Received: 7 August 2023 | Revised: 3 November 2023 | Accepted: 19 November 2023

IET Control Theory & Applications

DOI: 10.1049/cth2.12615

ORIGINAL RESEARCH

Robust and intelligent control of quadrotors subject to wind gusts

Paulo V. G. Simplício[1] | **João R. S. Benevides**[1] | **Roberto S. Inoue**[2] | **Marco H. Terra**[1]

[1]Department of Electrical and Computer Engineering, São Carlos School of Engineering, University of São Paulo, São Carlos, São Paulo, Brazil

[2]Department of Computer Sciences, Federal University of São Carlos, São Carlos, São Paulo, Brazil

Correspondence
Paulo V. G. Simplício, Department of Electrical and Computer Engineering, São Carlos School of Engineering, University of São Paulo, São Carlos, São Paulo 13566-590, Brazil.
Email: paulogalvao@usp.br

Funding information
Fundação de Amparo à Pesquisa do Estado de São Paulo, Grant/Award Numbers: 2014/50851-0, 2017/05668-0, 2018/13848-1; Conselho Nacional de Desenvolvimento Científico e Tecnológico, Grant/Award Numbers: 421131/2018-7, 465755/2014-3; Coordenação de Aperfeiçoamento de Pessoal de Nível Superior, Grant/Award Number: 88887.509571/2020

Abstract
The combination of artificial neural networks with advanced control techniques has shown great potential to reject uncertainties and disturbances that affect the quadrotor during trajectory tracking. However, it is still a complex and little-explored challenge. In this sense, this work proposes the development of robust and intelligent architectures for position control of quadrotors, improving flight performance during trajectory tracking. The proposed architectures combine a robust linear quadratic regulator (RLQR) with deep neural networks (DNNs). In addition, a comparative study is performed to evaluate the performance of the proposed architectures using three other widely used controllers: linear quadratic regulator (LQR), proportional-integral-derivative (PID), and feedback linearization (FL). The architectures were developed using the robot operating system (ROS), and the experiments were performed with a commercial quadrotor, the Parrot^{TM} Bebop 2.0. Flights were performed by applying wind gusts to the aircraft's body, and the experimental results showed that using neural networks combined with controllers, robust or not, improves quadrotors' flight performance.

1 | INTRODUCTION

Interest in research related to autonomous unmanned aerial vehicles (UAVs) has increased intensively, especially concerning quadrotors. The increasing use of this type of aircraft is mainly due to its characteristics and particularities. According to [1], the quadrotor can be defined as a UAV with rotary wings endowed with a rigid structure with rotors in the extremity. The quadrotor architecture, with rotating propellers, provides good stability in hovering flight and ease of taking off and landing in places with limited space [2].

The physical features combined with advanced control techniques have enabled the quadrotor to be used in a wide variety of applications, such as aerial mapping and photography [3], urban operations [4], precision agriculture [5], surveillance and reconnaissance missions [6], and military applications [7]. However, one of the main problems lies in ensuring that the quadrotor can operate and perform its assigned activity safely and reliably in the face of environmental adversity.

Given this, several control techniques have been proposed over the years, the main ones being classified into linear, nonlinear, and intelligent [8]. In dealing with linear controllers, the proportional-integral-derivative (PID) [9], the linear quadratic regulator (LQR) [10], and the H_∞ [11] stands out. Regarding nonlinear control techniques, we can highlight: feedback linearization (FL) [12], sliding mode control [13], and Backstepping [14]. However, using these control techniques in isolation has limitations when the aircraft is subject to parametric uncertainties and external disturbances.

Unlike most conventional control techniques (linear and nonlinear), intelligent control can adapt to the operating environment and lack plant information. In this sense, we can implement intelligent controllers to operate in conditions with changing process parameters and disturbances with unknown magnitudes [15]. According to Kim [16], the main intelligent techniques are fuzzy control [17], artificial neural networks (ANN) [18], and even some variations of model predictive control (MPC) [19].

The research line that contemplates the application of ANNs in quadrotors has received increasing attention from researchers in aerial robotics. This is because, according to [8], combining ANNs with controllers, linear or not, increases the robustness and effectiveness of the proposed control system. It is possible to find in the literature works that use deep neural networks (DNNs) within this context. This can be explained by the ability of this type of network to approximate complex nonlinear functions with superior performance. In the work of [18], a DNN was used to optimize the quadrotor trajectory tracking. The developed network was first trained offline, aiming to reproduce the operation of the PID controller, and later online, using fuzzy logic. In the work of [20], DNNs were combined with the PID controller to reduce the trajectory tracking error. In this case, six DNNs are responsible for sending reference signals to each of the degrees of freedom of the quadrotor.

While adding intelligent techniques in quadrotors improves flight performance, it is also necessary to consider external disturbances for increased robustness. This disturbance can lead the aircraft to instability and put the operation at risk. To address this problem, initially, several techniques for disturbance estimation and attenuation were developed, the most common being: active disturbance rejection control (ADRC), disturbance observer-based control (DOBC), disturbance accommodation control (DAC), and composite hierarchical antidisturbance control (CHADC). According to [21], the DOBC architecture has some advantages when compared to other techniques aimed at disturbance attenuation. In this case, we can highlight the modular structure, facilitating the combination with existing controllers; the feedforward compensation, which promotes a fast response; and the ability to adapt to different magnitudes of disturbances.

Recently, control architectures using ANNs have been proposed to treat the effects caused by wind gusts in quadrotors. These architectures combine ANNs with conventional controllers and, in some cases, disturbance estimation and attenuation techniques. Regarding the combination of ANNs with disturbance observers, in [22], for example, a robust architecture is proposed. In this case, the authors developed a reinforcement learning-based neural network that acts in conjunction with a disturbance observer, responsible for estimating the magnitude of wind gusts that affect the quadrotor body. In [23], an intelligent disturbance observer was developed to act in conjunction with an FL controller. The authors used a radial-based neural network to improve the disturbance estimation, which was then attenuated by a compensator, forming a DOBC architecture.

Considering the above scenario, especially the influence of wind gusts on the quadrotor body and the lack of adaptability when leading with robust and conventional controllers, we propose intelligent control architectures for trajectory tracking of a quadrotor affected by wind gusts in this paper. The proposed architectures combine a Robust Linear Quadratic Regulator (RLQR) with deep neural networks. The RLQR presented by [24] is based on penalty parameters and regularized least squares, forming an algorithm that does not depend on tuning parameters in online applications. This controller needs only the adjustment of the weighting matrices and previously defined uncertainties, which makes it useful for real-time applications.

The contributions of this paper can be summarized as follows:

- Two intelligent and robust control architectures have been proposed combining neural networks and a robust recursive controller. The architectures are responsible for the position control of a quadrotor affected by wind gusts during trajectory tracking.
- The combination of a DNN that generates a reference signal with a robust controller in order to provide a robust and adaptive architecture (Architecture 1).
- A new architecture that uses a DNN to estimate wind disturbances affecting the aircraft to avoid the use of complex estimators (Architecture 2).
- Real experiments and a comparative study with standard controllers were performed.

This paper is organized as follows: Section 2 presents the adopted dynamic model of the quadrotor. Section 3 presents the fundamentals of deep neural networks, focusing on the algorithms used. Section 4 briefly describes disturbance estimation and attenuation techniques, with details for the DOBC architecture and the controllers used. Section 5 presents how the controllers were combined with the DNNs. Section 6 brings the methodology used and the set-up of the experimental environment. Section 7 presents the flight results of practical experiments. In Section 8, the paper is concluded.

2 | QUADROTOR MODEL

In order to design an intelligent controller-based model of a quadrotor, we need to define its dynamic model. The quadrotor has six degrees of freedom, which can be given by the three Euler angles (ϕ, θ, ψ) and the three reference axes (x, y, z) [2]. In addition, four control inputs can be considered, characterizing the quadrotor as an underactuated system.

Over the years, several mathematical models have been proposed to represent the dynamics of the quadrotor. Both the Newton-Euler and the Euler-Lagrange formulations are abundant in the literature. Aiming to develop a simplified dynamic model of a quadrotor, [25] proposed a model for a Parrot$^{\text{TM}}$ AR.Drone. This model was adapted for Bebop 2.0 in [26] and, although it does not consider all the nonlinear dynamics of a quadrotor, it provides an effective approximation. The Bebop 2.0 is controlled with a control input of the type $\nu = [\nu_{\dot{x}}\ \nu_{\dot{y}}\ \nu_{\dot{z}}\ \nu_{\dot{\psi}}]^T$, where ν represents the linear velocity commands in the x, y and z axes, in addition to the angular velocity around the z axis, represented by $\dot{\psi}$. In this sense, we have that:

- $\nu_{\dot{x}}$ represents the limited rotation command around the y-axis, causing movement in the x direction;
- $\nu_{\dot{y}}$ represents the limited rotation command around the x-axis, causing movement in the y direction;

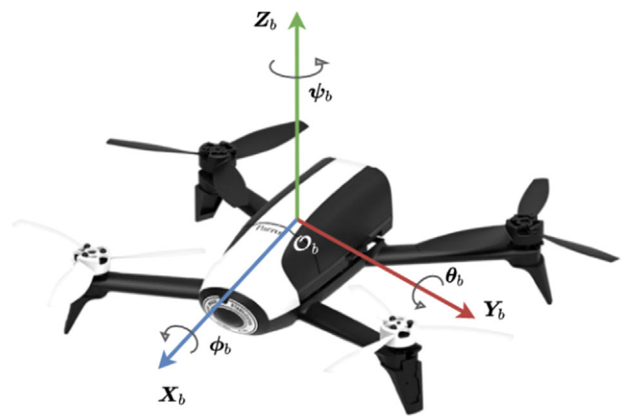

FIGURE 1 Adopted coordinate system for the Parrot Bebop 2.0.

- $\nu_{\dot{z}}$ linear speed control for the z-axis, causing movement in the z direction;
- $\nu_{\dot{\psi}}$ represents the angular velocity command that causes rotation about the z-axis.

The velocity vector is interpreted by the Parrot Bebop 2.0 embedded control system. Moreover, Figure 1 shows the adopted coordinate system for the Parrot Bebop 2.0.

Considering that attitude stabilization is guaranteed by Bebop 2.0's internal control system, a simplified model is given by:

$$\ddot{q}(t) = \Lambda\dot{q}(t) + \Gamma\nu(t), \tag{1}$$

where $\dot{q} = [q_{\dot{x}}\ q_{\dot{y}}\ q_{\dot{z}}\ q_{\dot{\psi}}]^T$ is the state vector, $\Lambda \in \mathbb{R}^{4\times4}$ and $\Gamma \in \mathbb{R}^{4\times4}$ are model matrices described as:

$$\Lambda = \begin{bmatrix} \gamma_2\cos(\psi) & -\gamma_4\sin(\psi) & 0 & 0 \\ \gamma_2\sin(\psi) & \gamma_4\cos(\psi) & 0 & 0 \\ 0 & 0 & \gamma_6 & 0 \\ 0 & 0 & 0 & \gamma_8 \end{bmatrix},$$

$$\Gamma = \begin{bmatrix} \gamma_1\cos(\psi) & -\gamma_3\sin(\psi) & 0 & 0 \\ \gamma_1\sin(\psi) & \gamma_3\cos(\psi) & 0 & 0 \\ 0 & 0 & \gamma_5 & 0 \\ 0 & 0 & 0 & \gamma_7 \end{bmatrix},$$

whose parameters $(\gamma_1, \dots \gamma_8)$ were identified as proposed in [26]. Given this, we can directly control the quadrotor position in the x, y, and z axis; and the orientation in the z-axis.

