Adaptive Outer-Loop Neural PID Control for Quadcopters Trained with EKF in a HIL Environment

J. Olin Estrada[0009−0006−2972−7539], Jorge D. Rios[0000−0001−7565−0874], and Alma Y. Alanis[0000−0001−9600−779X]

University Center for Exact and Engineering Sciences,
University of Guadalajara,
1421 Marcelino Garcia Barragan, Guadalajara, Jalisco 44430, Mexico
jose.estrada4333@alumnos.udg.mx, jorge.rarranaga@academicos.udg.mx,
alma.alanis@academicos.udg.mx

Abstract. Quadcopters require robust and adaptive control systems to ensure stability and trajectory tracking under dynamic environmental conditions. Traditional Proportional-Integral-Derivative (PID) controllers offer simplicity and effectiveness but lack adaptability to external disturbances such as wind gusts. This study proposes an Adaptive Neural PID controller trained using the Extended Kalman Filter (EKF) to dynamically adjust control gains in real time, enhancing trajectory tracking and disturbance rejection. The proposed approach is implemented as an outer-loop controller to regulate vehicle motion via velocity setpoints. The method is validated in a Hardware-in-the-Loop (HIL) simulation environment using a PX4-based quadcopter and Gazebo, assessing performance under nominal, moderate (6-8 m/s), and gusty wind conditions (8-14 m/s). Results demonstrate that Neural PID reduces settling time by up to 70.8% in Z-axis, improves mean squared error (MSE) by 34.2% in moderate wind, and decreases overshoot by 35.7% in lateral axes compared to conventional PID. Under gusty wind conditions, Neural PID provides enhanced stability in Y-axis, reducing deviations by 35.9%, though conventional PID maintains slightly lower error in X and Z. Overall, the findings highlight the effectiveness of adaptive neural control for improving quadcopter resilience against environmental disturbances, making it a viable alternative for real-world applications.
Keywords: UAV Control, Neural PID Control, Extended Kalman Filter, Hardware-in-the-Loop, Wind Disturbance Rejection.

1 Introduction

Unmanned Aerial Vehicles (UAVs) have experienced an accelerated growth in popularity due to advances in propulsion technologies and the development of lightweight, low-cost, and energy-efficient components. These systems have opened significant opportunities in military and civilian applications, including surveillance, fire detection, precision agriculture, traffic monitoring, and other activities with high social impact [1]. Among these platforms, quadcopters stand

https://doi.org/10.61728/AE20255251

out due to their Vertical Take-off and Landing (VTOL) capabilities, maneuverability, stable hovering, and adaptability to indoor and outdoor environments [2]. However, their complex, non-linear, and underactuated dynamics often demand sophisticated control methods to handle parameter uncertainties, external disturbances, and potential failures [3].

The design of a quadcopter control system presents significant challenges. Although Proportional-Integral-Derivative (PID) controller remains the most widely employed in UAVs due to its simplicity, ease of implementation, and effectiveness in structured environments [4], it relies on fixed parameter tuning, which limits its adaptability in real-world scenarios. When wind gusts or load variations occur, this approach may become inadequate, leading to oscillations, excessive overshoot, or instability [5]. These issues highlight the need for more adaptive and robust methods capable of adjusting in real time to changing flight conditions without requiring manual reconfiguration. In response, several advanced control strategies have been introduced, such as sliding mode control [6], backstepping control [7], and neural inverse optimal control [8] to improve the robustness and adaptability of UAVs. While these techniques have proven effective at directly tackling non-linearities of the system, they often require extensive modeling of low-level dynamics or significant tuning efforts.

A layered control strategy has become increasingly popular in practical applications, where an outer loop controller focuses on generating reference commands such as velocity or position setpoints, rather than directly regulating attitude or motor inputs [16]. Since u is not directly applied to actuators in this approach, it is referred to as velocity setpoints, as supported by previous research [17]. The resulting architecture enables modular and scalable control design by operating the outer loop at lower frequencies (e.g., 50 Hz), while an inner-loop manages attitude stabilization and motor actuation at higher frequencies (250 Hz–1 kHz), as illustrated in Fig. 1 [14].

