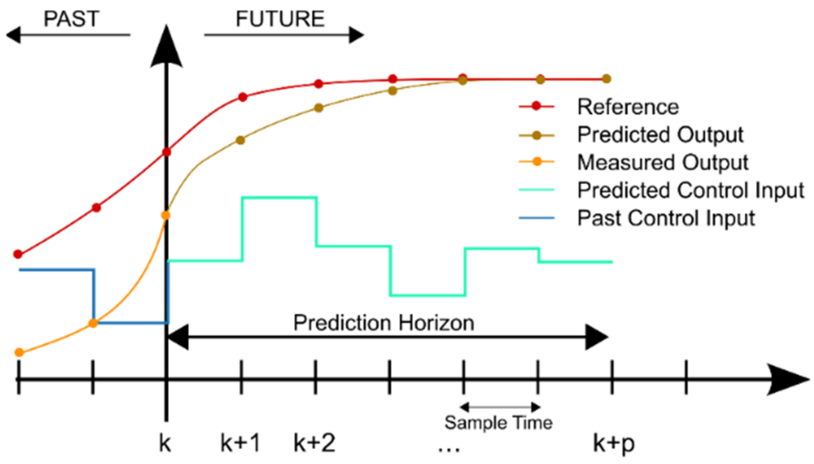

Figure 3. Prediction horizon and predicted control inputs in MPC.

Model predictive control is considered as a multivariable controlling algorithm incorporating the following components:

- A dynamics model of the system under control.
- A cost function J.
- An optimization mechanism. The optimal manipulated variable (u_{MPC}) is computed by minimizing the cost function J using the optimization algorithm.

A typical cost function in the MPC algorithm is made up of four terms, each of which focuses on a different element of controller performance (k represents the current control interval):

$$J(\boldsymbol{d}_k) = J_{RT}(\boldsymbol{d}_k) + J_u(\boldsymbol{d}_k) + J_{\Delta u}(\boldsymbol{d}_k) \tag{33}$$

$$\boldsymbol{d}_k = [u_{MPC}(k|k)\ u_{MPC}(k+1|k) \ldots\ u_{MPC}(k+p-1|k)] \tag{34}$$

where $\boldsymbol{d}_k$ signifies optimal control inputs that are obtained by solving a quadratic programming (QP) problem, as indicated in Equation (34). In Equation (33), $J_{RT}(\boldsymbol{d}_k)$ is reference tracking cost. Here, $J_u(\boldsymbol{d}_k)$ and $J_{\Delta u}(\boldsymbol{d}_k)$ are representations of manipulated variable tracking and manipulated variable move suppression, respectively. The last cost term in Equation (33), $J_{\Delta u}(\boldsymbol{d}_k)$, decreases the control effort, thereby reducing the energy consumption of the actuators (e.g., the dc motors of the quadrotor). The MPC cost function can be formulated as follow:

$$\begin{aligned} J(\boldsymbol{d}_k) = \textstyle\sum_{i=0}^{p-1} [\boldsymbol{e}_{\boldsymbol{RT}}^{\boldsymbol{T}}(k+i)\boldsymbol{Q}\boldsymbol{e}_{\boldsymbol{RT}}(k+i)] + [\boldsymbol{e}_{\boldsymbol{u}}^{\boldsymbol{T}}(k+i)\boldsymbol{R}\boldsymbol{e}_{\boldsymbol{u}}(k+i)] \\ + \left[\Delta \boldsymbol{u}^{\boldsymbol{T}}(k+i)\boldsymbol{R}_{\Delta \boldsymbol{u}}\Delta \boldsymbol{u}(k+i)\right] \end{aligned} \tag{35}$$

In Equation (35), $\boldsymbol{Q}$ is a ($n_{RT} \times n_{RT}$) weight matrix (n_{RT} is the number of plant output variables). Here, $\boldsymbol{R}$, and $\boldsymbol{R}_{\Delta \boldsymbol{u}}$ ($n_{\boldsymbol{u}} \times n_{\boldsymbol{u}}$) are positive-semi-definite weight matrices ($n_{\boldsymbol{u}}$ represents the number of manipulated variables). In the aforementioned cost function, p is the prediction horizon, which can be adjusted according to the controller performance and the processing power of the hardware. In Equation (35), $\boldsymbol{e_{RT}}$, $\boldsymbol{e_u}$, and $\Delta\boldsymbol{u}$ can be computed, using Equations (36)–(38).

$$\boldsymbol{e_{RT}}(i+k) = \boldsymbol{r}(k+i+1|k) - \boldsymbol{y}(k+i+1|k) \tag{36}$$

$$\boldsymbol{e_u}(i+k) = \boldsymbol{u_{des}}(k+i|k) - \boldsymbol{u_{MPC}}(k+i|k) \tag{37}$$

$$\Delta\boldsymbol{u}(k+i) = \boldsymbol{u_{MPC}}(k+i|k) - \boldsymbol{u_{MPC}}(k+i-1|k) \tag{38}$$

The reference value (or reference values) given to the controller at the ith prediction horizon step is specified as $\boldsymbol{r}(k+i|k)$. Similarly, the value (or values) of n_{RT} outputs variables of the plant, sampled at the ith prediction horizon step, is defined as $\boldsymbol{y}(k+i|k)$. In the equation, $\boldsymbol{u}_{des}(k+i|k)$ reflects the value (or values) of $n_{\boldsymbol{u}}$ desired control inputs corresponding to $\boldsymbol{u}_{MPC}(k+i|k)$. In the proposed MPC control architecture, in this paper, there is one manipulated variable, $u_{MPC}(k+i|k)$. In addition, in this paper, $y(k+i|k)$ is the altitude (z position) of the robot, while $r(k+i|k)$ is the desired altitude for the robot. In order to reduce the controller effort and alleviate the effects of arising fluctuations in the weight of the aerial robot, this study proposed to use a gravity compensator after the MPC controller. The final altitude control input u_1 is defined as:

$$u_1 = u_{MPC}(k|k) - g(nominal\ mass\ of\ the\ robot + a_1) \tag{39}$$

In Equation (39), a_1 is the scaling factor of the compensator that is actively estimated by the RL policy. In order to evaluate the performance of the proposed control architecture in controlling an aerial robot, a dynamics model of a commercial aerial robot named Parrot was chosen and implemented in the simulator environment as the aerial robot under control. The dynamics model specifications of the employed Parrot quadcopter are listed in Table 2.

Table 2. The Parrot aerial robot model specifications [48].

Specification	Parameter	Unit	Value
Drone Mass	m	kg	0.063
Lateral Moment Arm	l	m	0.0624
Thrust Coefficient	b	$\mathrm{N{\cdot}s^2}$	0.0107
Drag Coefficient	d	$\mathrm{N{\cdot}m{\cdot}s^2}$	0.7826×10^{-3}
Rolling Moment of Inertia	Ix	$\mathrm{Kg{\cdot}m^2}$	5.82857×10^{-5}
Pitching Moment of Inertia	Iy	$\mathrm{Kg{\cdot}m^2}$	7.16914×10^{-5}
Yawing Moment of Inertia	Iz	$\mathrm{Kg{\cdot}m^2}$	0.0001
Rotor Moment of Inertia	Ir	$\mathrm{Kg{\cdot}m^2}$	0.1021×10^{-6}

This research study employs a linear model predictive control. Hence, a linear state-space model of the quadcopter is required. Equation (40) describes the standard form of a linear time-invariant (LTI) state-space model, which has $\boldsymbol{p}$ inputs, $\boldsymbol{q}$ outputs, and $\boldsymbol{n}$ state variables.

$$\begin{aligned} \dot{\boldsymbol{x}} &= \boldsymbol{Ax} + \boldsymbol{Bu} \\ \boldsymbol{y} &= \boldsymbol{Cx} + \boldsymbol{Du} \end{aligned} \tag{40}$$

where $\boldsymbol{x}$, $\boldsymbol{y}$, and $\boldsymbol{u}$ are the state vector, the output vector, and the input vector, respectively. In the equation, $\boldsymbol{A}$, $\boldsymbol{B}$, and $\boldsymbol{C}$ are $(n \times n)$, the state matrix; $(n \times p)$, the input matrix; and $(q \times n)$, the output matrix, respectively. It should be noted that $\boldsymbol{D}$ is a $(q \times p)$ feedforward matrix. The $\boldsymbol{D}$ matrix is zero in this study, as there is no direct feedthrough. The linear model was computed around the operational point in the state-space model using the MPC Designer app in the Model Predictive Control Toolbox. The toolbox included in the MATLAB software provides control blocks for developing not only linear model predictive control but also nonlinear and adaptive model predictive controllers. In addition, the MPC Designer app included in the software package facilitates the design of an LMPC by automating the processes for plant linearization and tuning the controller parameters. The state-space parameters of the linearized plant can be found in the MPC object (MPCobj) generated by the MPC Designer app. The Simulink MPC Designer linearizes each block in the model independently, then combines the outputs of the individual linearized models to produce the linearized model of the whole plant. Table 3 presents the resulting values for the state-space model generated by the app.