In order to perform trajectory tracking, the model can be rewritten as a function of the error state space, where the controller will act. Considering the q^d as the desired trajectory signal, we have:

$$x(t) = \begin{bmatrix} \dot{q}(t) - \dot{q}^d(t) \\ q(t) - q^d(t) \end{bmatrix}, \tag{2}$$

as the trajectory error, where $q^d = [q_x^d\ q_y^d\ q_z^d\ q_\psi^d]$ is the desired position vector in x, y, z, and ψ. Thus, the state space of the error, considering external disturbances, is given by:

$$\dot{x}(t) = \underbrace{\begin{bmatrix} \Lambda & 0 \\ I & 0 \end{bmatrix}}_{A} x + \underbrace{\begin{bmatrix} I \\ 0 \end{bmatrix}}_{B} u + \underbrace{\begin{bmatrix} \Xi \\ 0 \end{bmatrix}}_{B_d} d, \tag{3}$$

where $x = [\dot{q}\ q]$ is the state vector, d is the disturbance affecting the quadrotor, composed mainly of model uncertainties, neglected dynamics, and external disturbances such as wind gusts; $\Xi = diag(\frac{1}{m} I_{3\times3},\ \frac{1}{I_z} I_{1\times1})$ is the matrix that maps external disturbances, with m being the mass of the quadrotor and I_z the moment of inertia with respect to the z-axis, and u is used to calculate the actual control input sent to Bebop 2.0, given by:

$$\nu = \Gamma^{-1}(u + \ddot{q}^d - \Lambda\dot{q}^d). \tag{4}$$

For controller implementation, we can discretize (3) using Euler's method, resulting in:

$$x_{i+1} = F_i x_i + G u_i + G_d d_i, \tag{5}$$

where $F_i = I + TA(t)$, $G = TB$, $G_d = TB_d$, and T is the sampling time.

3 | DEEP NEURAL NETWORKS

Deep Neural Networks (DNNs) are part of the field of machine learning in artificial intelligence. These networks can learn from large amounts of data and are composed of multiple processing layers. Training a DNN involves applying specific techniques to automatically adjust the network's synaptic weights by minimizing a cost function. After training, the DNN will provide general solutions for the class of problems it has been trained for.

We can choose different algorithms that update the network weights to optimize the cost function minimization process. One of the most common, Mini-Batch Gradient Descent (MBGD), partitions the data set into batches, where the network weights are updated after each batch is computed [27]. In addition, it is essential to mention the momentum technique of momentum [28], which is used to speed up learning by making the gradient continue to decay in the same direction as the average decay of the previous steps [27]. The Equation 6 presents this technique.

$$v_{i+1} = \beta_1 v_i - (1 - \beta_1)\nabla J(\mathrm{W}), \tag{6}$$

where $0 \le \beta_1 < 1$ is responsible for preventing the speed at which the gradient decays from increasing too much and $\nabla J(\mathrm{W})$ is the gradient of the cost function $J(W)$ with respect to the weights.

In order to optimize the adjustment of DNN weights, especially when there is a large amount of data and parameters, several optimization algorithms have emerged based on stochastic mini-batch methods. The following are details of the optimization algorithms used to train the neural networks developed in this paper: RMSProp and Adam (Adaptive Moment Estimation) algorithm.

Unlike many other optimization algorithms, RMSProp (Equation 7) works well for non-convex functions. This algorithm accumulates only the gradient of the most recent iterations using the exponentially decaying moving average [27]. In this way, it converges faster to the global minimum point. It works from the following equation

$$s_{i+1}^{dw} = \beta_2 s_i^{dw} + (1-\beta_2)\nabla J(W) \otimes \nabla J(W), \tag{7}$$

where β_2 is the decay rate, usually set to a value of 0.9, and the $\otimes$ symbol stands for the element-wise multiplication.

The Adam algorithm can be thought of as a combination of RMSProp, and the momentum technique [29]. The main idea is to calculate adaptive learning rates for each network parameter under training. For this, the Equations 8 and 9 are applied, using the parameters β_1 and β_2, already presented.

$$v_{i+1} = \beta_1 v_i - (1-\beta_1)\,\nabla J(W), \tag{8}$$

$$s_{i+1}^{dw} = \beta_2 s_i^{dw} + (1-\beta_2)\nabla J(W) \otimes \nabla J(W), \tag{9}$$

where v is the first-order moment and s^{dw} is the second-order moment, both initialized to zero.

In addition to use suitable optimization algorithms for training a DNN, it is also important to use regularization techniques. These techniques are essential to ensure that the DNN performs well on the data set used for training and on a new data set. In the literature, one of the most used is a dropout, proposed by [30], which dynamically changes the topology of the DNN during training. Here, some neurons are inactivated according to a defined probability. The idea is that by turning off some units in the DNN, it becomes simpler and avoids the phenomenon of overfitting.

4 | DISTURBANCE ESTIMATION AND CONTROL

This section describes the DOBC approach used to combine a DNN with the controllers in one of the proposed architectures. Afterward, we present four control strategies tested on both architectures. The first control approach is a robust linear quadratic regulator, useful for online applications. The second approach is a recursive linear quadratic regulator, which follows the same structure as the previous robust controller, but does not consider parametric uncertainties associated with the dynamic model. Thus, we will introduce the RLQR controller to deal with the uncertainties of the system. The third approach is a classical PID controller. Finally, the last approach is a standard computed torque based on feedback linearization.

4.1 | Disturbance observer-based control

External disturbances are responsible for depreciating the performance and stability of control systems. In most cases, they cannot be directly measured. Consequently, it is challenging to implement a control technique to attenuate or eliminate such disturbances without adding sensors to the aircraft body. Due to this, several techniques for disturbance estimation have been proposed to generate a control action based on the estimated disturbance [31].

Although many control architectures for estimating and attenuating external disturbances have emerged, the fundamental idea of the operation is the same: create a disturbance observation mechanism and a compensator based on the obtained estimate [32]. These techniques can be found in the literature by the name of Disturbance and Uncertainty Estimation and Attenuation (DUEA) [31].

According to [33], one of the most widely used DUEA techniques is DOBC, mainly because of its intuitiveness and modular architecture. In addition, compared to other similar techniques, it provides a more rigorous analysis regarding stability and other system performance requirements [31]. Since its emergence, the DOBC technique has been used in a wide variety of systems, including aerospace and nuclear systems [34], servo motor control [35], and quadrotor control [36]. Recently, some works have proposed the possibility of using intelligent systems, such as artificial neural networks, to attenuate the influence of external disturbances in robotic systems, see for example [37–39].

Recall that we must implement a controller to ensure trajectory tracking before disturbance estimation and attenuation steps. In this regard, the following sections discuss the controllers implemented in this work.

4.2 | Controllers

This section presents the implemented controllers to form the proposed control architectures. These controllers were combined with DNNs for trajectory tracking of a quadrotor.

4.2.1 | Robust linear quadratic regulator

In [24], a Robust Linear Quadratic Regulator for discrete-time linear systems subject to parametric uncertainties was developed. This regulator was developed based on penalty parameters and regularized least squares, providing an algorithm that does not rely on tuning parameters in online applications. Consider the discrete-time linear system with parametric uncertainties in state space:

$$x_{i+1} = (F_i + \delta F_i)x_i + (G_i + \delta G_i)u_i; \quad i = 0, \dots, N, \tag{10}$$

where $x_i \in \mathbb{R}^n$ is the state vector, $u_i \in \mathbb{R}^m$ is the control input, $F_i \in \mathbb{R}^{nxn}$ and $G_i \in \mathbb{R}^{nxm}$ are nominal parameter matrices with appropriate system dimension, and N is an integer defining the number of interactions. The initial state x_0 is constant and known, and the matrices $\delta F_i \in \mathbb{R}^{nxn}$ and $\delta G_i \in \mathbb{R}^{nxm}$ are unknown uncertainty matrices, modeled as

$$[\delta F_i \;\; \delta G_i] = H_i \Delta_i [E_{F_i} \;\; E_{G_i}]; \quad i = 0, \dots, N, \tag{11}$$

being $H_i \in \mathbb{R}^{nxk}$, $E_{F_i} \in \mathbb{R}^{nxn}$, $E_{G_i} \in \mathbb{R}^{nxn}$ known matrices, determined mostly heuristically. $\Delta_i \in \mathbb{R}^{kxl}$ is an arbitrary matrix $(\Delta_i) \leq 1$. The RLQR is obtained by solving the following optimization problem:

$$\min_{x_{i+1}, u_i} \max_{\delta F_i, \delta G_i} \left\{ \tilde{J}_i^{\mu}(x_{i+1}, u_i, \delta F_i, \delta G_i) \right\}, \tag{12}$$

where J_i^u is the regularized quadratic cost function, defined as

$$\tilde{J}_i^{\mu}(x_{i+1}, u_i, \delta F_i, \delta G_i) =$$

$$\begin{bmatrix} x_{i+1} \\ u_i \end{bmatrix}^T \begin{bmatrix} P_{i+1} & 0 \\ 0 & R_i \end{bmatrix} \begin{bmatrix} x_{i+1} \\ u_i \end{bmatrix} + \Phi^T \begin{bmatrix} Q_i & 0 \\ 0 & \mu I \end{bmatrix} \Phi, \tag{13}$$

where $P_{N+1} \succ 0$, $Q \succ 0$, and $R \succ 0$ are known matrices and $\mu > 0$ is a fixed penalty parameter, responsible for ensuring that the equality of Equation 10 holds, and

$$\Phi = \left\{ \begin{bmatrix} 0 & 0 \\ I & -G_i - \delta G_i \end{bmatrix} \begin{bmatrix} x_{i+1} \\ u_i \end{bmatrix} - \begin{bmatrix} -I \\ F_i + \delta F_i \end{bmatrix} x_i \right\}.$$

According to [24] the RQLR is based on the solution $(x_{i+1}^T(\mu), u_i^*(\mu))$ of the $min - max$ optimization problem, in which the intent is to obtain the smallest state magnitude and the smallest control action magnitude for the worst-case parametric uncertainties, for each penalty parameter $\mu > 0$. The optimal solution can be found recursively and is given by Equations 14 and 15. Furthermore, when the penalty parameter $\mu \to \infty$, the robustness of the controller is achieved.

$$\begin{bmatrix} x_{i+1}^*(\mu) \\ u_i^*(\mu) \\ \tilde{J}_i^{\mu}(x_{i+1}^*(\mu), u_i^*(\mu)) \end{bmatrix} = \begin{bmatrix} I & 0 & 0 \\ 0 & I & 0 \\ 0 & 0 & x_i^*(\mu)^T \end{bmatrix} \begin{bmatrix} L_i, \mu \\ K_i, \mu \\ P_i, \mu \end{bmatrix} x_i^*(\mu), \tag{14}$$

$$\begin{bmatrix} L_i \\ K_i \\ P_i \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 & I & 0 \\ 0 & 0 & 0 & 0 & 0 & I \\ 0 & 0 & -I & \mathcal{F}_i^T & 0 & 0 \end{bmatrix}$$

$$\times \begin{bmatrix} P_{i+1}^{-1} & 0 & 0 & 0 & I & 0 \\ 0 & R_i^{-1} & 0 & 0 & 0 & I \\ 0 & 0 & Q_i^{-1} & 0 & 0 & 0 \\ 0 & 0 & 0 & \Sigma_i(\mu, \lambda_i) & \mathcal{I} & -\mathcal{G}_i^T \\ I & 0 & 0 & \mathcal{I} & 0 & 0 \\ 0 & I & 0 & -\mathcal{G}_i^T & 0 & 0 \end{bmatrix}^{-1} \begin{bmatrix} 0 \\ 0 \\ -I \\ \mathcal{F}_i \\ 0 \\ 0 \end{bmatrix}, \tag{15}$$

where

$$\Sigma_i = \begin{bmatrix} \mu^{-1} I - \hat{\lambda}_i^{-1} H_i H_i^T & 0 \\ 0 & \hat{\lambda}_i^{-1} I \end{bmatrix}, \qquad \mathcal{I} = \begin{bmatrix} I \\ 0 \end{bmatrix},$$

$$\mathcal{G}_i = \begin{bmatrix} G_i \\ E_{G_i} \end{bmatrix}, \; \mathcal{F}_i = \begin{bmatrix} F_i \\ E_{F_i} \end{bmatrix}.$$

Thus, the optimal control law is given by

$$u_i^* = K_i x_i^*, \qquad i = 0, \dots, N. \tag{16}$$

Details about the convergence and stability guarantee can be found in [24]. Recall that when parametric uncertainties are not considered, the framework falls into the form of recursive LQR.