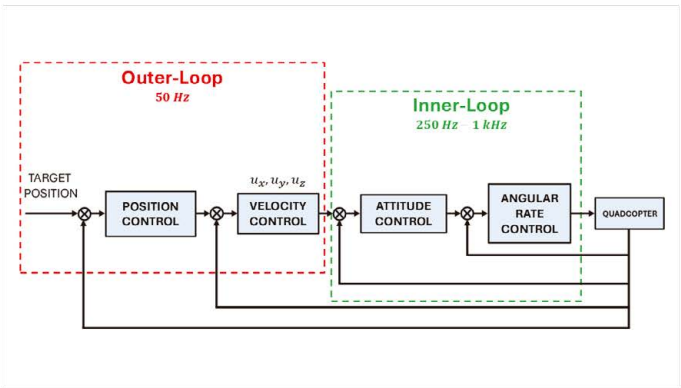

Fig. 1. Quadcopter Cascaded Control Architecture.

In this study, an outer loop position controller is implemented to regulate quadcopter motion via velocity setpoints. These setpoints are processed by a dedicated autopilot, such as Pixhawk[1], which executes inner-loop control for attitude stabilization and actuator signals. To evaluate the effectiveness of this approach, a conventional PID controller is compared against a Neural PID controller trained with Extended Kalman Filter (EKF) algorithm, aiming to improve trajectory tracking under parameter uncertainties and external disturbances. The proposed method is validated using Hardware-in-the-Loop (HIL) simulation integrating Gazebo[2] for realistic physics-based evaluation.

This paper is organized as follows: Section 2 describes methodology, including quadcopter kinematic model and proposed control approaches (conventional PID and Neural PID with EKF). Section 3 details experimental setup and implementation aspects. Section 4 presents results and performance metrics. Finally, Section 5 summarizes conclusions and discusses possible future research directions.

2 Methodology

2.1 Quadcopter Kinematic Model

The proposed quadcopter system is analyzed as a six-degree-of-freedom (DoF) system within the North-East-Down (NED) reference frame, which is commonly used in aerial robotics due to its alignment with navigation and mapping conventions [18], as shown in Fig. 2.

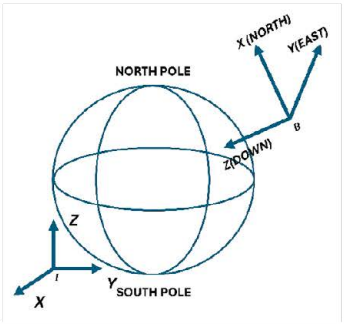

Fig. 2. North–East–Down Configuration.

Based on [19], the general motion of a quadcopter is defined in terms of the inertial reference frame $\varepsilon_I(x_I, y_I, z_I)$ and the body-fixed reference frame

[1] Pixhawk is a registered trademark of Dronecode Project, Inc.

[2] Gazebo is an open-source robotics simulator developed by the Open Source Robotics Foundation (OSRF).

$\varepsilon_B(x_B, y_B, z_B)$, with origin located at the center of mass. The orientation of the body-fixed reference frame ε_B on vehicle with respect to the inertial reference frame ε_I is determined by the rotation angles $\mu = [\phi, \theta, \psi]^\top$ using Euler angles or roll-pitch-yaw.

The body-fixed reference frame $\varepsilon_B(x_B, y_B, z_B)$ is rigidly attached to the quadcopter. The coordinates of the origin of $\varepsilon_B(x_B, y_B, z_B)$ with respect to $\varepsilon_I(x_I, y_I, z_I)$ are determined by Cartesian coordinate vector $\xi = [\xi_{xI}, \xi_{yI}, \xi_{zI}]^\top \in \varepsilon_I(x_I, y_I, z_I)$.