Table 3. The parameters of the state space model.

Parameter	Value
$\boldsymbol{A}$	$\begin{bmatrix} 1 & 0 \\ 1.985 \times 10^{-4} & 1 \end{bmatrix}$
$\boldsymbol{B}$	$\begin{bmatrix} 7.94 \times 10^{-3} \\ 1.985 \times 10^{-4} \end{bmatrix}$
$\boldsymbol{C}$	$\begin{bmatrix} 0 & 1 \end{bmatrix}$
$\boldsymbol{D}$	0

3.4. Deep Reinforcement Learning for Online Parameter Estimation and Tuning

The proposed parameter estimation approach in the study utilizes a DRL algorithm to actively estimate and adjust the parameters both in PID controllers and the compensator. A reinforcement learning agent was developed in the Simulink environment to construct an adaption topology for actively estimating the tuning parameters of the designed controllers. In the first stage, the reinforcement learning algorithm interacted with the dynamics model of the robot (in the simulator environment) to learn the appropriate tuning rules. During the operation time of the robot, the trained policy is used to actively adjust the gain values of the controllers. In this study, the RL algorithm is the Deep Deterministic Policy Gradient (DDPG). The DDPG is an approach that combines the idea from DPG with the Deep Q-learning Network (DQN) to create an off-policy algorithm that is able to work with continuous action space [49]. It aims to generate the optimal action policy for the agent to maximize rewards while fulfilling its objectives [50]. The DDPG algorithm can work over continuous action spaces, which is a significant challenge for traditional RL approaches such as Q-learning. The DDPG algorithm is a hybrid technique that incorporates both the policy gradient and the value function. The architecture of DDPG is based on the actor-critic framework. In the algorithm, the actor refers to the policy function, whereas the critic refers to the value function Q. The critic network evaluates the actions of the actor based on the rewards and the subsequent state resulting from the environment. The role of the critic is to adjust the weights of the actor network so that the future actions of the actor result in the highest potential cumulative reward. The goal in the DDPG algorithm is to learn a parameterized deterministic policy $\mu_\theta(s)$, such that the obtained optimal policy maximizes the expected reward over all states reachable by the policy:

$$J(\theta) = \mathbb{E}_{\boldsymbol{s}\sim\rho_\mu}[R(s, \mu_\theta(s))] \tag{41}$$

where ρ_μ is the distribution of all states reachable by the policy. It is proven that maximizing the returns or true Q-value of all actions leads to the same optimal policy. This is an idea introduced in dynamic programing, where policy evaluation first finds the true Q-value of all state-action pairs, and policy improvements change the policy by selecting the action(s) with the maximal Q-value:

$$a_t^* = \text{argmax}_a Q_\theta(s_t, a) \tag{42}$$

in Equation (42), a_t^* is the action(s) with maximal Q-value. The action is expected to generate the maximum expected reward. In the continuous space, the gradient of objective function can be considered to be the same a as the gradient of the Q-value. If we have an estimate $Q^{\mu}(s,a)$ of all the value of any action (a), changing the policy $\mu_{\theta}(s)$ in the direction of $\nabla_{\theta}Q^{\mu}(s,a)$ leads to an action with higher Q-value and associated return:

$$\nabla_{\theta}J(\theta) = \mathbb{E}_{s\sim\rho_{\mu}}\left[\nabla_{\theta}Q^{\mu}(s,a)|_{a=\mu_{\theta}(s)}\right] \tag{43}$$

In Equation (43), gradient with regard to action (a) of the Q-value is taken. We can expand the above equation by applying the chain rule.

$$\begin{gathered}\frac{\partial Q(s,a)}{\partial\theta} = \frac{\partial Q(s,a)}{\partial a} \times \frac{\partial a}{\partial\theta} \\ \nabla_{\theta}J(\theta) = \mathbb{E}_{s\sim\rho_{\mu}}\left[\nabla_{\theta}\mu_{\theta}(s) \times \nabla_{a}Q^{\mu}(s,a)|_{a=\mu_{\theta}(s)}\right]\end{gathered} \tag{44}$$

To obtain an unbiased estimate of the Q-value of any action and compute its gradient, it is possible to calculate a function approximator $Q_{\varphi}(s,a)$, as long as it is compatible, and to minimize the quadratic error with the true Q-values:

$$\nabla_{\theta}J(\theta) = \mathbb{E}_{s\sim\rho_{\mu}}\left[\nabla_{\theta}\mu_{\theta}(s) \times \nabla_{a}Q_{\varphi}(s,a)|_{a=\mu_{\theta}(s)}\right] \tag{45}$$

$$J(\varphi) = \mathbb{E}_{s\sim\rho_{\mu}}\left[\left(Q^{\mu}(s,\mu_{\theta}(s)) - Q_{\varphi}(s,\mu_{\theta}(s))\right)^{2}\right] \tag{46}$$

The goal of the DDPG algorithm was to extend the DPG to incorporate non-linear function approximators. The objective was fulfilled by combining DQN and DPG concepts to create an algorithm working over continuous space. Furthermore, the following elements were added to the base DPG algorithm.

- An experience reply memory to store past transitions and learn off-policy.
- Target networks to stabilize learning.

DDPG updates the parameters of target networks after each update of the trained network using a sliding average (soft update) for both the actor and the critic:

$$\theta^{Q_{target}} \leftarrow \tau\theta^{Q} + (1-\tau)\theta^{Q_{target}} \tag{47}$$

$$\theta^{\mu_{target}} \leftarrow \tau\theta^{\mu} + (1-\tau)\theta^{\mu_{target}} \tag{48}$$

where τ is a hyperprameter between 0 and 1. The modified update rule guarantees that the target networks are always lagging behind the trained networks, providing more stability to the learning of Q-values. The key idea borrowed from DPG is the policy gradient for the actor. The critic is learned using regular Q-learning and target networks:

$$J(\varphi) = \mathbb{E}_{s\sim\rho_{\mu}}\left[\left(r(s,a,s') + \gamma Q_{target}(s',\mu_{\theta'}(s')) - Q_{\varphi}(s,a)\right)^{2}\right] \tag{49}$$

where $Q_{\varphi'}(s',\mu_{\theta'}(s'))$ is the value of the action that is estimated to return the largest total future reward, based on all possible actions that can be made in the next state. In the Equation (49), γ is the discount factor. Noise is added to improve exploration:

$$a_t = \mu_{\theta}(s_t) + \xi_{noise} \tag{50}$$

The additive noise is an Ornstein–Uhlenbeck process that generate temporally correlated noise with zero mean. The target value can be computed using the target network:

$$y_{RLi} = r_i + \gamma Q_{target}(s_{i+1}, \mu_{target}(s_{i+1}|\theta^{\mu_{target}})\big|\theta^{Q_{target}}) \tag{51}$$

The following loss function is minimized, resulting in an update in the critic. A sampling policy gradient can also be applied to update the actor:

$$L = \left(\frac{1}{\text{N}}\right)\sum_{i}\left(y_{RLi} - Q\left(s_i, a_i\middle|\theta^Q\right)\right)^2 \tag{52}$$

$$\nabla_{\theta^\mu} J \approx \frac{1}{N}\sum_{i}\nabla_a Q\left(s, a\middle|\theta^Q\right)\Big|_{s=s_i\ ,\ a=\mu(s_i)}\nabla_{\theta^\mu}\mu(s|\theta^\mu)\Big|_{s_i}\nabla_a \tag{53}$$

Figure 4 depicts the aforementioned learning process in the DDPG algorithm. The architecture is comprised of two actor-critic networks. The target networks in the architecture stabilized the learning procedure. The noise improves the exploration of the reachable control states. The proposed RL-based control algorithm (Algorithm 1) is described in the following pseudocode.