4.2.2 | Proportional-integral-derivative

Most of the controllers in use in industries are of the PID type. Its wide use is directly related to the ease of application in the most varied control systems projects. Besides this, the PID control performs well in various operating conditions. However, in more critical situations, it may not offer satisfactory control of the system [40].

In order to formulate the PID controller, the control law must take as reference the error information obtained through the difference between the expected response and the actual response of the system $(\tilde{q} = q - q^d$. Thus, the control vector is given by

$$\nu_{(t)} = K_p \, \tilde{q}(t) + K_i \int_{t_0}^{t} \tilde{q}(\tau) d\tau + K_d \, \dot{\tilde{q}}(t), \tag{17}$$

where K_p is the proportional gain, K_i is the integral gain, and K_d is the derivative gain.

4.2.3 | Feedback linearization

The feedback linearization technique was initially proposed by [41]. Considered a precursor technique for the control of nonlinear systems, it has been helpful for the evolution of studies in this research line. The central idea of this technique is to algebraically transform nonlinear systems into fully, or partially linear systems [42].

For the quadrotor model used in this paper, we have the feedback linearization control law given by

$$\nu = \Gamma^{-1}(u - \Lambda R_t^T \dot{q}), \tag{18}$$

where $u = \ddot{q}^d - K_p \tilde{q} - K_d \dot{\tilde{q}}$, with q^d being the desired trajectory, $\dot{q}$ the state vector of the quadrotor and $\tilde{q}$ is the trajectory tracking error.

5 | INTELLIGENT CONTROL ARCHITECTURES

The two control architectures proposed to improve quadrotor trajectory tracking will be presented in the following. In both architectures, the goal is to combine a DNN with the controllers shown in Subsection 4.2. The DNN sends a feasible reference signal to the controller based on experience in the first control architecture. This technique was based on the work of [20], which combines six DNNs with a PID controller to control the six degrees of freedom of a quadrotor. In the second architecture, the DNN acts as a disturbance estimator in conjunction with the Kalman filter and the implemented controllers, forming a DOBC architecture. In this case, the DNN is responsible for estimating the wind disturbance affecting the quadrotor.

5.1 | DNN as a reference for controllers

This section discusses the operation of the first proposed control architecture, where the DNN has the function of sending a reference signal to the implemented controllers. In general, for position control in the trajectory tracking problem, the controller receives the desired trajectory and provides a control law that aims to minimize the position error of the quadrotor during the path. However, with the proposed architecture 1, the controller receives the reference trajectory from the DNN instead of the desired trajectory. The difference is that the reference signal provided by DNN is computed from the knowledge acquired during training, considering the current ($\bar{q}$) and desired position ($\bar{q}^d$) of the aircraft. In other words, the network can provide a signal to the controller to auxiliary in the minimization of the trajectory tracking error after training.

The main idea is that if the current trajectory is equal to the desired trajectory, the network should provide this stored reference value so that the tracking continues with as little error as possible. On the other hand, when the quadrotor's trajectory is different from the desired one, the mapping learned by the network will generate a feasible reference signal for the controller to correct the quadrotor's course and make it converge more quickly to the desired path. In this case, the signal provided by the network has a magnitude slightly different from the desired trajectory and may have a higher or lower amplitude.

Considering that the goal of DNN is to map a reference signal ($\bar{q}^r$) at each time instant and send it to the controller, the network is trained offline through the supervised learning method, in which details will be discussed in Section 6.4.1. After training, the acquired knowledge is applied in real-time with the controller, improving performance during trajectory tracking. We can visualize the operation of the proposed architecture together with the controllers in Figure 2. It is worth noting that when parametric uncertainties are not considered, the RLQR falls in the form of the LQR. From the figure, we can see that the network obtains information from both the trajectory generator (desired trajectory) and the optical motion capture system (current trajectory), which we will discuss in Section 6.5.

ALGORITHM 1 DNN as a Reference for Controllers

```
Require Desired trajectory $q^d$
Ensure Velocity vector $\nu$
1:  if the RLQR controller is selected then
2:      Consider Equations 10, 11, and 13 with $F_i$, $G_i$, $E_{F_i}$, $E_{G_i}$, $Q_i \succ 0$, and
        $R_i \succ 0$ known for every instant $i$.
3:  if the LQR controller is selected then
4:      Consider $F_i$, $G_i$, $Q_i \succ 0$, and $R_i \succ 0$ known for every instant $i$.
5:  if the PID controller is selected then
6:      Gets the $K_p$, $K_d$, and $K_i$ gain matrices.
7:  else if the FL controller is selected then
8:      Gets the $K_p$ and $K_d$ gain matrices.
9:  while receive the desired trajectory do
10:     Retain $q^d$ and $q$
11:     Calculate $\dot{q}^d$ and $\dot{q}$
12:     Compute $q^r$ and $\dot{q}^r$
13:     Calculate $\tilde{q}$, $\dot{\tilde{q}}$, and $\int \tilde{q}$
14:     if Using RLQR or LQR controller then
15:         Calculate the RLQR/LQR gain with Equation 15
16:         $u_i = K_i x_i$
17:         Uses $u_i$ in Equation 4 to generate $\nu$
18:     if Using the PID controller then
19:         $u_i = K_p\tilde{q} + K_d\dot{\tilde{q}} + K_i \int \tilde{q}$.
20:         Uses $u_i$ in Equation 4 to generate $\nu$
21:     else if Select the FL controller then
22:         $u_i = K_p\tilde{q} + K_d\dot{\tilde{q}}$
23:         Uses $u_i$ in Equation 18 to generate $\nu$
24: end while
```

The desired trajectory (Section 6.3) is formed by the vector $q^d = [q_x^d, q_y^d, q_z^d, q_\psi^d]$, where q_ψ^d is set to zero. For simplicity, the signal provided by the network acts on the x and y axes, keeping the z axis at the default value provided by the trajectory generator. We made this selection because we aim to reduce the influence of lateral wind gusts on the quadrotor in our experiments. Consequently, the z axis is less affected by the disturbance, and keeping its default value reduces the complexity of training the DNN. Thus, the network input is composed of the vector $\{(\bar{q}_i, \bar{q}_i^d)\}$, where $\bar{q}_i = [q_x, q_y, q_{\dot{x}}, q_{\dot{y}}]$ and $\bar{q}_i^d = [q_x^d, q_y^d, q_{\dot{x}}^d, q_{\dot{y}}^d]$ correspond to current and desired positions and velocities on the x and y axes, respectively. The generated output is composed of the vector $\bar{q}_i^r = [q_x^r, q_y^r, q_{\dot{x}}^r, q_{\dot{y}}^r]$, corresponding to reference positions and velocities.

It is also worth mentioning that in Figure 2, $\nu = [\nu_{\dot{x}_b}\ \nu_{\dot{y}_b}\ \nu_{\dot{z}_b}\ \nu_{\dot{\psi}}]$ represents the velocity vector sent to the quadrotor and d is the wind gust affecting the aircraft. In order to facilitate the understanding regarding the proposed architecture and complement the diagram presented in Figure 2, the Algorithm 1 presents the operating steps for all the controllers implemented.

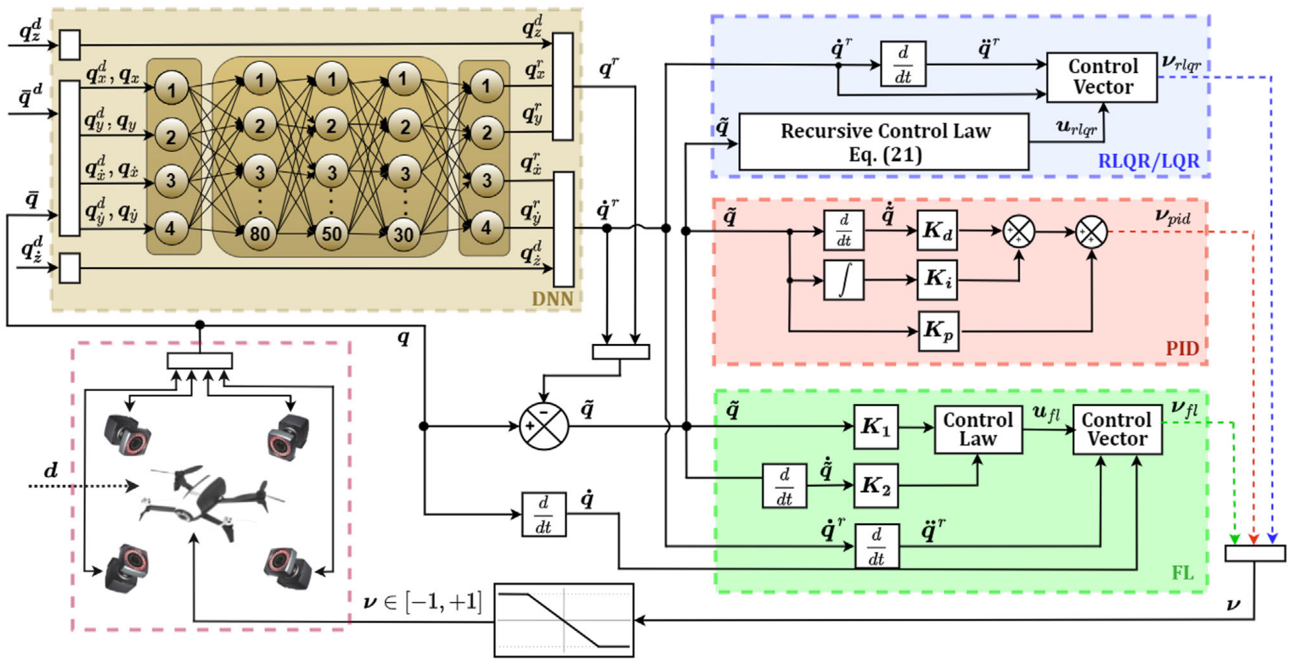

FIGURE 2 Deep Neural Network as a Reference for Controllers (Architecture 1).

5.2 | DNN as a disturbance estimator

This section discusses the working details of the second proposed control architecture, where the DNN is responsible for estimating the external disturbances. For this case, the DOBC (Subsection 4.1) architecture was used with the controllers mentioned in Subsection 4.2. Here, the controllers are responsible for trajectory tracking, the DNN for disturbance estimation, and a compensator was designed to attenuate the estimated disturbance.

The main objective of DNN is to provide an estimate of the disturbance in the x and y axes from the trajectory error $\tilde{q}_x$ and $\tilde{q}_y$, respectively. To this end, we created a data set in which wind gusts with different magnitudes were applied to the quadrotor. The position error generated by these disturbances was stored with the respective label (Force in Newtons of the disturbance). During operation, the disturbance signal provided by the DNN is filtered and directed to a compensator. Subsequently, a composite control law is generated with the ability to attenuate the wind disturbances affecting the aircraft.

Figure 3 presents the operation of the proposed architecture. Similar to Figure 2, each controller is used individually with the proposed neural network. We calculated the position error using the current and desired position at each instant. For practical experiments, the optical motion capture system (Section 6.5) provided a current position at each instant. After the error values pass through the network, the Kalman filter smoothes the generated disturbance signal. Then the compensator (gray block) is responsible for attenuating the disturbance, forming the composite control law.

The following subsections present the filtering and compensation steps of the estimated disturbance. The composite control law, which uses both the control signal coming from the controller and the estimated disturbance, will be presented too.