The mathematical structure that describes the general motion of orientation and position in three-dimensional space for a quadcopter, considering the analysis in moving reference frames using Euler angles, is given by:

$$p_I = \xi + R^c_{zyx}(\psi, \theta, \phi) p_B \tag{1}$$

where the term p_I represents the position of a point in the inertial frame, while p_B denotes the same point expressed in the body-fixed frame. The vector ξ defines the position of the body-fixed frame origin relative to the inertial frame. The rotation matrix $R_{ZYX}(\psi, \theta, \phi)$ transforms coordinates from the body-fixed frame to the inertial frame using a ZYX Euler angle convention, where ψ corresponds to yaw, θ to pitch, and ϕ to roll.

In this research, the outer-loop controller operates on desired velocity setpoints expressed in the NED frame. These commands are then processed by the onboard autopilot, which generates the necessary thrust and attitude angles to achieve the commanded motion.

2.2 Velocity Control Approaches

Conventional PID Control. Appropriate velocities are computed to achieve position tracking. Consequently, the controller determines velocities $u_x(t), u_y(t)$ and $u_z(t)$ at time step t to guide quadrotor from its current position (2) to desired position (3).

$$\begin{bmatrix} x(t) & y(t) & z(t) \end{bmatrix}^\top \tag{2}$$

$$\begin{bmatrix} x_d(t) & y_d(t) & z_d(t) \end{bmatrix}^\top \tag{3}$$

Error between desired and current positions is defined as

$$\begin{bmatrix} x(t) \\ y(t) \\ z(t) \end{bmatrix} = \begin{bmatrix} x_d(t) - x(t) \\ y_d(t) - y(t) \\ z_d(t) - z(t) \end{bmatrix} \tag{4}$$

Discrete PID control laws [11] at step t are provided in equations (5)–(7). In this control scheme, there is a dedicated PID module for each error signal $u_x(t)$, $u_y(t)$, and $u_z(t)$. The parameters K_P^x, K_I^x, and K_D^x correspond to proportional, integrative, and derivative gains for error x_e, respectively. Similarly, the parameters K_P^y, K_I^y, and K_D^y are applied to error y_e, while K_P^z, K_I^z, and K_D^z are used for error z_e.

$$u_x(t) = K_P^x x_e(t) + K_I^x \sum_{j=1}^{t} x_e(j) + K_D^x \left[x_e(t) - x_e(t-1)\right] \tag{5}$$

$$u_y(t) = K_P^y y_e(t) + K_I^y \sum_{j=1}^{t} y_e(j) + K_D^y \left[y_e(t) - y_e(t-1)\right] \tag{6}$$

$$u_z(t) = K_P^z z_e(t) + K_I^z \sum_{j=1}^{t} z_e(j) + K_D^z \left[z_e(t) - z_e(t-1)\right] \tag{7}$$

The gains are manually adjusted to ensure a stable response under nominal conditions. However, a main limitation of this method is the need for recalibration whenever the system experiences variations in its dynamics or flying conditions.

Adaptive Neural PID Control. The Adaptive Neural PID controller dynamically adjusts control gains in real time to enhance trajectory tracking and disturbance rejection. It replaces fixed PID parameters with a single-neuron architecture that learns and updates its proportional, integral, and derivative gains. Contrary to conventional PID control, which requires manual tuning, this adaptive approach improves system robustness by continuously refining control actions based on real-time error feedback.