Algorithm 1. The proposed algorithm.

```
1:  Initial policy network $\mu$ and critic network $Q$ with weights $\theta^{\mu} and\ \theta^{Q}$ respectively.
2:  Set target policy network $\mu_{target}$ and target critic network $Q_{target}$ with weights $\theta^{\mu_{target}} and\ \theta^{Q_{target}}$
3:  Set target parameters weights equal to main parameters weights:
                $\theta^{\mu_{target}} \leftarrow \theta^{\mu}\ ,\ \theta^{Q_{target}} \leftarrow \theta^{Q}$
4:  for episode = 1, M do
5:      Initialize a random process noise $\mathcal{N}$ for action
        exploration.
6:      Receive initial observation state $s_1$.
7:      for t = 1, T do
8:          Select actions $a = \mu_{\theta_i} + \xi_{noise}$ where $\xi_{noise} \sim \mathcal{N}$
9:          Observe a vector of states $s$
            Apply actions ($a_2\ to\ a_7$) to outer loop PID controllers as follow:
10:         $u1_x = (a_2 \cdot e_x) + (a_3 \cdot \dot{e}_x) + (a_4 \int e_x dt)$
            $u1_y = (a_5 \cdot e_y) + (a_6 \cdot \dot{e}_y) + (a_7 \int e_y dt)$
11:         Use the updated $u1_x$ and $u1_y$ to generate $\theta_{des}\ , \varphi_{des}$
12:         Use $\theta_{des}\ , \varphi_{des}$ as setpoints for inner loop PIDs and generate $u_2$,$u_3$ and $u_4$.
            Compute $u_{MPC}$ by minimizing the MPC cost function.
13:          $\min\ \{J(d_k) = \sum_{i=0}^{p-1} e_{RT}^T(k+i) Q_{MPC} e_{RT}(k+i) + e_u^T(k+i) R e_u(k+i) +$
                  $\Delta u^T(k+i) R_{\Delta u} \Delta u(k+i)\}$
            Compute $u_1$ using the generated namipulated variable from MPC and the
14:         scalling factor of the compensator
            $u_1 = u_{MPC}(k+i|k) - g(nominal\ mass\ of\ the\ robot + a_1)$
15:         Apply control inputs $u_1$, $u_2$, $u_3$ and $u_4$ to the drone dynamics model.
16:         Observe the next vector of states $s_{t+1}$, and the next reward $r$
17:         Store $(s, a, r, s_{t+1}\ )$ in reply buffer $\boldsymbol{D}$.
18:         Randomly sample a minibatch of $N$ transitions $(s_i, a_i, r, s_{i+1})$ from $\boldsymbol{D}$.
19:         Compute targets: $y_{RLi} = r_i + \gamma Q_{target}(s_{i+1}, \mu_{target}(s_{i+1}\big|\theta^{\mu_{target}})\big|\theta^{Q_{target}})$
20:         Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i \left(y_{RLi} - Q(s_i, a_i|\theta^Q)\right)^2$
            Update the actor policy using a sampled policy gradient:
21:         $\nabla_{\theta^\mu} J \approx \frac{1}{N}\sum_i \nabla_a Q(s, a|\theta^Q)\Big|_{s=s_i\ ,\ a=\mu(s_i)} \nabla_{\theta^\mu}\mu(s|\theta^\mu)\Big|_{s_i}$
22:         Update the target networks:
23:         $\theta^{Q_{target}} \leftarrow \tau\theta^{Q} + (1-\tau)\theta^{Q_{target}}$,
            $\theta^{\mu_{target}} \leftarrow \tau\theta^{\mu} + (1-\tau)\theta^{\mu_{target}}$
24:     end for
25: end for
```

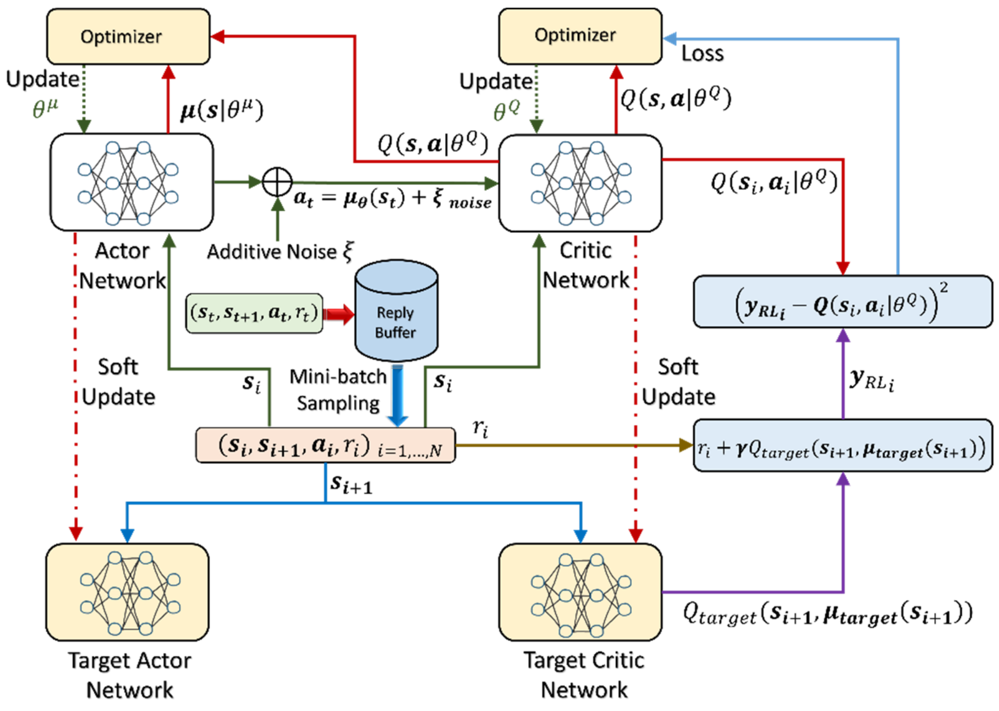

Figure 4. The architecture and learning process of deep deterministic policy gradient.

The actor outputs are actions ($a = \mu_\theta(s) + \xi_{noise}$) selected from a continuous action space by considering the current state of the environment. In this study, the action is comprised of a vector of actions $\boldsymbol{a} = [a_1\ a_2\ a_3\ a_4\ a_5\ a_6\ a_7]$. Here, a_2 to a_7 are used to actively estimates and tune the gain coefficients of the outer loop PID controllers for x and y positions, which are two controllers located in outer loop PID control block. The action (a_1) generates an updated estimation of the proper scaling factor for updating the compensator. The defined states were errors (e_x, e_y, e_z) and $(\int e_x,\ \int e_y,\ \int e_z)$. The Q function is computed using the rewards from the environments. The notion is that the best policy can be found by maximizing the true Q-value of all actions. The architecture of the proposed adaptive RL-based controller is shown in Figure 5.

To find the optimal architecture for the actor-critic network, many simulations were run with different numbers of layers and neuron counts in neural networks. According to the simulations findings, it was observed that increasing the size of hidden layers encourages polynomial exploration, whereas increasing the number of layers promotes exponential exploration. The number of layers in the actor network varied from 1 to 5 for the purpose of the experiment. Increasing the size of the actor network layers to a value of more than 400 neurons per layer resulted in over-parameterization and oscillation of the discounted long-term reward. Over-reducing the number of layers, on the other hand, resulted in poor performance. According to the simulation results, it was concluded that at least 130 neurons in each layer of the actor and the critic are necessary to learn the policy effectively, with 200 being the ideal number. The architecture of the designed actor and critic neural networks is shown in Figure 6.