5.3 | Kalman filter

The Kalman filter (KF) [43] can be defined as a set of important mathematical tools capable of providing an estimate of process variables through the minimization of the mean square error. By using the augmented state technique, the Kalman filter can provide the filtered form of states and external disturbances affecting a system [44]. For this, an augmented state vector is constructed, where the disturbance d and its respective derivative are arranged together with the state vector x, forming $s_{i+1} = [x_{i+1} \quad d_{a_{i+1}} \quad d_{b_{i+1}}]^T$, in which $d_{a_{i+1}}$ and $d_{b_{i+1}}$ correspond to the external disturbance in discrete-time and its respective derivative. Thus, for a discrete-time model, the system in augmented form is given by Equation 19.

$$\underbrace{\begin{bmatrix} x_{i+1} \\ d_{a_{i+1}} \\ d_{b_{i+1}} \end{bmatrix}}_{s_{i+1}} = \underbrace{\begin{bmatrix} F_i & G_{d,i} & 0 \\ 0 & I & TI \\ 0 & 0 & I \end{bmatrix}}_{\Phi_i} \underbrace{\begin{bmatrix} x_i \\ d_{a_i} \\ d_{b_i} \end{bmatrix}}_{s_i} + \underbrace{\begin{bmatrix} G_i \\ 0 \\ 0 \end{bmatrix}}_{\Xi_i} u_i,$$

$$z_i = \underbrace{\begin{bmatrix} I & 0 & 0 \\ 0 & I & 0 \\ 0 & 0 & 0 \end{bmatrix}}_{\Omega_i} \underbrace{\begin{bmatrix} x_i \\ d_{a_i} \\ d_{b_i} \end{bmatrix}}_{s_i} + v_i,$$

where F_i, G_i, and $G_{d,i}$ comes from discrete-time system (5) presented. In this way, the filtered forms of the states and

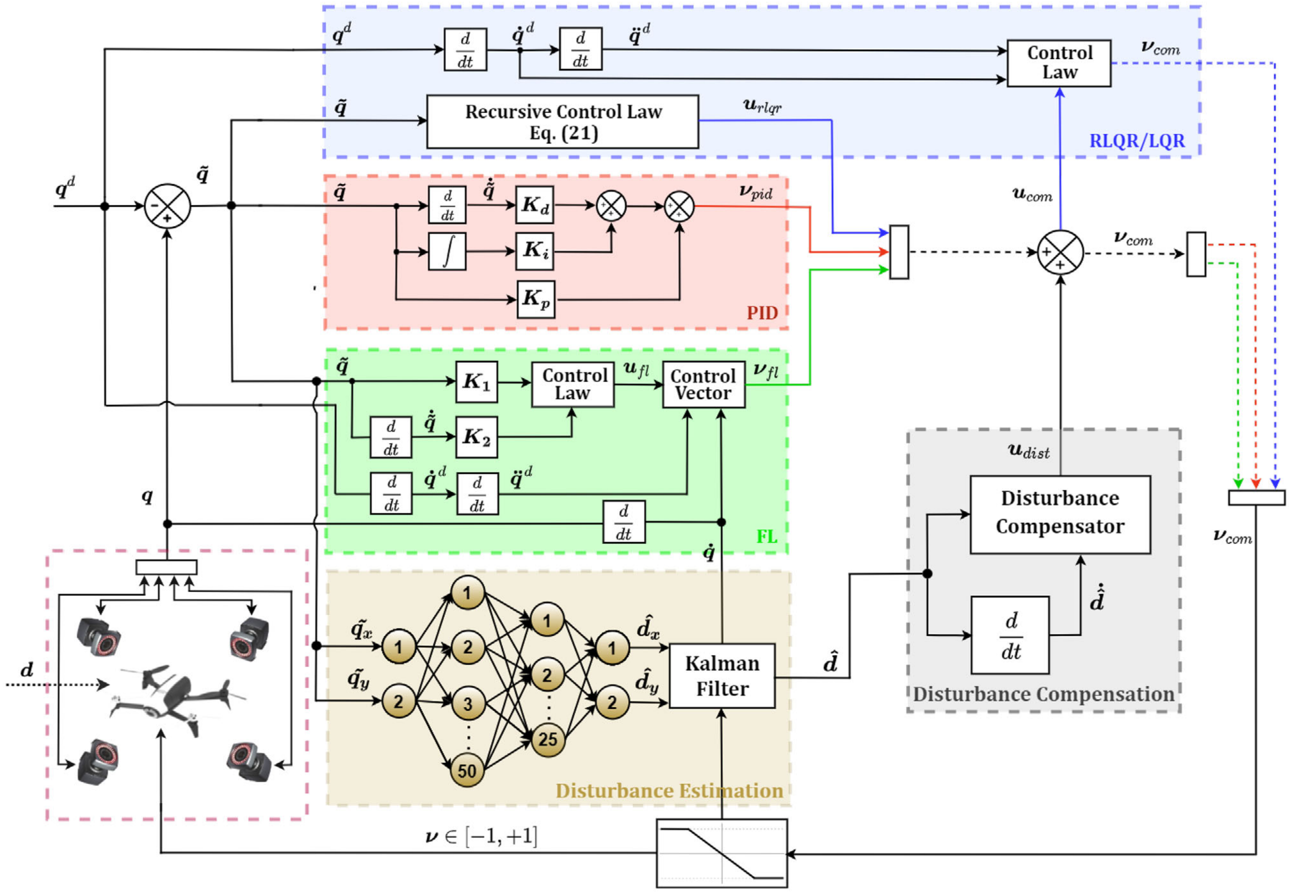

FIGURE 3 Disturbance observer-based control with neural network and Kalman filter (Architecture 2).

the disturbance estimated by DNN are obtained. Next, the estimated disturbance goes through the compensation stage, where a compensator is responsible for attenuating its effects on the system.

5.4 | Disturbance attenuation

With the filtered estimate of the disturbance provided by DNN, a compensator should be designed to attenuate or reject the influence of the real disturbance on the control system. According to [31] and [21], for the case where a plant is affected by wind disturbances or even by torques caused by unmodeled dynamics, we can get a compensator by considering the control law. Using the RLQR as an example with $u_{rlqr,i} = -K_{rlqr,i}x_i$, where K_{rlqr} is the recursive optimal gain, and x is the state vector, the composite control law is formed by:

$$u_{com,i} = u_{rlqr,i} + u_{dist,i}, \tag{20}$$

where $u_{dist,i} = K_{com}\hat{d}_i$ is the disturbance compensation control law. Using the composite control law u_{com} with the system (5) discretized, and considering that the $\hat{d}$ estimate will converge to the value of the actual disturbance, it is possible to find a gain K_{com} that eliminates the disturbance at the output (y) of the system.

$$K_{com,i} = -\left[C(F_i - G_iK_{rlqr,i})^{-1}G_i\right]^{-1} \times C(F_i - G_iK_{rlqr,i})^{-1}G_{d,i}. \tag{21}$$

According to [45], remotely controlled real-time systems may be affected by communication delays, resulting in a performance loss. In this case, it is possible to anticipate the action of the disturbance compensator using its derivative. This way, a proportional-derivative control compensation law is generated, as described by Equation 22.

$$u^{pd}_{dist,i} = \alpha K_{com,i}\hat{d}_i + \varrho\,\hat{d}_{i+1}, \tag{22}$$

where $|\,\alpha\,| < 1$ and ϱ are scalars, defined as design parameters. With this, the compound control law (20) is rewritten as $u_{com,i} = u_{rlqr,i} + u^{pd}_{dist,i}$. The general operation of the proposed architecture 2 can be summarized by the Algorithm 2.

6 | METHODOLOGY

In this section, we present the methodology and tools used to develop this work. We developed the entire architecture using a

ALGORITHM 2 DNN as a Disturbance Estimator

```
Require Desired trajectory $q^d$
Ensure Velocity vector $\nu$
1:  if the RLQR controller is selected then
2:      Consider Equations 10, 11, and 13 with $F_i, G_i, E_{F_i}, E_{G_i}, Q_i \succ 0$, and $R_i \succ 0$ known for every instant $i$.
3:  if the LQR controller is selected then
4:      Consider $F_i, G_i, Q_i \succ 0$, and $R_i \succ 0$ known for every instant $i$.
5:  if the PID controller is selected then
6:      Gets the $K_p$, $K_d$, and $K_i$ gain matrices.
7:  else if the FL controller is selected then
8:      Gets the $K_p$ and $K_d$ gain matrices.
2:  while receive the desired trajectory do
3:      Retain $q^d$ and $q$
4:      Calculate $\dot{q}^d$ and $\dot{q}$
5:      Compute $\tilde{q}$ and $\dot{\tilde{q}}$, and $\int \tilde{q}$
14:     if Using RLQR or LQR controller then
15:         Calculate the RLQR/LQR gain with Equation 15
16:         $u_i = K_i x_i$
18:     if Using the PID controller then
19:         $u_i = K_p\tilde{q} + K_d\dot{\tilde{q}} + K_i \int \tilde{q}$.
21:     else if Select the FL controller then
22:         $u_i = K_p\tilde{q} + K_d\dot{\tilde{q}}$
8:      Estimate $d$ with the DNN
9:      Compute the filtered form of $d$
10:     Calculate $\dot{d}$
11:     $u^{pd}_{dist,i} = \alpha K_{com,i}\hat{d}_i + \varrho\,\hat{d}_{i+1}$
12:     $u_{com,i} = u_{rlqr,i} + u^{pd}_{dist,i}$
18:     if using the RLQR, LQR, or PID controller then
19:         Uses $u_{com}$ in Equation 4 to generate $\nu$
21:     else if Select the FL controller then
22:         Uses $u_{com}$ in Equation 18 to generate $\nu$
14: end while
```

Dell Inspiron 14 7460 notebook with Ubuntu 16.04 LTS operating system. The hardware used has an Intel Core i7-7500U processor with 2 cores of 2.7 GHz, 16GB DDR4 RAM, and Nvidia GPU Geforce 940MX.

6.1 | Parrot Bebop 2.0

The quadrotor chosen for practical experiments was the Parrot™ Bebop 2.0. Besides being a low-cost quadrotor with a high-resolution camera, the Bebop 2.0 has the facility to communicate with the ROS platform through the `bebop_autonomy` package. This ROS package allows sending control commands and receiving pose estimation from Bebop 2.0. The model parameters used for the experiments were $\gamma = [4.5922, 0.4349, 5.2157, 0.4364, 4.6026, 3.0992, 5.9388\ {-0.3980}]^T$.

6.2 | Robot operating system

The ROS [46] is an open-source platform with a series of tools that enables the programming of different types of robots. The operation of ROS is based on integrating different nodes that carry information about the robotic system. These nodes can represent, for example, sensors or actuators whose messages generated by each of them are published in topics accessible to other nodes. This way, it is possible to read the information received by sensors and send commands to the actuators easily. One of the main advantages of ROS is the possibility of using several programming languages, such as Python, C++, Java, and LISP. This advantage allowed implementing the controllers in C++ and developing the neural networks in Python using specific libraries.

6.3 | Desired trajectory

The desired trajectory used in the experiments has a circular shape in the xyz plane, with a radius of 0.3 m. Sending the trajectory to the quadrotor is done through a ROS node, which is responsible for publishing in the topic `/bebop/waypoint` both the position and the desired velocity at a frequency of 50 Hz. This topic, which provides $q^d = [q^d_x, q^d_y, q^d_z, q^d_\psi]^T$ and $\dot{q}^d$, is accessible to both DNN and controllers, being:

$$q^d = \begin{bmatrix} \varphi\cos\omega t & \varphi\sin\omega t & \frac{\varphi}{2}\sin\omega t & 0 \end{bmatrix}^T, \tag{23}$$

$$\dot{q}^d = \begin{bmatrix} -\omega\varphi\sin\omega t & \omega\varphi\cos\omega t & \frac{\omega\varphi}{2}\cos\omega t & 0 \end{bmatrix}^T, \tag{24}$$

where $\omega = 0.5$ rad/s is the angular velocity, given by the ratio between the average velocity $v = 0.15$ m/s and the radius $\varphi = 0.3$ m. For each experiment, the quadrotor starts from the origin ($x = 0, y = 0, z = 0$) and performs the proposed trajectory.

6.4 | Training and evaluation

This section discusses details regarding the training and evaluation of the developed DNNs. It is worth mentioning that, to develop the presented DNNs, the Python language was used through the framework Keras with backend in Tensorflow.

6.4.1 | DNN as a reference for controllers

The DNN architecture that acts as a reference for the controllers can be defined as a multi-layer network. In this case, the network architecture comprises an input layer, three hidden layers containing 80, 50, and 30 neurons, respectively, and an

TABLE 1 DNN training parameters (Architecture 1).