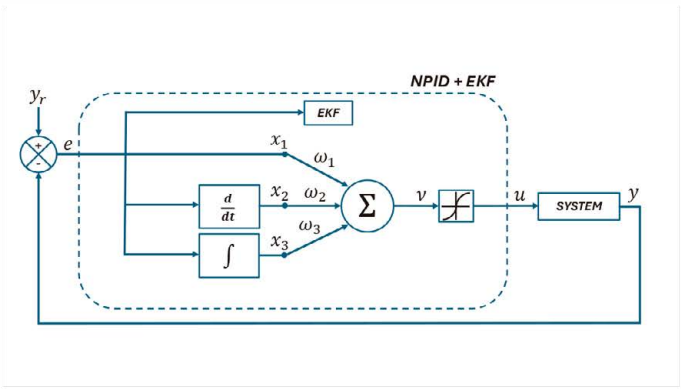

Fig. 3. Scheme of the Proposed Adaptive Neural PID Controller Trained With the EKF Algorithm.

$$\hat{y} = \Theta(\omega^\top x) \tag{8}$$

where ω is a weight vector, x input vector, $\hat{y}$ is output of neuron and Θ is an activation function. Training can be perform using different paradigms [10].

Neural PID Architecture. In Fig. 3 illustrates proposed Adaptive Neural PID controller. In this configuration, e denotes system error (9), defined as difference between reference y_r and output of the system y. The neuron receives x_1, x_2 and x_3 as inputs, representing proportional error (10), integral of error (12), and derivative of error (11), respectively. The weights ω_1, ω_2, and ω_3 are adapted online through EKF, and they correspond to K_P, K_I and K_D gains. The weighted sum v is defined in (13), and neuron output u is given in (14). The activation function $tanh(.)$ is employed, and α scales amplitude of $tanh$.

$$e(t) = y_r(t) - y(t) \tag{9}$$

$$x_1(t) = e(t) \tag{10}$$

$$x_2(t) = e(t) - e(t-1) \tag{11}$$

$$x_3(t) = \sum_{j=1}^{t} e(t) \tag{12}$$

$$v(t) = \sum_{i=1}^{3} \omega_i(t) x_i(t) \tag{13}$$

$$u(t) = \alpha \tanh(v(t)) \tag{14}$$

The activation function $tanh(.)$ operates within the interval $[-1, 1]$. In addition, parameter α must be chosen based on the level of control action necessary for the corresponding tasks.

EKF-Based Online Training. The EKF training algorithm is given by Equations (15)–(17)[9]:

$$\mathbf{K}(t) = \mathbf{P}(t)\mathbf{H}^{\top}(t)\left[\mathbf{R}(t) + \mathbf{H}^{\top}(t)\mathbf{P}(t)\mathbf{H}(t)\right]^{-1} \tag{15}$$

$$\omega(t+1) = \omega(t) + \eta \mathbf{K}(t)\left[y(t) - \hat{y}(t)\right] \tag{16}$$

$$\mathbf{P}(t+1) = \mathbf{P}(t) - \mathbf{K}(t)\mathbf{H}(t)\mathbf{P}(t) + \mathbf{Q}(t) \tag{17}$$

$$\mathbf{H}_j(t) = \left[\frac{\partial u(t)}{\partial \omega_j(t)}\right] = \left[\frac{\partial u(t)}{\partial v(t)}\frac{\partial u(t)}{\partial \omega_j(t)}\right] = \alpha\left[1 - tanh^2(v(t))x_j(t)\right] \tag{18}$$

where $\mathbf{K}$ denotes the Kalman gain, while $\mathbf{P}$, $\mathbf{Q}$, and $\mathbf{R}$ are the covariance matrices for weight estimation errors, estimation noise, and output noise, respectively. The matrix $\mathbf{H}$ encapsulates the partial derivatives of the neuron output with respect to each weight, η denotes the learning rate, and u is the neuron output. The error e corresponds to the difference between the desired output and the neuron output. Collectively, these elements form an iterative procedure that updates the neural network parameters, ensuring stable and efficient adaptation of the Single Neuron PID controller.

The training objective is to find a weight vector ω such that u minimizes the error between the reference signal and the system output, as shown in (9).

3 Experimental Setup

3.1 Hardware-in-the-Loop Configuration

The HIL configuration allows Pixhawk to process simulated sensor data and execute control commands in a virtual environment, replicating real-flight conditions. The system components include:

- Pixhawk 2.4.8 running PX4 firmware (v15.1.0).
- Host Computer: Intel Core i5 processor, 8GB RAM, Ubuntu 20.04.
- Simulator: Gazebo 11 with PX4 SITL.