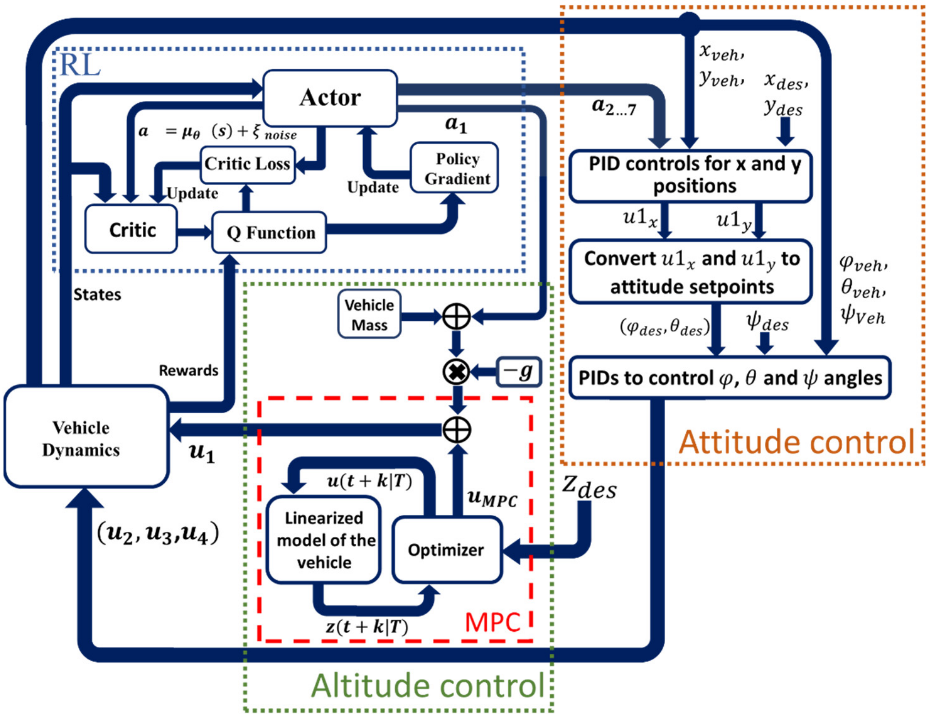
RL
Actor
$a = \mu_\theta(s) + \xi_{noise}$
a_1
Critic Loss
Policy Gradient
Update
Update
Critic
Q Function
States
Rewards
Vehicle Dynamics
u_1
(u_2, u_3, u_4)
Vehicle Mass
$-g$
$u(t+k|T)$
u_{MPC}
Linearized model of the vehicle
Optimizer
$z(t+k|T)$
MPC
Altitude control
z_{des}
$a_{2\ldots7}$
x_{veh}, y_{veh}, x_{des}, y_{des}
PID controls for x and y positions
$u1_x$ $u1_y$
Convert $u1_x$ and $u1_y$ to attitude setpoints
$(\varphi_{des}, \theta_{des})$ ψ_{des}
φ_{veh}, θ_{veh}, ψ_{Veh}
PIDs to control φ, θ and ψ angles
Attitude control

Figure 5. The architecture of the proposed adaptive RL-based controller.

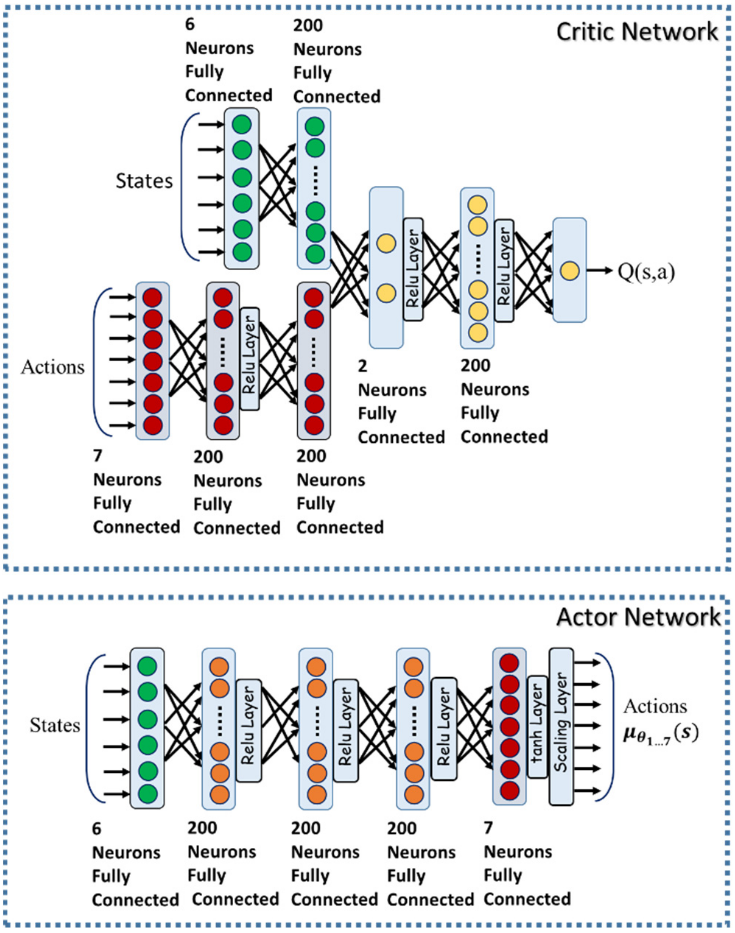
Critic Network
6 Neurons Fully Connected
200 Neurons Fully Connected
States
Actions
7 Neurons Fully Connected
200 Neurons Fully Connected
200 Neurons Fully Connected
Relu Layer
2 Neurons Fully Connected
200 Neurons Fully Connected
Q(s,a)
Actor Network
States
6 Neurons Fully Connected
200 Neurons Fully Connected
200 Neurons Fully Connected
200 Neurons Fully Connected
7 Neurons Fully Connected
tanh Layer
Scaling Layer
Actions
$\mu_{\theta_{1\ldots7}}(s)$

Figure 6. The architecture of actor and critic neural networks.

4. Simulations

As mentioned in Section 3, a dynamics model of a commercial aerial robot (Parrot) was selected and implemented in the simulator environment as an aerial robot under control to test the effectiveness of the proposed control architecture in controlling an aerial robot. The AR Drone 2.0, the Rolling Spider, and Mambo are all Parrot products that can be programmed using MATLAB [51]. The AR Drone 2.0 toolbox was developed and distributed by researchers. In addition, a MATLAB Simulink Support Package for Rolling Spider is available as an add-on development package based on MIT's Aerospace Blockset supporting simulation, hardware code generation, and interface for Parrot Minidrones [52]. The Parrot Mambo (6-DOF small quadcopter) comes with ultrasonic, accelerometer, gyroscope, air pressure, and down-facing camera sensors. It allows algorithms to be applied to the robot via a Bluetooth link over a Personal Area Network (PAN).The Simulink Support Package for Parrot Minidrones [53], which is developed using the UAV toolbox from Mathworks [54], was utilized in this research study to simulate the quadrotor. The aerodynamics effect was deemed minor for the purpose of simplicity, hence the block associated with the aerodynamics effect was deactivated throughout the simulations. Furthermore, during the simulations, parameters of the environment (such as air pressure at various elevations) were also assumed to be constant. A UAV waypoint follower block (included in the UAV toolbox) was employed to generate several successive waypoints. In the Simulink environment, two experiments were carried out to evaluate the performance of the proposed control architecture, compared to that of typical PID controllers.

The experiments aimed at evaluating the effectiveness of the controller in controlling the aerial robot in the presence of weight disturbances affecting the dynamics model of the robot. When a payload is attached to a quadrotor, three main changes may happen. The overall mass of the system increases. The fluctuations in the mass of the robot can also shift the gravitational center, and therefore the inertia. In the experiments, the payload was considered to be an isotropic symmetric stiff solid mass connected to the aerial robot, rather than being suspended. Another assumption was that the physical dimensions of the attached mass were smaller than the dimensions of the quadrotor. The distance between the center of gravity of the robot and the center of gravity of the load is associated with the shape and the weight distribution of the connector, as well as with the shape and the weight distribution of the load. In our experiments, this distance was assumed to be zero. As a result, the effects of changes in inertia and gravity were deemed insignificant. Therefore, the focus of our article was solely on analyzing the effects of mass variation on the performance of the controller. When the robot is executing a logistic task, the mass could be in two stable states: one before the load comes into contact with the quadrotor, and the other when the payload mass is integrated into the system. Depending on the gripping technique and the qualities of the surface of the load, there may be many profiles of mass fluctuation between these two states. We assumed that there was no gripping and the mass was connected to the robot body directly. Another assumption was that the load had a non-deformable surface; therefore, a single step profile for the mass disturbance could be used, as it was implemented in many prior publications. The variations in the mass were simulated by adding a term named $Mass_{dist}$ to (16)–(18), resulting in Equations (54)–(56). $Mass_{dist}$ is calculated using Equation (57).

$$\ddot{x} = \frac{1}{m + Mass_{dist}}\left(u_1 u_x - C_{dx}\dot{x}\right) \tag{54}$$

$$\ddot{y} = \frac{1}{m + Mass_{dist}}\left(u_1 u_y - C_{dy}\dot{y}\right) \tag{55}$$

$$\ddot{z} = \frac{1}{m + Mass_{dist}}\left(u_1(\cos\theta\cos\varphi) - C_{dz}\dot{z} - gm\right) \tag{56}$$

$$Mass_{dist} = Rand{\cdot}step(t-5) \tag{57}$$

In Equation (57), "*Rand*" represents a random value between 0% and 70% of the mass of the robot. Every 5 seconds, the mass of the robot is perturbed by a weight disturbance with the value of $Mass_{dist}$. An RL agent block from the reinforcement learning toolbox was used in the Simulink environment to implement the deep reinforcement learning policy. During the experiments, the presented RL algorithm interacted with the environment in the simulator, thereby learning the appropriate strategy for estimating and updating the parameters of the PID controllers and the gravity compensator. The achieved reward throughout the training process is shown in Figure 7.