Activation function	ReLU
Loss function	Mean squared error (MSE)
Optimization algorithm	RMSprop
Learning rate	0,008
Dropout	25%
Weight-initialization	Xavier [47]
Biases initialization	0
Max. number of epochs	30
Batch size	32

output layer. The number of layers and neurons present in each was chosen empirically. To this end, we conducted a series of training sessions in which the DNN was adjusted and optimized based on the performance presented for the assigned task. This way, it was possible to achieve the architecture presented in Figure 2.

For DNN training, we use the supervised learning technique, in which a set of labeled samples was necessary. We perform flights with the quadrotor to create the data set using the pre-tuned implemented controllers separately. The quadrotor performed a circular trajectory in the xyz plane with the radius ranging from 0.2 m to 1.0 m. Thus, we trained four DNNs, where, after training, each one acted independently in conjunction with the respective controller used in training (RLQR, LQR, PID, and FL). From the flights performed, it was possible to obtain data samples with information about both the current and desired trajectory of the quadrotor at each instant of time. We obtain 10,000 samples to compose the data set and use 90% for the training stage and 10% for the testing stage.

After obtaining the samples, we proceeded to create the network training set, which was formed by $\{(\bar{q}_i, \bar{q}_{i+1}),\ \bar{q}_i^d\}$, where $\bar{q}_i$ and $\bar{q}_{i+1}$ represent the position of the quadrotor at time i and $(i+1)$, respectively; and $\bar{q}^d$ represents its corresponding output data (label). Note that $\{(\bar{q}_i, \bar{q}_{i+1})\} \rightarrow \bar{q}_i^d$, used in training, is an approximate mapping of $\{(\bar{q}_i, \bar{q}_i^d)\} \rightarrow \bar{q}_i^r$ (Figure 2). Thus, when the current trajectory of the quadrotor is equal to the desired trajectory, the DNN understands that the tracking is perfect and $\bar{q}_i^r$ will be equal to $\bar{q}_i^d$. However, when the current trajectory of the quadrotor is different from the desired trajectory, the network provides a reference signal based on past experience in order to make $\bar{q}_i$ converge to $\bar{q}_i^d$. The parameters used for training are presented in Table 1.

The ReLU activation function was chosen, above all, because it is one of the most efficient in training deep neural networks. One of its significant advantages is its ability to avoid the disappearance of the gradient. In other words, it prevents the gradient from assuming values very close to zero, a situation that would make it difficult to adjust the synaptic weights of the DNN. Furthermore, compared with other activation functions, better network performance was verified with the ReLU function.

Regarding the chosen optimization algorithm, besides the used RMSprop, the stochastic gradient descent and adaptive moment estimation (Adam) were also tested. After a comparative analysis, for the task performed by the network in this work, it was noted a better performance after training with the RMSprop algorithm. Regarding the regularization technique, we observe the better performance of the network with the application of dropout of 25%. Also, after training with different epochs to avoid an underfitting and overfitting model, we noticed that the trained model performed well with 30 epochs. Furthermore, the remaining parameters used in the network training were adjusted empirically to find the best network performance without over-adjusting the synaptic weights.

With the training parameters set (Table 1), the DNN took about 4.000 iterations to converge. For evaluation, the metric used was the Mean Absolute Error (MAE), where the absolute value of the difference between the network's prediction and the set of labeled samples was calculated. Then, after training, the DNN was evaluated with the test data set and from experimental flights using the quadrotor, presenting a satisfactory performance for the task treated in this work.

6.4.2 | DNN as a disturbance estimator

As well as the DNN used as a reference for the controllers, the DNN for disturbance estimation has a multi-layer neural network architecture. In addition to the input and output layers, the network is formed by two hidden layers with 50 and 25 neurons, respectively. The network topology, shown in Figure 3, was obtained empirically, considering the disturbance estimation task.

We use the supervised learning method to train the network. In order to make the DNN capable of providing the wind disturbance in Newton that affects the quadrotor, we use the position error as a network input. A total of 2000 samples were obtained to make the data set, with 90% and 10% used for the training and testing steps.

The Parrot-Sphinx simulator supported by the BebopS package was used to create the data set. This simulator allows applying the wind disturbance to the quadrotor in the format of a force in Newton. We kept the quadrotor hovering in the air and applied a wind disturbance with different magnitudes to the aircraft to compute the position error. First, the disturbance with magnitudes ranging from 0.1 N to 0.5 N was applied along the X axis of the quadrotor. Then, the same procedure was performed for the Y axis of the aircraft. In this way, it was possible to obtain the position error of the quadrotor in the X and Y axes from the intensity of the wind disturbance affecting it. This force magnitude was chosen so that all of the controllers could still guarantee trajectory tracking. For this case, Table 2 presents the training parameters used.

Regarding the optimization algorithm chosen, for this case, the adaptive moment estimation (Adam) demonstrated better performance, providing better answers than the stochastic gradient descent and the tested RMSProp. Regarding the activation function, the ReLU function also demonstrated better responses for the task performed by the network. As with the

TABLE 2 DNN training parameters (Architecture 2).

Activation function	ReLU
Loss function	Mean squared error (MSE)
Optimization algorithm	Adam
Learning rate	0,001
Weight-initialization	Xavier [47]
Biases initialization	0
Max. number of epochs	30
Batch size	32

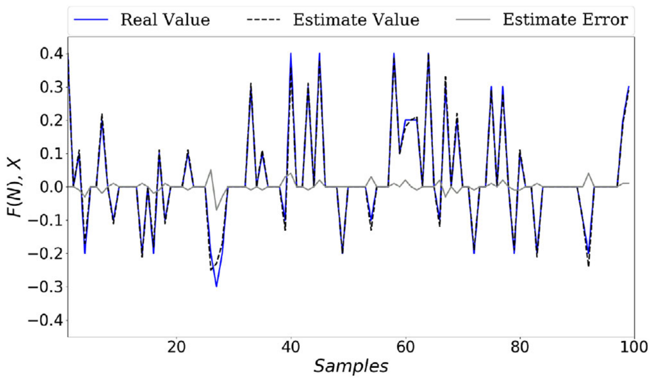

FIGURE 4 Estimated disturbance in X axis.

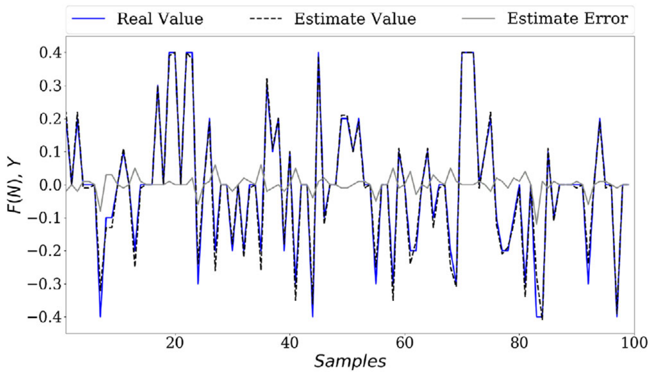

FIGURE 5 Estimated disturbance in Y axis.

first DNN, we empirically adjust the other parameters used in training to find the best neural network topology.

During training, it was found that the DNN took about 1.600 iterations to converge. The Mean Absolute Error (MAE) was used to calculate the difference between the network's prediction and the set of labeled samples. After training, the DNN was evaluated with the test data set and a series of experimental flights using the quadrotor in both the simulated and real environment. In order to show the convergence of the disturbance to the real value, Figures 4 and 5 show the real estimated disturbance curves and the estimated error in the X and Y axis of the quadrotor after training.

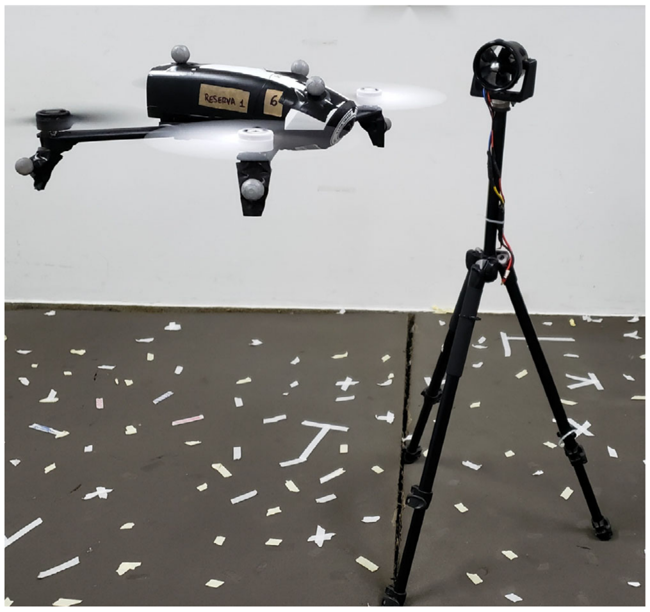

FIGURE 6 Parrot Bebop 2.0 flying under wing gust in our experimental environment.

6.5 | Vicon motion capture system

The Vicon® motion capture system is used to estimate the position and orientation of people or objects using the infrared light emission technology. In general, this system is composed of four specific cameras, and its operation is done by installing reflective markers on the body of the object to be tracked. The system emits infrared light reflected by the markers present on the object and detected by the cameras. In this work, Vicon® cameras are responsible for generating a reference system for Bebop 2.0. The pose data are sent to a specific software, the tracker 3[1], which track objects and provides 6 degrees of freedom data with low latency. The software sends the information at 100 Hz using a Wi-Fi communication protocol to the control system's laptop, and it is available in the topic `/Vicon/bebop/bebop`.

6.6 | Wind gusts source

In order to obtain actual results, an experimental environment was set up by creating a wind disturbance generator. A brushless motor driven by an Electronic Speed Controller (ESC) and powered by a LiPo battery was used. Figure 6 shows Bebop 2.0 flying under wind gusts in our experimental environment. Also, a demo video of the experiment can be found at https://youtu.be/e9EC-k0ooWI.

To measure the wind speed that affects the quadrotor during the trajectory tracking, we used an Incoterm® portable anemometer. Thus, a wind map was synthesized with the respective speeds generated by the disturbance system setup.

[1] https://shorturl.at/BD026

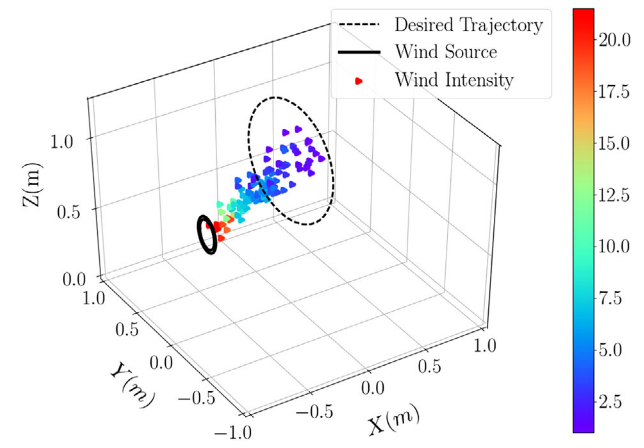

FIGURE 7 Wind disturbance map.

This map was done with the aid of the Vicon® motion capture system and presented in Figure 7. The collected wind intensity points made it possible to visualize a maximum wind speed of approximately 20.5 m/s and a minimum of 1.0 m/s.

6.7 | Controller parameters

This section presents parameters used for each of the implemented controllers. It is essential to mention that all the parameters used were chosen to perform the best tracking performance with each controller. With the RLQR, the weighting and uncertainty matrices were heuristically obtained as $Q = diag(1.2,\ 1.2,\ 1,\ 1,\ 1,\ 1,\ 5.0,\ 5.0)$, $R = 0.15I_{4\times4}$, $\mu = 10^{15}$, $P = I_{8\times8}$, $E_G = 0.01875I_{4\times4}$, and

$$E_F = \begin{bmatrix} 0.0354I_{2\times2} & 0_{2\times2} & 0.0285I_{2\times2} & 0_{2\times2} \\ 0_{2\times2} & 0.0295I_{2\times2} & 0_{2\times2} & 0.1425I_{2\times2} \end{bmatrix}.$$

For disturbance compensation with RLQR, we use $\alpha = 0.25$ and $\varrho = 0.01$.