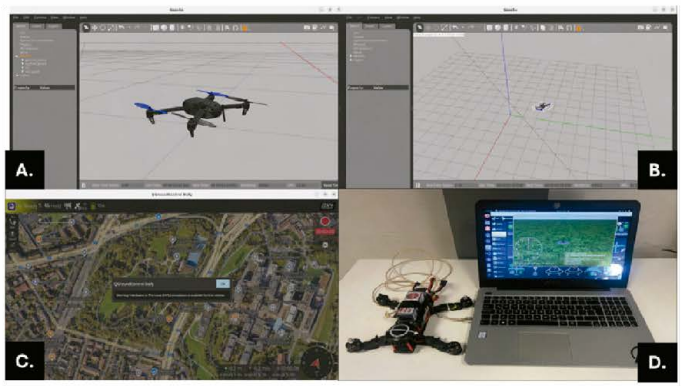

Fig. 4. Setup HIL.

3.2 Control System Implementation

To establish this configuration, Pixhawk is connected to host computer via USB, and QGroundControl[3] is used to activate HIL mode. Under this setup, physical sensors such as the Inertial Measurement Unit (IMU), Global Positioning System (GPS), and barometer are bypassed, and all measurements are generated by the simulation. These simulated sensor values are transmitted to autopilot via MAVLink[4] messages. The flight controller processes these commands, executes control signals, and sends actuator responses back to the simulator, creating a fully closed-loop system. [14].

Fig. 5 illustrates the data flow between the flight controller, QGroundControl, and simulation.

[3] QGroundControl is an open-source ground control station maintained by the Dronecode Project, Inc.

[4] MAVLink is an open-source communication protocol for unmanned vehicles maintained by the Dronecode Project, Inc.

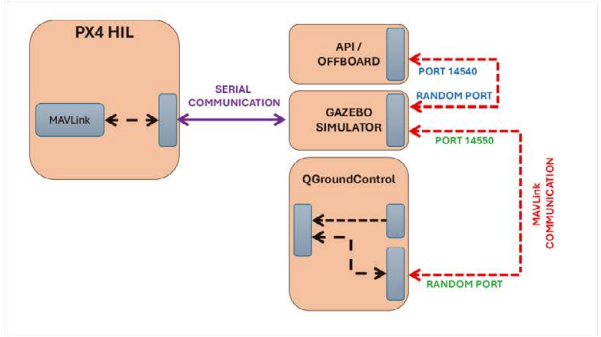

Fig. 5. Diagram of HIL Simulation Environment.

3.3 Control System Implementation

The outer-loop velocity controller is implemented in Python using the `pymavlink` library, which allows direct communication with the PX4 autopilot. The control system computes velocity setpoints based on position errors and transmits them to the Pixhawk via the MAVLink message `SET_POSITION_TARGET_LOCAL_NED`, configured with a mask that enables only velocity control in the NED frame.

To ensure real-time performance and reliability, the following optimizations are applied:

- Communication Baud Rate: MAVLink at 921600 bps.
- Controller Update Frequency: 50 Hz for velocity setpoints.
- Gazebo Physics Update Rate: 250 Hz to match real-time dynamics.

Additionally, MAVProxy is used to monitor and log MAVLink messages for debugging and analysis.

The closed-loop system operates by continuously receiving state feedback from Gazebo, which includes position, velocity, and attitude data. These measurements are used to correct velocity setpoints in real time, ensuring trajectory tracking even under external disturbances.