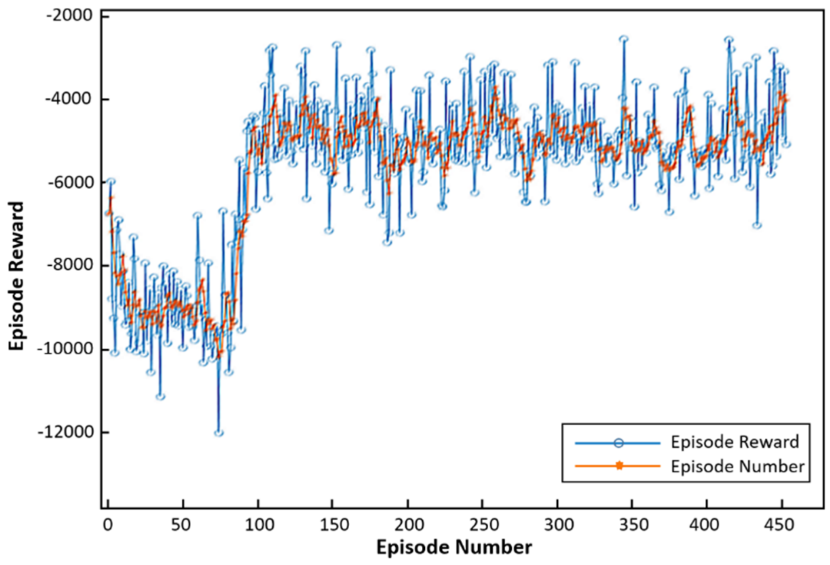

Figure 7. Episode reward and average reward recorded during training.

To train the neural networks, different values for hyperparameters were applied, aimed at finding the best possible values. Table 4 shows the optimal values obtained for the hyperparameters.

Table 4. The optimal values for hyperparameters.

Hyperparameter	Value
Critic Learning Rate	0.0001
Actor Learning Rate	0.00001
Critic Gradient Threshold	1
Actor Gradient Threshold	4
Variance	0.3
Variance Decay Rate	0.00001
Experience Buffer	1,000,000
Mini-Batch Size	64
Target Smooth Factor	0.001

During the aerial robot operation, the trained policy actively estimated the parameters of the controllers. The first experiment was done in two stages (the first time with the proposed controller and the second time with conventional PID controllers) to compare the performance of the proposed controller with that of the conventional PID controller. During the experiments, some weight disturbances (random values between 0% and 70% of the mass of the robot) perturbed the system. The first defined maneuver for the aerial robot was to get to the coordinates (x, y, z = 3.5, 3.5, 4) and stabilize its position in space. Figures 8–10 show the performance of the PID controller and the proposed controller.

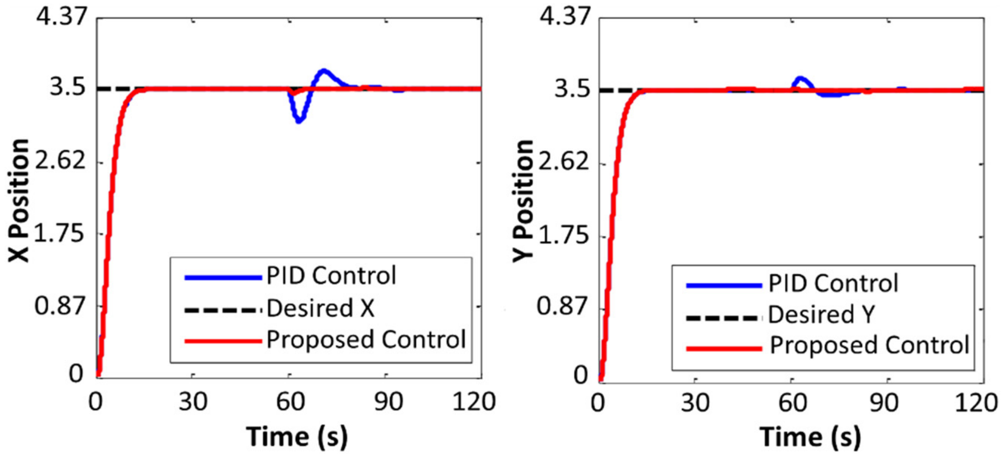

Figure 8. Performance of the quadcopter in tracking the desired X and Y positions.

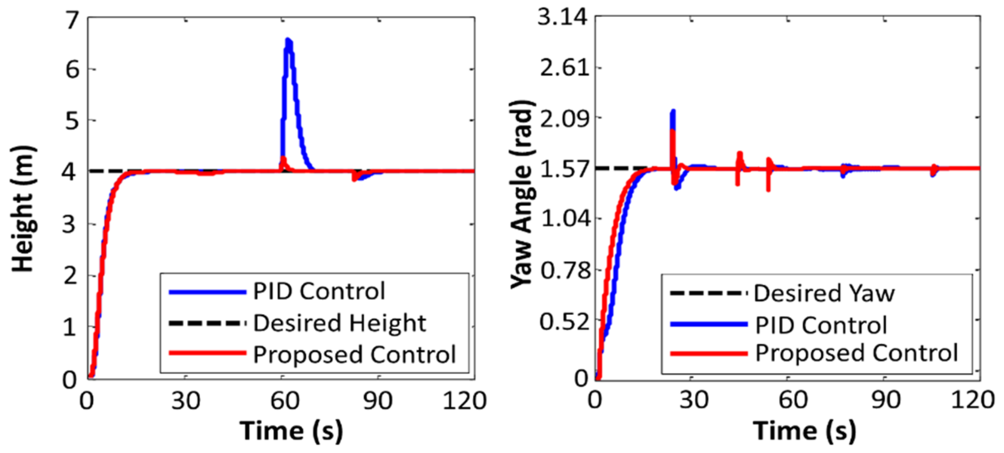

Figure 9. Performance of the quadcopter in attaining desired height and yaw angle.

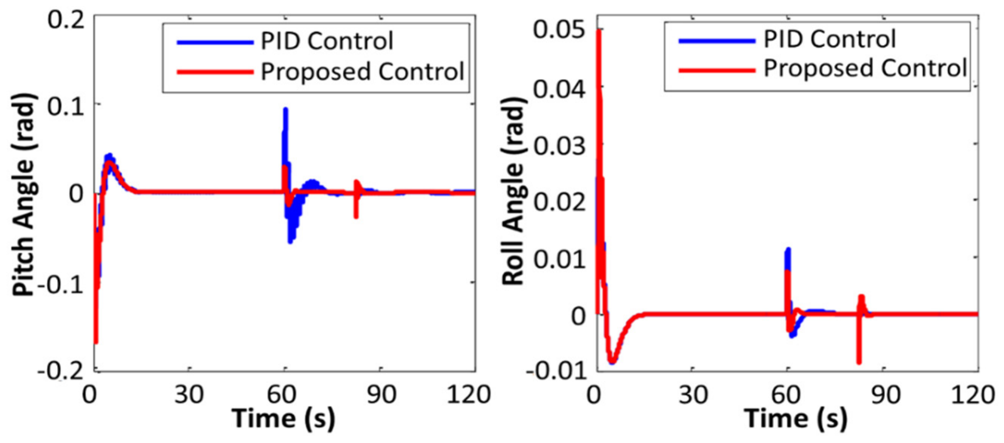

Figure 10. Performance of the quadcopter in tracking the desired Roll and Pitch angles.

Variations in the values of the manipulated variables u_1, u_2, u_3 and u_4 are plotted in Figures 11 and 12. Figures 13 and 14 exhibit changes in control gain values that occurred during the operation of the robot.

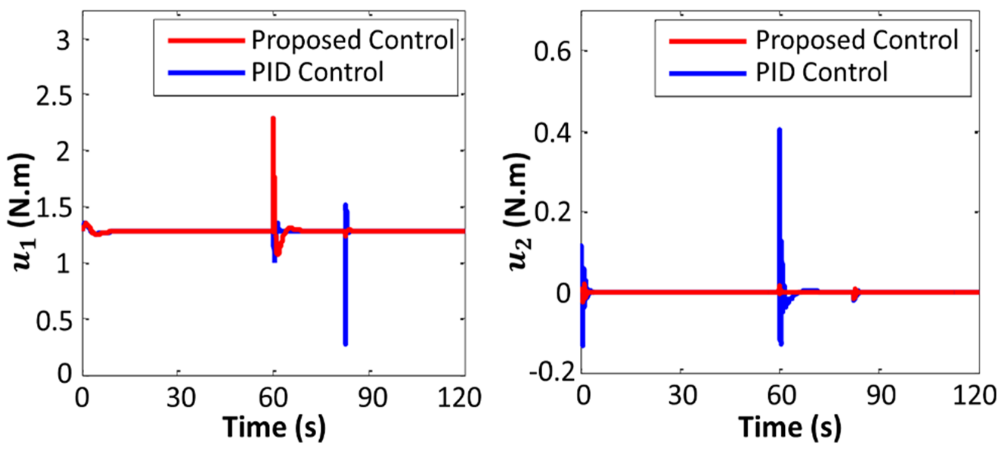

Figure 11. Variations in manipulated variables u_1 and u_2.