With the LQR, the weighting matrices were chosen with the aid of Bryson's rule and experimental flights, where $Q = diag(1.2,\ 1.2,\ 1,\ 1,\ 1,\ 1,\ 5.0,\ 5.0)$, $R = 0.15I_{4\times4}$, and $P = I_{8\times8}$. For disturbance compensation with LQR, we use $\alpha = 0.4$ and $\varrho = 0.001$.

For the PID controller, the proportional, derivative, and integral gains were tuned heuristically as $K_p = diag(0.35,\ 0.35,\ 3.0,\ 1.0)$, $K_d = diag(0.4,\ 0.4,\ 0.4,\ 1.0)$ and $K_i = diag(0.001,\ 0.0001,\ 0.01,\ 0.01)$. For disturbance compensation with PID, we use $\alpha = 0.06$ and $\varrho = 0.00025$.

With respect to the FL controller, the proportional and derivative gains are heuristically tune as $K_p = diag(1.25,\ 1.25,\ 3.5,\ 1.0)$ and $K_d = diag(1.1,\ 1.1,\ 2.0,\ 2.0)$. For disturbance compensation with FL, we use $\alpha = 0.05$ and $\varrho = 0.001$.

6.8 | Metrics for trajectory tracking

In order to statistically analyze the performance of the proposed architecture for each of the implemented controllers, we use the ℓ_1 norm and the mean absolute error for the controlled position variables. It is worth mentioning that, for higher data accuracy, five experiments were performed for each of the controllers implemented in each of the architectures. Thus, the metrics below correspond to the average of the five experiments performed, where:

$$E(i) = \frac{1}{5}\sum_{j=1}^{5} \|q_i - q_i^d\|, \tag{25}$$

the ℓ_1 norm, and

$$\overline{E}(i) = \frac{1}{N}\sum_{i=1}^{N} E(i), \tag{26}$$

the mean absolute error, where q_i and q_i^d are the position and desired position vectors, respectively; j represents the current experiment number for a total of five; i corresponds to the value of an iteration with N being the maximum number of iterations.

In order to calculate the percentage improvement of the proposed architectures concerning the standalone controllers, we use the following percentage index:

$$I^{\%} = \left(1 - \frac{\overline{E}}{\overline{E}_{ref}}\right) \times 100\,, \tag{27}$$

where $\overline{E}$ is the mean absolute error for each controller in each of the proposed architectures, and $\overline{E}_{ref}$ is the mean absolute error for the standalone controllers.

7 | EXPERIMENTAL RESULTS

This section presents trajectory tracking results when the quadrotor is affected by wind gusts. Based on experimental flights, we provide a comparative study among the controllers related to the architectures proposed.

Figures 8, 9 and 10 present the results of real flights using the controllers when the aircraft is under the influence of wind gusts. In this sense, the 3D view of the trajectory tracking is presented in Figure 8 and the controlled variables (x, y, z, ψ) are presented in Figure 9. The error evolution for this case is exposed in Figure 10, and is complemented by the performance indices presented in Table 3.

For this case, we can see that FL obtained the highest trajectory tracking error among the implemented controllers, performing below the other controllers. Moreover, it was responsible for the most significant maximum error after 3 s of flight elapsed. On the other hand, the RLQR controller outperformed the other controllers in the trajectory tracking task, providing a lower position error.

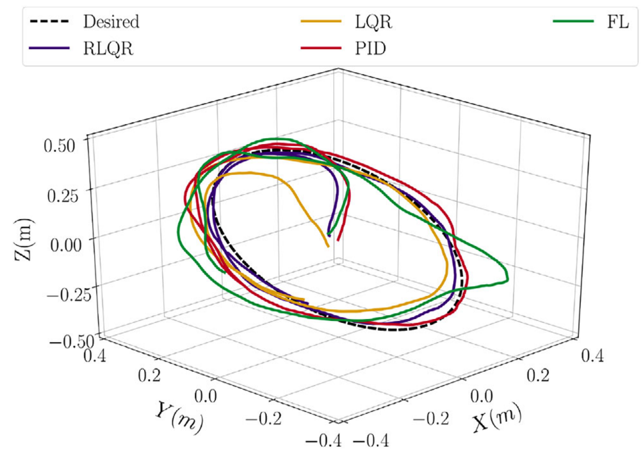

FIGURE 8 3D view of trajectory tracking with standalone controllers.

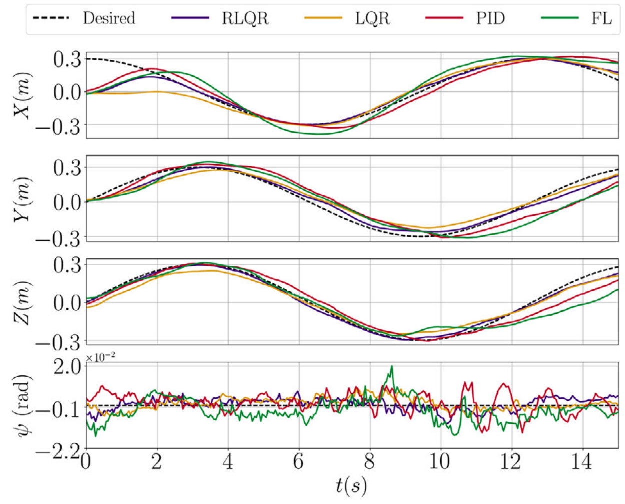

FIGURE 9 Controlled variables (x, y, z, and ψ) along time with standalone controllers.

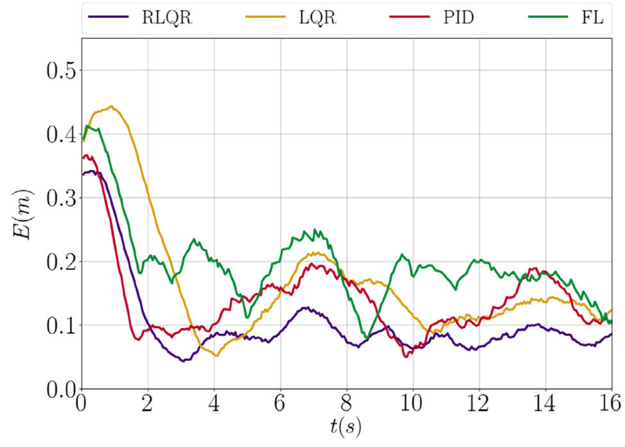

FIGURE 10 Error (ℓ_1 norm) evolution along time for trajectory tracking with standalone controllers.

TABLE 3 Performance metrics using standalone controllers.

	Standalone controllers			
Metrics	**RLQR**	**LQR**	**PID**	**FL**
MAE, $\overline{E}$ (m)	0.0842	0.1344	0.1337	0.1712
Peak Error (m)	0.1282	0.2139	0.1967	0.2500

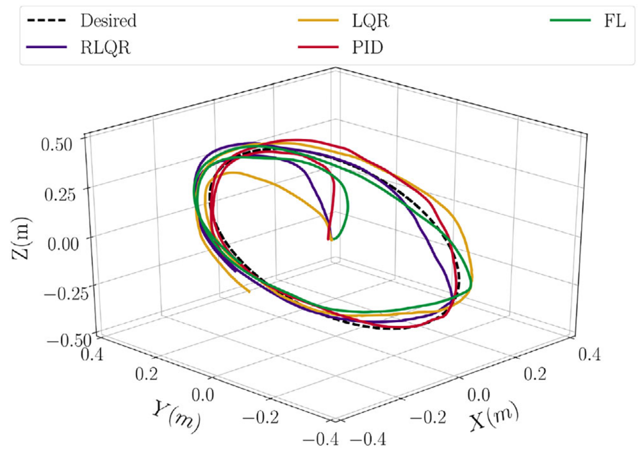

FIGURE 11 3D view of trajectory tracking with DNN as a reference for the controllers.

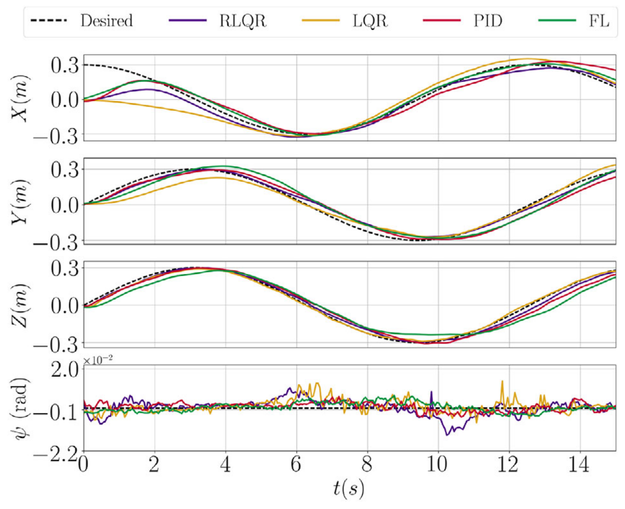

FIGURE 12 Controlled variables (x, y, z, and ψ) along time with DNN as a reference for the controllers.

According to Figure 10, we can see that the RLQR outperforms the other controllers in terms of position errors to the desired trajectory. From the numerical indices presented in Table 3, we can observe similar performance between the LQR and the PID controller.

The curves presented in Figures 11, 12 and 13 represent the results of real flights using architecture 1, where the DNN is responsible for assisting the controller during trajectory tracking. Given this, the 3D view is presented in Figure 11 and the controlled variables (x, y, z, ψ) are presented in Figure 12. For

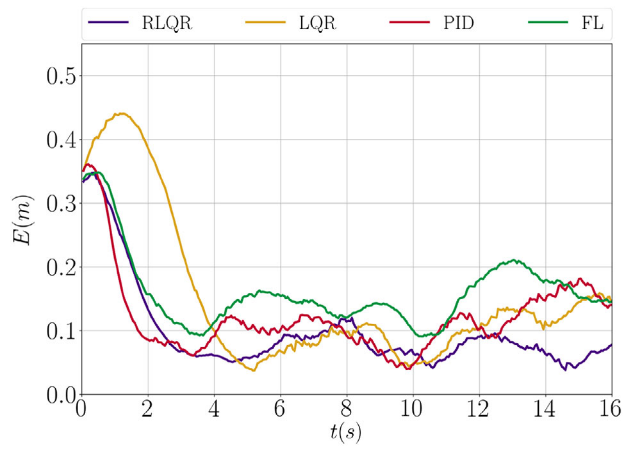

FIGURE 13 Error (ℓ_1 norm) evolution along time for trajectory tracking with DNN as a reference for the controllers.

TABLE 4 Performance metrics using DNN as a reference for the controllers (Architecture 1).

	DNN as reference (Architecture 1)			
Metrics	**RLQR**	**LQR**	**PID**	**FL**
MAE, $\overline{E}$ (m)	0.0728	0.1115	0.1121	0.1458
Peak error (m)	0.1203	0.1493	0.1820	0.2109
Improvement, $I^{\%}$	13.54%	17.04%	16.15%	14.84%

this case, the error evolution is evidenced in Figure 13 and the numerical performance indices are presented in Table 4.

Analyzing the results presented for the proposed architecture 1, we can observe that, as for the first case investigated (standalone controllers), the RLQR obtained both the lowest trajectory tracking error and the lowest maximum error after 3 s of flight elapsed. In addition, it is again possible to visualize a similar performance between the LQR and PID, with the LQR controller being slightly better than the PID controller in this case. As expected, the performance of the FL controller was considerably lower than the other controllers, especially concerning the RLQR controller. Notice according to Figure 13, the RLQR, PID, and FL controllers showed similar convergence times to the desired trajectory while the LQR, again, showed a longer convergence time.

In order to analyze the proposed architecture, Table 4 presents the percentages of improvement of each controller combined with DNN about the standalone controllers. Given this, there is an improvement by all controllers in the trajectory tracking task. With architecture 1, it was possible to improve further the RLQR controller's performance, which had already demonstrated its effectiveness when it is working in the standalone form. Regarding the improvement percentages, we can see that the RLQR controller obtained the highest percentage among the evaluated controllers, with 17.04%. It performed trajectory tracking with an error below its standalone version.