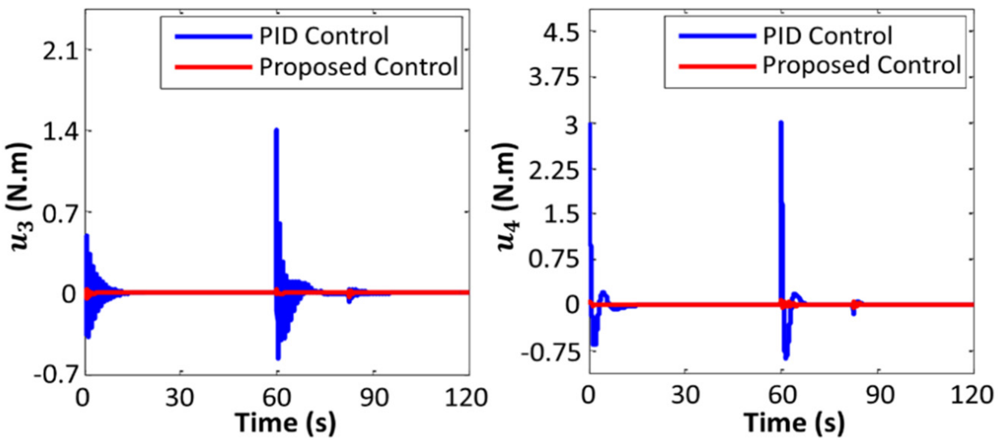

Figure 12. Variations in manipulated variables u_3 and u_4.

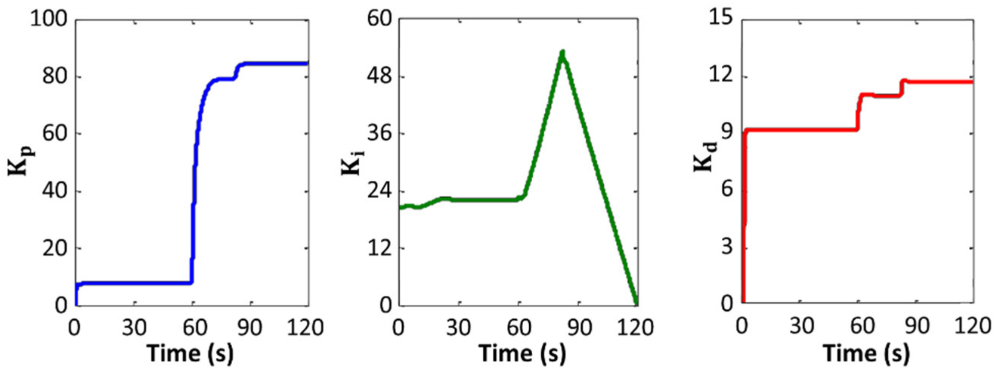

Figure 13. Variations in adaptive gain values of the X position PID controller.

A comparative study using a range of values was conducted to explore the influence of RL hyperparameters on the steady-state error of the altitude control. According to the results of the experiments, zero noise generated the highest steady-state error. Setting the noise to a very high value, however, prevented the actor from learning the best policy, resulting in more errors. It was observed that increasing the variance enhances the exploration of action space. For both actor and critic, the gradient threshold varied between 1, 4, and infinity. Table 5 summarizes the results of the trials.

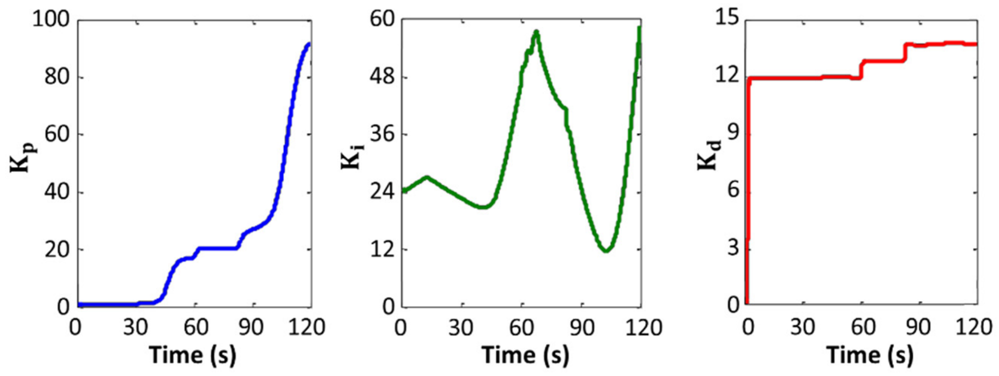

Figure 14. Variations in adaptive gain values of the Y position PID controller.

Table 5. Variation of RL hyperparameters and their effect on the steady-state error.

Critic Learning Rate	Actor Grad Threshold	Critic Grad Threshold	Variance (Noise)	Mini Batch Size	Steady State Error
0.0001	4	1	0.3	64	0.00064
0.0001	inf	1	0.3	64	0.00093
0.0001	1	4	0.3	64	0.00011
0.0001	1	inf	0.3	64	0.00008
0.0001	1	1	0	64	0.00435
0.0001	1	1	0.5	64	0.00010
0.0001	1	1	0.3	128	0.00045
0.00005	4	1	0.3	64	0.00001

In the second experiment, a waypoint tracking maneuver was conducted to evaluate the ability of the robot, with the proposed control architecture, in tracking consecutive waypoints. Figure 15 illustrates the performance of the aerial robot in tracking the waypoints.

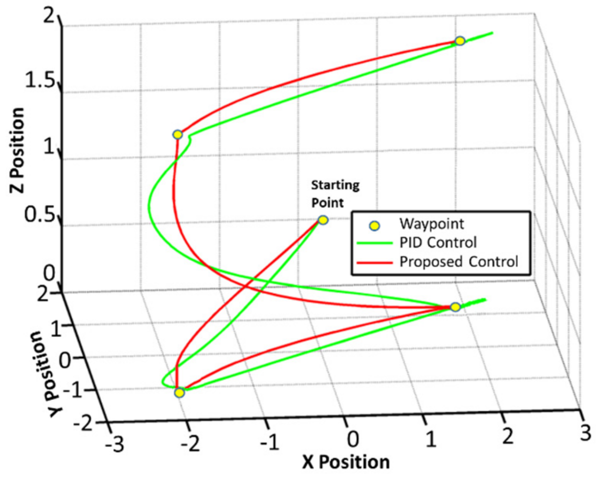

Figure 15. Performance of the aerial robot in tracking waypoints.

From the results of the experiments, it can be observed that the robot with the proposed controller provided smoother trajectory. Furthermore, the error in the trajectory tracking of the robot with the proposed controller was less than that of the PID controller. The results showed that the proposed control algorithm is able to stabilize the system performance when the robot is subjected to weight disturbances. It must be noted that the performance of the conventional PID controller was satisfactory as long as the extra weight was low, but when the added weight was large, the basic PID controller failed to control the robot properly.

5. Conclusions

A new deep reinforcement learning-based adaptive controller for controlling an aerial robot was proposed in this research paper. To interact with the robot dynamics model and learn the right policy for actively adjusting the controller, the proposed adaptive control method leveraged a deep deterministic policy gradient algorithm. A linear model predictive controller and an adaptive gravity compensator gain were used in the proposed control system for the robot altitude controller. The performance of the proposed control architecture was compared to that of traditional PID controllers with fixed settings in the Simulink environment. Experiments in a simulated environment demonstrated that the presented control algorithm outperforms ordinary PID controllers in terms of trajectory tracking and altitude control.

Author Contributions: Conceptualization, A.B. and D.-J.L.; methodology, A.B.; software, A.B.; validation, A.B.; formal analysis, A.B.; investigation, A.B.; resources, A.B.; writing—original draft preparation, A.B.; writing—review and editing, A.B.; visualization, A.B.; supervision, D.-J.L.; project administration, D.-J.L.; funding acquisition, D.-J.L. All authors have read and agreed to the published version of the manuscript.