Next, Figures 14–16 present the results of real flights using the DOBC architecture with neural network acting as a dis-

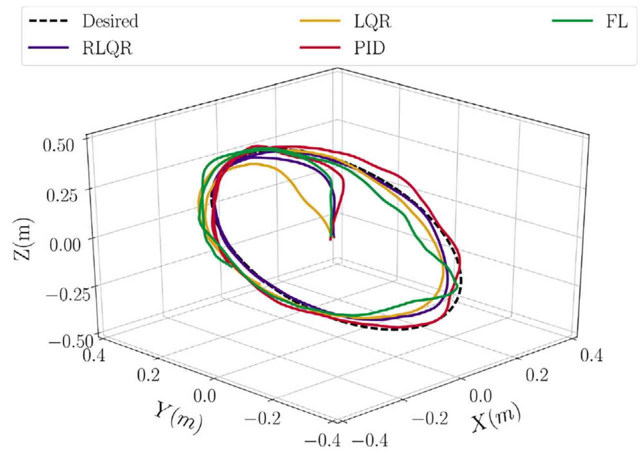

FIGURE 14 3D view of trajectory tracking with DNN as a disturbance observer in DOBC architecture.

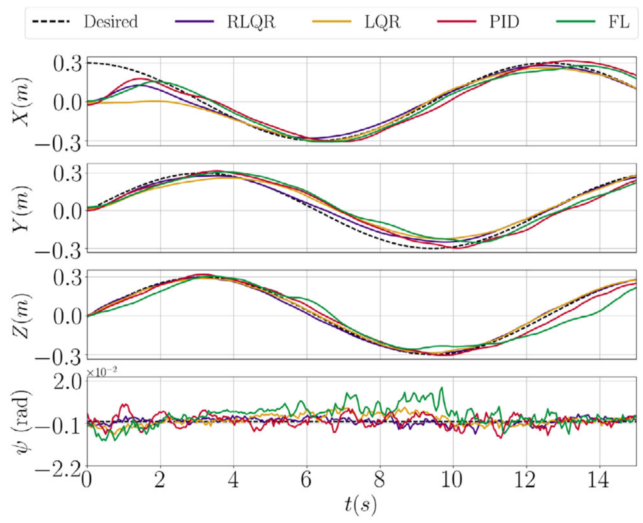

FIGURE 15 Controlled variables (x, y, z, and ψ) along time with DNN as a disturbance observer in DOBC architecture.

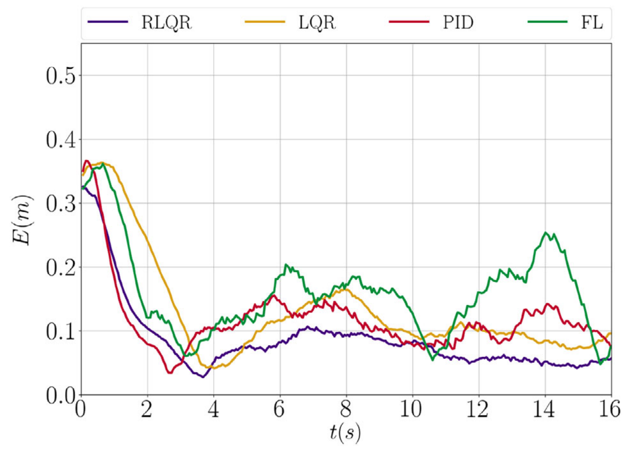

FIGURE 16 Error (ℓ_1 norm) evolution along time for trajectory tracking with DNN as a disturbance observer in DOBC architecture.

TABLE 5 Performance metrics using DNN as a disturbance observer in DOBC architecture (Architecture 2).

	DNN as a disturbance observer (Architecture 2)			
Metrics	**RLQR**	**LQR**	**PID**	**FL**
MAE, $\overline{E}$ (m)	0.0679	0.1062	0.1089	0.1401
u_{com} (%)	u_{rlqr} : 63.81	u_{lqr} : 65.01	u_{pid} : 67.65	u_{fl} : 75.31
	u_{dist} : 36.19	u_{dist} : 34.99	u_{dist} : 32.35	u_{dist} : 24.69
Peak error (m)	0.1069	0.1659	0.1557	0.2535
Improvement, $I^{\%}$	19.36%	20.98%	18.55%	18.17%

turbance estimator. The 3D view of the trajectory tracking is shown in Figure 14, followed by the Figure 15, which presents the controlled variables (x, y, z, ψ). The error evolution is displayed in Figure 16 and, for a quantitative analysis, evaluated performance indices are presented in Table 5.

Analyzing these results, we can see that the FL controller showed the worst performance among the evaluated controllers. As it was expected, the RLQR controller showed the best performance, keeping the quadrotor close to the trajectory during the entire time of the experiment. In addition, it showed a similar convergence time to the PID controller and the lowest maximum error after 3 s of flight, contributing to the good performance of the controller.

Again, the overall results showed a closeness between the LQR and PID controllers for practical experiments with wind disturbance. However, with the proposed architecture 2, the LQR controller converged faster to the desired trajectory when compared to its standalone version and architecture 1. The PID controller continues to be the controller that converges faster to the desired trajectory when compared to the other controllers.

Regarding the DOBC architecture, in which the network acts as a disturbance estimator (Architecture 2), the results showed that it could improve all controllers' performance. Thus, it reduced the position error and the maximum error of the evaluated controllers. From Table 5, improvement percentages around 18% to 20% can be observed for each controller.

The composition of the control signal can be seen in Figure 17 together with Table 5. In this way, we can observe the numerical indices regarding the composite control law for each controller. Notice that for each controller, the composition of the control law is different. In this case, the RLQR presented the strongest influence of the control law coming from the disturbance, followed by LQR and PID. On the other hand, the FL controller was the least influenced by the disturbance control law, with 24.69%.

The box-plot for comparing trajectory tracking performance is shown in Figure 18, where we can see that both architectures provide superior performance when compared to using the standalone controllers.

Furthermore, as architecture 2 focuses on estimating and attenuating external disturbances, it is possible to visualize a superior performance for all controllers implemented with this architecture. For quantitative analysis, the percentage of

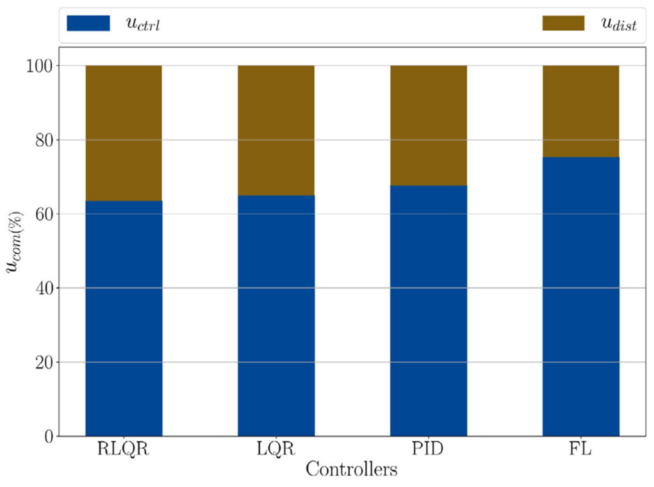

FIGURE 17 Control law composition for each implemented controller in the DOBC architecture.

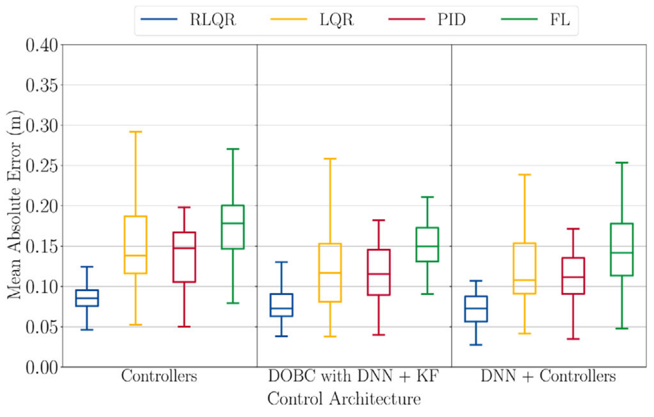

FIGURE 18 Box-plot of the trajectory tracking performance of all implemented controllers in each architecture.

TABLE 6 Percentage of improvement for the two proposed control architectures compared to standalone controllers.

Control	**Percentage of improvement ($\overline{E}$)**			
	RLQR	**LQR**	**PID**	**FL**
Architecture 1	13.54%	17.04%	16.15%	14.84%
Architecture 2	19.36%	20.98%	18.55%	18.17%

improvement of the two proposed control architectures in relation to the standalone controllers is presented in Table 6. Thus, it is possible to confirm that both control architectures improved the quadrotor performance in the trajectory tracking task.

In order to analyze the computational cost, Table 7 presents the average computation time required for performing a single control loop using the standalone controllers and the proposed architectures. The results provided were obtained by running each architecture 100 times. Notice that, for the hardware described in the methodology section, the lowest average loop

TABLE 7 Comparison of average loop time for the controllers inside the proposed architecture.

Architecture	RLQR	LQR	FL	PID
Standalone (s)	0.0037	0.0032	1.2608e-05	7.0953e-06
Architecture 1 (s)	0.0039	0.0033	9.0941e-04	1.3125e-05
Architecture 2 (s)	0.0043	0.0036	7.7599e-04	9.7755e-04

time was 70μs, and the highest was 4.3 ms. In this sense, for the scenario that requires more computational cost, the algorithm can still keep an update rate of ≈230 Hz for the trajectory tracking task.

8 | CONCLUSIONS

In this paper, we implemented two robust and intelligent architectures for position control of a quadrotor in the trajectory tracking task. The two proposed architectures are formed by an RLQR controller and DNNs that provide a reference signal to the controller (Architecture 1) and estimate wind disturbances affecting the quadrotor (Architecture 2). In addition, we used three other controllers within the proposed framework for comparison purposes: LQR, PID, and FL. This approach allows us to assess not only the individual performance of each controller but also the overall effectiveness of the architectures when implemented with different control strategies The results showed a performance improvement in the trajectory tracking task for all the implemented controllers combined with the DNNs. This combination provides a remarkable gain in flexibility and adaptability, allowing for real-time decision-making based on historical or live data. Moreover, By implementing a DOBC architecture, we effectively address two critical challenges: parametric uncertainties and external disturbances.

The inherent advantage of utilizing the RLQR as the core controller is its reliance on a single auxiliary parameter, requiring tuning. Nevertheless, the stability region of this controller remains constant, ensuring optimality across a broad spectrum of parametric uncertainties. This strategic integration not only enhances robustness but also upholds performance in dynamic environments, positioning it as a promising path for advancing quadrotor capabilities. Future work will validate the proposed control architectures through flights in outdoor environments. Also, due to the modular architecture developed, it will be possible to test the two proposed architectures combined.

NOMENCLATURE

ϕ	Pitch angle
θ	Roll angle
ψ	Yaw angle
q	State vector
Λ	Continuous state matrix
Γ	Continuous input matrix
R_t	Rotation matrix
δA	Continuous state uncertainty matrix
δB	Continuous input uncertainty matrix
d	External disturbance
B_d	Matrix mapping the external disturbance
m	Quadrotor mass
I_z	Moment of inertia with respect to the Z-axis
$\tilde{q}$	Trajectory tracking error
q^d	Desired trajectory vector
u	Virtual control input
I	Identity matrix
x	Augmented error state vector
y	Output of a dynamic system
W	Synaptic weight of the neural network
η	Learning rate
∇	Gradient operator
v_t	Momentum term
β_1	Momentum effect adjustment constant
β_2	Regulation term for RMSProp
s^{dw}	Second-order momentum term
v	First-order momentum term
ν	Control inputvector
F	Discrete state matrix
G	Discrete input matrix
δF	Discrete state uncertainty matrix
δG	Discrete input uncertainty matrix
K	RLQR/LQR controller gain
L	Closed-loop system matrix for RLQR/LQR
Q	State weighting matrix
R	Control input weighting matrix
P	Weighting matrix for RLQR/LQR
$\mathcal{J}$	Cost function for RLQR/LQR
μ	Penalty parameter
E_{F_i}	Known uncertainty matrix for F
E_{G_i}	Known uncertainty matrix for G
H	Auxiliary uncertainty matrix
Δ	Contraction matrix
K_p	Proportional gain
K_d	Derivative gain
K_i	Integral gain
s	Discretized augmented state vector
Φ	Discretized augmented state matrix
Ξ	Discretized augmented input matrix
Ω	Observation matrix
z	Measurement vector
d_a	Discretized disturbance
d_b	Derivative of discretized disturbance
u_{com}	Composite control law
u_{dist}	Disturbance compensation control law
C	System output matrix
K_{com}	Compensation gain
G_d	Discretized external disturbance mapping matrix
$\hat{d}$	Disturbance estimate
u_{dist}^{pd}	Proportional-derivative compensator
α	Proportional gain of the compensator
ω	Angular velocity
φ	Trajectory radius

AUTHOR CONTRIBUTIONS

Paulo V. G. Simplício: Data curation; investigation; methodology; software; validation; writing—original draft. **João R. S. Benevides**: Formal analysis; investigation; validation; writing—review and editing. **Roberto S. Inoue**: Conceptualization; formal analysis; supervision; writing—review and editing. **Marco H. Terra**: Formal analysis; funding acquisition; project administration; supervision; writing—review and editing.

CONFLICT OF INTEREST STATEMENT

The authors declare no conflicts of interest.

DATA AVAILABILITY STATEMENT

The data that support the findings of this study are available from the corresponding author upon reasonable request.

ORCID

Paulo V. G. Simplício https://orcid.org/0000-0002-9547-0624
João R. S. Benevides https://orcid.org/0000-0002-7175-0883
Roberto S. Inoue https://orcid.org/0000-0003-2813-9330
Marco H. Terra https://orcid.org/0000-0002-4477-1769

REFERENCES

1. Powers, C., Mellinger, D., Kumar, V.: Quadrotor Kinematics and Dynamics. In: Handbook of Unmanned Aerial Vehicles. Springer-Verlag GmbH, Berlin, Germany (2014)
2. Mahony, R., Kumar, V., Corke, P.: Multirotor aerial vehicles: Modeling, estimation, and control of quadrotor. IEEE Rob. Autom. Mag. 19(3), 20–32 (2012)
3. Liu, J., Xu, W., Guo, B., Zhou, G., Zhu, H.: Accurate mapping method for uav photogrammetry without ground control points in the map projection frame. IEEE Trans. Geosci. Remote Sens. 59(11), 1–9 (2021)
4. Adkins, K.A.: Urban flow and small unmanned aerial system operations in the built environment. Int. J. Aviation Aeronaut. Aerospace 6(1), 1–22 (2019)
5. Duggal, V., Sukhwani, M., Bipin, K., Reddy, G.S., Krishna, K.M.: Plantation monitoring and yield estimation using autonomous quadcopter for precision agriculture. In: IEEE International Conference on Robotics and Automation (ICRA), pp. 5121–5127. IEEE, Piscataway (2016)
6. Yang, H.C., AbouSleiman, R., Sababha, B., Gjoni, E., Korff, D., Rawashdeh, O.: Implementation of an autonomous surveillance quadrotor system. AIAA Infotech at Aerospace Conference and Exhibit and AIAA Unmanned, pp. 1–7. American Institute of Aeronautics & Astronautics, Reston, VA (2009)
7. Xiaoning, Z.: Analysis of military application of UAV swarm technology. In: 2020 3rd International Conference on Unmanned Systems (ICUS), pp. 1200–1204. IEEE, Piscataway (2020)
8. Abdelmaksoud, S.I., Mailah, M., Abdallah, A.M.: Control Strategies and Novel Techniques for Autonomous Rotorcraft Unmanned Aerial Vehicles: A Review. IEEE Access 8, 195142–195169 (2020)
9. Li, J., Li, Y.: Dynamic analysis and PID control for a quadrotor. In: IEEE International Conference on Mechatronics and Automation, pp. 573–578. IEEE, Piscataway (2011)
10. Cohen, M.R., Abdulrahim, K., Forbes, J.R.: Finite-horizon LQR control of quadrotors on $SE_2(3)$. IEEE Rob. Autom. Lett. 5(4), 5748–5755 (2020)
11. Yuan, X., Ren, X., Zhu, B., Zheng, Z., Zuo, Z.: Robust $\mathcal{H}_\infty$ control for hovering of a quadrotor uav with slung load. In: 2019 12th Asian Control Conference (ASCC), pp. 114–119. IEEE, Piscataway (2019)
12. Jebelli, A., Yagoub, M.C.E., Dhillon, B.S.: Feedback linearization approach to fault tolerance for a micro quadrotor. In: 2018 IEEE International Conference on Industrial Technology (ICIT), pp. 165–168. IEEE, Piscataway (2018)
13. Shang, W., Jing, G., Zhang, D., Chen, T., Liang, Q.: Adaptive fixed time nonsingular terminal sliding-mode control for quadrotor formation with obstacle and inter-quadrotor avoidance. IEEE Access 9, 60640–60657 (2021)
14. Almakhles, D.J.: Robust backstepping sliding mode control for a quadrotor trajectory tracking application. IEEE Access 8, 5515–5525 (2020)
15. Liu, J.: Intelligent Control Design and MATLAB Simulation. Tsinghua University Press, Beijing; Springer, Singapore (2017)
16. Kim, J., Gadsden, S.A., Wilkerson, S.A.: A Comprehensive Survey of Control Strategies for Autonomous Quadrotors. Can. J. Electr. Comput. Eng. 43(1), 3–16 (2020)
17. Lee, H., Kim, H.J.: Robust control of a quadrotor using takagi-sugeno fuzzy model and an lmi approach. In: 2014 14th International Conference on Control, Automation and Systems (ICCAS 2014), pp. 370–374. IEEE, Piscataway (2014)
18. Sarabakha, A., Kayacan, E.: Online Deep Learning for Improved Trajectory Tracking of Unmanned Aerial Vehicles Using Expert Knowledge. IEEE International Conference on Robotics and Automation (ICRA), pp. 7727–7733. IEEE, Piscataway (2019)
19. Kostadinov, D., Scaramuzza, D.: Online weight-adaptive nonlinear model predictive control. In: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 1180–1185. IEEE, Piscataway (2020)
20. Li, Q., Qian, J., Zhu, Z., Bao, X., Helwa, M.K., Schoellig, A.P.: Deep neural networks for improved, impromptu trajectory tracking of quadrotors. IEEE International Conference on Robotics and Automation (ICRA), pp. 5183–5189. IEEE, Piscataway (2017)
21. Li, S., Yang, J., Chen, W.H., Chen, X.: Disturbance Observer-Based Control: Methods and Applications, 1st ed. CRC Press, Boca Raton, FL (2014)
22. Pi, C.H., Ye, W.Y., Cheng, S.: Robust quadrotor control through reinforcement learning with disturbance compensation. Appl. Sci. 11(7), 3257 (2021)
23. Lazim, M.I., Husain, A., Basri, A., Mohd Subha, N.A.: Feedback linearization with intelligent disturbance observer for autonomous quadrotor with time-varying disturbance. Int. J. Mech. Mechatron. Eng. 18, 47–55 (2018)
24. Terra, M.H., Cerri, J.P., Ishihara, J.Y.: Optimal robust linear quadratic regulator for systems subject to uncertainties. IEEE Trans. Autom. Control 59(9), 2586–2591 (2014)
25. Santana, L.V., Brandao, A.S., Sarcinelli Filho, M., Carelli, R.: A trajectory tracking and 3D positioning controller for the AR.Drone quadrotor. In: International Conference on Unmanned Aircraft Systems, ICUAS 2014 - Conference Proceedings, pp. 756–767. IEEE, Piscataway (2014)
26. Benevides, J.R.S., Inoue, R.S., Paiva, M.A.D., Terra, M.H.: ROS-based robust and recursive optimal control of commercial quadrotors. IEEE International Conference on Automation Science and Engineering, pp. 998–1003. IEEE, Piscataway (2019)
27. Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. Adaptive Computation and Machine Learning Series. MIT Press, Cambridge, MA (2016)
28. Polyak, B.T.: Some methods of speeding up the convergence of iteration methods. USSR Comput. Math. Math. Phys. 4(5), 1–17 (1964)
29. Géron, A.: Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems. O'Reilly Media, Sebastopol, CA (2019)
30. Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.: Dropout: A simple way to prevent neural networks from overfitting. J. Mach. Learn. Res. 15(56), 1929–1958 (2014). http://jmlr.org/papers/v15/srivastava14a.html
31. Chen, W.H., Yang, J., Guo, L., Li, S.: Disturbance-observer-based control and related methods - An overview. IEEE Trans. Ind. Electron. 63(2), 1083–1095 (2016)
32. Gao, Z.: On the centrality of disturbance rejection in automatic control. ISA Trans. 53(4), 850–857 (2014)
33. Guo, L., Cao, S.: Anti-disturbance control theory for systems with multiple disturbances: A survey. ISA Trans. 53(4), 846–849 (2014)
34. Li, S., Yang, J.: Robust autopilot design for bank-to-turn missiles using disturbance observers. IEEE Trans. Aerosp. Electron. Syst. 49(1), 558–579 (2013)

35. Sun, J., Wang, C., Xin, R.: On Disturbance Rejection Control of Servo System Based on the Improved Disturbance. In: Chinese Control Conference, CCC, pp. 2554–2559. IEEE, Piscataway (2018)
36. Smith, J., Liu, C., Chen, W.H.: Disturbance observer based control for gust alleviation of a small fixed-wing UAS. In: 2016 International Conference on Unmanned Aircraft Systems, ICUAS 2016, pp. 97–106. IEEE, Piscataway (2016)
37. Xu, B., Wang, D., Zhang, Y., Shi, Z.: DOB-based neural control of flexible hypersonic flight vehicle considering wind effects. IEEE Trans. Ind. Electron. 64(11), 8676–8685 (2017)
38. Xu, B., Shou, Y., Luo, J., Pu, H., Shi, Z.: Neural learning control of strict-feedback systems using disturbance observer. IEEE Trans. Neural Networks Learn. Syst. 30(5), 1296–1307 (2019)
39. Huo, J., Meng, T., Jin, Z.: Adaptive attitude control using neural network observer disturbance compensation technique. In: Proceedings of 9th International Conference on Recent Advances in Space Technologies, pp. 697–701. IEEE, Piscataway (2019)
40. Ogata, K.: Modern Control Engineering. Instrumentation and Controls Series. Prentice Hall, New Jersey (2010)
41. Brockett, R.W.: Feedback Invariants for Nonlinear Systems. IFAC Proc. Vol. 11(1), 1115–1120 (1978)
42. Sastry, S.: Nonlinear Systems: Analysis, Stability, and Control. Interdisciplinary Applied Mathematics. Springer, New York (2013)
43. Kalman, R.E.: A new approach to linear filtering and prediction problems. J. Fluids Eng. Trans. ASME 82(1), 35–45 (1960)
44. Kim, D.W., Park, C.S.: Application of Kalman filter for estimating a process disturbance in a building space. Sustainability 9(10), 1868 (2017)
45. Benevides, J.R.S., Paiva, M.A.D., Simplício, P.V.G., Inoue, R.S., Terra, M.H.: Disturbance observer-based robust control of a quadrotor subject to parametric uncertainties and wind disturbance. IEEE Access 10, 7554–7565 (2022)
46. Quigley, M., Conley, K., Gerkey, B.P., Faust, J., Foote, T., Leibs, J., et al.: Ros: an open-source robot operating system. In: ICRA Workshop on Open Source Software. IEEE, Piscataway (2009)
47. Glorot, X., Bengio, Y.: Understanding the difficulty of training deep feedforward neural networks. In: Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, pp. 249–256. IEEE, Piscataway (2010)

How to cite this article: Simplício, P.V.G., Benevides, J.R.S., Inoue, R.S., Terra, M.H.: Robust and intelligent control of quadrotors subject to wind gusts. IET Control Theory Appl. 1–18 (2024). https://doi.org/10.1049/cth2.12615