Funding: This research was supported by the Unmanned Vehicles Core Technology Research and Development Program through the National Research Foundation of Korea (NRF), Unmanned Vehicle Advanced Research Center (UVARC), which is funded by the Ministry of Science and ICT, the Republic of Korea (2020M3C1C1A01082375). This research was also supported by DNA+Drone Technology Development Program through the National Research Foundation of Korea (NRF), which is funded by the Ministry of Science and ICT (No. NRF-2020M3C1C2A01080819).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Saunders, J.; Saeedi, S.; Li, W. Autonomous Aerial Delivery Vehicles, a Survey of Techniques on how Aerial Package Delivery is Achieved. *arXiv* **2021**, arXiv:2110.02429.
2. Joshi, G.; Virdi, J.; Chowdhary, G. Design and flight evaluation of deep model reference adaptive controller. In Proceedings of the AIAA Scitech 2020 Forum, Orlando, FL, USA, 6–10 January 2020; p. 1336.
3. Balcazar, R.; Rubio, J.D.J.; Orozco, E.; Cordova, D.A.; Ochoa, G.; Garcia, E.; Pacheco, J.; Gutierrez, G.J.; Mujica-Vargas, D.; Aguilar-Ibañez, C. The Regulation of an Electric Oven and an Inverted Pendulum. *Symmetry* **2022**, *14*, 759. [CrossRef]
4. Rubio, J.D.J.; Orozco, E.; Cordova, D.A.; Islas, M.A.; Pacheco, J.; Gutierrez, G.J.; Zacarias, A.; Soriano, L.A.; Meda-Campana, J.A.; Mujica-Vargas, D. Modified Linear Technique for the Controllability and Observability of Robotic Arms. *IEEE Access* **2022**, *10*, 3366–3377. [CrossRef]
5. Aguilar-Ibanez, C.; Moreno-Valenzuela, J.; García-Alarcón, O.; Martinez-Lopez, M.; Acosta, J.Á.; Suarez-Castanon, M.S. PI-Type Controllers and Σ–Δ Modulation for Saturated DC-DC Buck Power Converters. *IEEE Access* **2021**, *9*, 20346–20357. [CrossRef]
6. Soriano, L.A.; Rubio, J.D.J.; Orozco, E.; Cordova, D.A.; Ochoa, G.; Balcazar, R.; Cruz, D.R.; Meda-Campaña, J.A.; Zacarias, A.; Gutierrez, G.J. Optimization of Sliding Mode Control to Save Energy in a SCARA Robot. *Mathematics* **2021**, *9*, 3160. [CrossRef]
7. Vosoogh, M.; Piltan, F.; Mirshekaran, A.M.; Barzegar, A.; Siahbazi, A.; Sulaiman, N. Integral Criterion-Based Adaptation Control to Vibration Reduction in Sensitive Actuators. *Int. J. Hybrid Inf. Technol.* **2015**, *8*, 11–30. [CrossRef]
8. Soriano, L.A.; Zamora, E.; Vazquez-Nicolas, J.M.; Hernández, G.; Madrigal, J.A.B.; Balderas, D. PD Control Compensation Based on a Cascade Neural Network Applied to a Robot Manipulator. *Front. Neurorobotics* **2020**, *14*, 78. [CrossRef]
9. Kada, B.; Ghazzawi, Y. Robust PID controller design for an UAV flight control system. In Proceedings of the World Congress on Engineering and Computer Science, San Francisco, CA, USA, 19–21 October 2011; Volume 2, pp. 1–6.
10. Silva-Ortigoza, R.; Hernández-Márquez, E.; Roldán-Caballero, A.; Tavera-Mosqueda, S.; Marciano-Melchor, M.; García-Sánchez, J.R.; Hernández-Guzmán, V.M.; Silva-Ortigoza, G. Sensorless Tracking Control for a "Full-Bridge Buck Inverter–DC Motor" System: Passivity and Flatness-Based Design. *IEEE Access* **2021**, *9*, 132191–132204. [CrossRef]
11. Mirshekaran, A.M.; Piltan, F.; Sulaiman, N.; Siahbazi, A.; Barzegar, A.; Vosoogh, M. Design Intelligent Model-free Hybrid Guidance Controller for Three Dimension Motor. *Int. J. Inf. Eng. Electron. Bus.* **2014**, *6*, 29–35. [CrossRef]
12. Barzegar, A.; Piltan, F.; Mirshekaran, A.M.; Siahbazi, A.; Vosoogh, M.; Sulaiman, N. Research on Hand Tremors-Free in Active Joint Dental Automation. *Int. J. Hybrid Inf. Technol.* **2015**, *8*, 71–96. [CrossRef]
13. He, X.; Kou, G.; Calaf, M.; Leang, K.K. In-Ground-Effect Modeling and Nonlinear-Disturbance Observer for Multirotor Unmanned Aerial Vehicle Control. *J. Dyn. Syst. Meas. Control* **2019**, *141*, 071013. [CrossRef]
14. Barzegar, A.; Doukhi, O.; Lee, D.J.; Jo, Y.H. Nonlinear Model Predictive Control for Self-Driving cars Tra-jectory Tracking in GNSS-denied environments. In Proceedings of the 2020 20th International Conference on Control, Automation and Systems (ICCAS), Busan, Korea, 13–16 October 2020; IEEE: Piscataway, NJ, USA, 2020; pp. 750–755.

15. Cao, G.; Lai, E.M.-K.; Alam, F. Gaussian Process Model Predictive Control of an Unmanned Quadrotor. *J. Intell. Robot. Syst.* **2017**, *88*, 147–162. [CrossRef]
16. Mehndiratta, M.; Kayacan, E. Gaussian Process-based Learning Control of Aerial Robots for Precise Visualization of Geological Outcrops. In Proceedings of the 2020 European Control Conference (ECC), St. Petersburg, Russia, 12–15 May 2020; IEEE: Piscataway, NJ, USA, 2020; pp. 10–16. [CrossRef]
17. Caldwell, J.; Marshall, J.A. Towards Efficient Learning-Based Model Predictive Control via Feedback Lineari-zation and Gaussian Process Regression. In Proceedings of the 2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Prague, Czech Republic, 27 September–1 October 2021; IEEE: Piscataway, NJ, USA, 2021; pp. 4306–4311.
18. Chee, K.Y.; Jiahao, T.Z.; Hsieh, M.A. KNODE-MPC: A Knowledge-Based Data-Driven Predictive Control Framework for Aerial Robots. *IEEE Robot. Autom. Lett.* **2022**, *7*, 2819–2826. [CrossRef]
19. Richards, A.; How, J. Decentralized model predictive control of cooperating UAVs. In Proceedings of the 2004 43rd IEEE Conference on Decision and Control (CDC) (IEEE Cat. No. 04CH37601), Nassau, Bahamas, 14–17 December 2004; IEEE: Piscataway, NJ, USA, 2004; Volume 4, pp. 4286–4291.
20. Scholte, E.; Campbell, M. Robust Nonlinear Model Predictive Control With Partial State Information. *IEEE Trans. Control Syst. Technol.* **2008**, *16*, 636–651. [CrossRef]
21. Mathisen, S.H.; Gryte, K.; Johansen, T.; Fossen, T.I. Non-linear Model Predictive Control for Longitudinal and Lateral Guidance of a Small Fixed-Wing UAV in Precision Deep Stall Landing. In Proceedings of the AIAA Infotech@ Aerospace, San Diego, CA, USA, 4–8 January 2016; p. 0512. [CrossRef]
22. Barzegar, A.; Doukhi, O.; Lee, D.-J. Design and Implementation of an Autonomous Electric Vehicle for Self-Driving Control under GNSS-Denied Environments. *Appl. Sci.* **2021**, *11*, 3688. [CrossRef]
23. Iskandarani, M.; Givigi, S.N.; Fusina, G.; Beaulieu, A. Unmanned Aerial Vehicle formation flying using Linear Model Predictive Control. In Proceedings of the 2014 IEEE International Systems Conference Proceedings, Ottawa, ON, Canada, 31 March–3 April 2014; IEEE: Piscataway, NJ, USA, 2014; pp. 18–23. [CrossRef]
24. Britzelmeier, A.; Gerdts, M. A Nonsmooth Newton Method for Linear Model-Predictive Control in Tracking Tasks for a Mobile Robot with Obstacle Avoidance. *IEEE Control Syst. Lett.* **2020**, *4*, 886–891. [CrossRef]
25. Wang, Q.; Zhang, A.; Sun, H.Y. MPC and SADE for UAV real-time path planning in 3D environment. In *Proceedings 2014 IEEE International Conference on Security, Pattern Analysis, and Cybernetics (SPAC), Wuhan, China, 18–19 October 2014*; IEEE: Piscataway, NJ, USA, 2014; pp. 130–133. [CrossRef]
26. Pan, Z.; Li, D.; Yang, K.; Deng, H. Multi-Robot Obstacle Avoidance Based on the Improved Artificial Potential Field and PID Adaptive Tracking Control Algorithm. *Robotica* **2019**, *37*, 1883–1903. [CrossRef]
27. Doukhi, O.; Fayjie, A.R.; Lee, D.J. Intelligent Controller Design for Quad-Rotor Stabilization in Presence of Parameter Variations. *J. Adv. Transp.* **2017**, *2017*, 4683912. [CrossRef]
28. Rosales, C.D.; Tosetti, S.R.; Soria, C.M.; Rossomando, F.G. Neural Adaptive PID Control of a Quadrotor using EFK. *IEEE Lat. Am. Trans.* **2018**, *16*, 2722–2730. [CrossRef]
29. Rosales, C.; Soria, C.M.; Rossomando, F.G. Identification and adaptive PID Control of a hexacopter UAV based on neural networks. *Int. J. Adapt. Control Signal Process.* **2018**, *33*, 74–91. [CrossRef]
30. Sarhan, A.; Qin, S. Adaptive PID Control of UAV Altitude Dynamics Based on Parameter Optimization with Fuzzy Inference. *Int. J. Model. Optim.* **2016**, *6*, 246–251. [CrossRef]
31. Siahbazi, A.; Barzegar, A.; Vosoogh, M.; Mirshekaran, A.M.; Soltani, S. Design Modified Sliding Mode Controller with Parallel Fuzzy Inference System Compensator to Control of Spherical Motor. *Int. J. Intell. Syst. Appl.* **2014**, *6*, 12–25. [CrossRef]
32. Hu, X.; Liu, J. Research on uav balance control based on expert-fuzzy adaptive pid. In Proceedings of the 2020 IEEE International Conference on Advances in Electrical Engineering and Computer Applications (AEECA), Dalian, China, 25–27 August 2020; IEEE: Piscataway, NJ, USA, 2020; pp. 787–789.
33. Barzegar, A.; Piltan, F.; Vosoogh, M.; Mirshekaran, A.M.; Siahbazi, A. Design Serial Intelligent Modified Feedback Linearization like Controller with Application to Spherical Motor. *Int. J. Inf. Technol. Comput. Sci.* **2014**, *6*, 72–83. [CrossRef]
34. Duan, Y.; Chen, X.; Houthooft, R.; Schulman, J.; Abbeel, P. Benchmarking deep reinforcement learning for continuous control. In Proceedings of the International Conference on Machine Learning, New York, NY, USA, 20–22 June 2016; PMLR: London, UK, 2016; pp. 1329–1338.
35. Kober, J.; Bagnell, J.A.; Peters, J. Reinforcement learning in robotics: A survey. *Int. J. Robot. Res.* **2013**, *32*, 1238–1274. [CrossRef]
36. Claus, C.; Boutilier, C. The dynamics of reinforcement learning in cooperative multiagent systems. In Proceedings of the Fifteenth National/Tenth Conference on Artificial Intelligence/Innovative Applications of Artificial Intelligence, Madison, WI, USA, 26–30 July 1998; pp. 746–752.
37. Bernstein, A.V.; Burnaev, E.V. Reinforcement learning in computer vision. In Proceedings of the Tenth International Conference on Machine Vision (ICMV 2017), Vienna, Austria, 13–15 November 2048; Volume 10696, pp. 458–464.
38. Bohn, E.; Coates, E.M.; Moe, S.; Johansen, T.A. Deep Reinforcement Learning Attitude Control of Fixed-Wing UAVs Using Proximal Policy optimization. In Proceedings of the 2019 International Conference on Unmanned Aircraft Systems (ICUAS), Atlanta, GA, USA, 11–14 June 2019; IEEE: Piscataway, NJ, USA, 2019; pp. 523–533. [CrossRef]
39. Koch, W.; Mancuso, R.; West, R.; Bestavros, A. Reinforcement Learning for UAV Attitude Control. *ACM Trans. Cyber-Phys. Syst.* **2019**, *3*, 1–21. [CrossRef]

40. Polvara, R.; Patacchiola, M.; Sharma, S.; Wan, J.; Manning, A.; Sutton, R.; Cangelosi, A. Toward End-to-End Control for UAV Autonomous Landing via Deep Reinforcement Learning. In Proceedings of the 2019 International Conference on Unmanned Aircraft Systems (ICUAS), Dallas, TX, USA, 12–15 June 2018; IEEE: Piscataway, NJ, USA, 2018; pp. 115–123. [CrossRef]
41. Passalis, N.; Tefas, A. Continuous drone control using deep reinforcement learning for frontal view person shooting. *Neural Comput. Appl.* **2019**, *32*, 4227–4238. [CrossRef]
42. Zheng, L.; Zhou, Z.; Sun, P.; Zhang, Z.; Wang, R. A novel control mode of bionic morphing tail based on deep reinforcement learning. *arXiv* **2020**, arXiv:2010.03814.
43. Botvinick, M.; Ritter, S.; Wang, J.X.; Kurth-Nelson, Z.; Blundell, C.; Hassabis, D. Reinforcement learning, fast and slow. *Trends Cogn. Sci.* **2019**, *23*, 408–422. [CrossRef]
44. Pi, C.-H.; Ye, W.-Y.; Cheng, S. Robust Quadrotor Control through Reinforcement Learning with Disturbance Compensation. *Appl. Sci.* **2021**, *11*, 3257. [CrossRef]
45. Shi, Q.; Lam, H.-K.; Xuan, C.; Chen, M. Adaptive neuro-fuzzy PID controller based on twin delayed deep deterministic policy gradient algorithm. *Neurocomputing* **2020**, *402*, 183–194. [CrossRef]
46. Dooraki, A.R.; Lee, D.-J. An innovative bio-inspired flight controller for quad-rotor drones: Quad-rotor drone learning to fly using reinforcement learning. *Robot. Auton. Syst.* **2020**, *135*, 103671. [CrossRef]
47. Quan, Q. *Introduction to Multicopter Design and Control*, 1st ed.; Springer Nature: Singapore, 2017; pp. 99–120.
48. Hernandez, A.; Copot, C.; De Keyser, R.; Vlas, T.; Nascu, I. Identification and path following control of an AR. Drone quadrotor. In Proceedings of the 2013 17th International Conference on System Theory, Control and Computing (ICSTCC), Sinaia, Romania, 11–13 October 2013; IEEE: Piscataway, NJ, USA, 2013; pp. 583–588.
49. Fan, J.; Wang, Z.; Xie, Y.; Yang, Z. A theoretical analysis of deep Q-learning. In Proceedings of the Learning for Dynamics and Control, Virtual, 7–8 June 2020; PMLR: London, UK, 2020; pp. 486–489.
50. Jesus, C., Jr.; Bottega, J.A.; Cuadros, M.A.S.L.; Gamarra, D.F.T. Deep deterministic policy gradient for navigation of mobile robots in simulated environments. In Proceedings of the 2019 19th International Conference on Advanced Robotics (ICAR), Belo Horizonte, Brazil, 2–6 December 2019; pp. 362–367.
51. Sandipan, S.; Wadoo, S. Linear optimal control of a parrot AR drone 2.0. In Proceedings of the 2017 IEEE MIT Undergraduate Research Technology Conference (URTC), Cambridge, MA, USA, 3–5 November 2017; IEEE: Piscataway, NJ, USA, 2017; pp. 1–5.
52. Glazkov, T.V.; Golubev, A.E. Using Simulink Support Package for Parrot Minidrones in nonlinear control education. *AIP Conf. Proc.* **2019**, *2195*, 020007. [CrossRef]
53. Kaplan, M.R.; Eraslan, A.; Beke, A.; Kumbasar, T. Altitude and Position Control of Parrot Mambo Minidrone with PID and Fuzzy PID Controllers. In Proceedings of the 2019 11th International Conference on Electrical and Electronics Engineering (ELECO), Bursa, Turkey, 28–30 November 2019; IEEE: Piscataway, NJ, USA, 2019; pp. 785–789. [CrossRef]
54. Gill, J.S.; Velashani, M.S.; Wolf, J.; Kenney, J.; Manesh, M.R.; Kaabouch, N. Simulation Testbeds and Frameworks for UAV Performance Evaluation. In Proceedings of the 2021 IEEE International Conference on Electro Information Technology (EIT), Mt. Pleasant, MI, USA, 14–15 May 2021; IEEE: Piscataway, NJ, USA, 2021; pp. 335–341.