MAVLink Message Structure: `SET_POSITION_TARGET_LOCAL_NED`**.** This message specifies position, velocity, acceleration, and yaw setpoints in the local NED frame [15]. It is preferred over alternatives such as `SET_ATTITUDE_TARGET` because it allows direct velocity control while the autopilot autonomously manages attitude stabilization. The relevant dimensions are shown in Table. 1:

Type Mask Configuration. The `type_mask` is a 16-bit bitmap that determines which dimensions in the message are ignored by the autopilot. A value of 1 in a bit position indicates that the corresponding is ignored, while a value of 0 enables it. The specific mapping used in this implementation is shown in Table 2.

Table 1. Message Dimensions Used for the Implementation of Controllers.

Field	Type	Description
`target_system`	`uint8_t`	System ID
`target_component`	`uint8_t`	Component ID
`coordinate_frame`	`uint8_t`	`MAV_FRAME_LOCAL_NED`
`type_mask`	`uint16_t`	Bitmap
`vx`	`float`	X velocity setpoint (m/s)
`vy`	`float`	Y velocity setpoint (m/s)
`vz`	`float`	Z velocity setpoint (m/s)

Table 2. Bitwise Definition of Type Mask for Velocity Control. The Resulting Value for this Configuration is `0b110111000111`

Bit Position	Description	Value
0	Ignore position X	1
1	Ignore position Y	1
2	Ignore position Z	1
3	Ignore velocity X	0
4	Ignore velocity Y	0
5	Ignore velocity Z	0
6	Ignore acceleration X	1
7	Ignore acceleration Y	1
8	Ignore acceleration Z	1
9	Use force instead of acceleration	0
10	Ignore yaw	1
11	Ignore yaw rate	1

3.4 Simulation Environment

The Gazebo 11 is configured with Iris quadcopter from PX4. This model has the following properties:

- Mass: 1.5 kg.
- Inertia matrix: Defined based on standard Iris parameters.
- Motor configuration: Quadcopter X-frame with four brushless motors.
- Actuator response: Modeled based on real-world thrust-to-weight ratios.

Wind Simulation and Test Scenarios. To evaluate robustness of the control system under different environmental conditions, a wind disturbance model is implemented using the Gazebo Wind Plugin [14].

To establish realistic wind conditions, the Beaufort scale is used as a reference for selecting wind speeds, ensuring consistency with standardized meteorological classifications [20], [21]. Three scenarios are defined to assess the robustness of each controller under varying wind disturbances:

- Nominal Condition: No wind disturbances, evaluating reference tracking accuracy under ideal conditions ($0\ m/s$).

- Moderate Wind: A steady wind classified as Beaufort 4 (moderate breeze), with speeds ranging from 6 to 8 m/s. The disturbance is applied uniformly in all three axes (X, Y, Z), using an azimuthal angle of 0° with variations up to ±5°, simulating typical outdoor conditions.
- Gusty Wind: A continuous wind of 8 to 11 m/s (Beaufort 6) combined with intermittent gusts reaching 11 to 14 m/s, representing severe turbulence. The wind acts in all three axes (X, Y, Z) to evaluate disturbance rejection capabilities.

Each scenario is repeated multiple times, and performance metrics are statistically analyzed to ensure reproducibility and reliability of results.

3.5 Metrics

Performance of the controllers is evaluated using several quantitative metrics:

- Mean Squared Error (MSE): Measures the average squared deviation of system response from the desired trajectory.
- Steady-State Error (SSE): Quantifies the difference between desired and actual state once system has stabilized.
- Settling Time: The time required for the response to remain within a specified tolerance of the reference trajectory.
- Overshoot (%): The maximum deviation beyond the reference before stabilization.
- Integral Absolute Error (IAE): Evaluates accumulated absolute error over time, reflecting overall tracking accuracy.
- Standard Deviation (STD): Assesses response variability and fluctuations in velocity control.

Due to the transient nature of wind disturbances in the gusty wind scenario, conventional metrics such as settling time and steady-state error are not suitable. Instead, the evaluation is based on the system's ability to recover stability after disturbances. This is assessed through qualitative observation of trajectory tracking and real-time logging of MAVLink messages.

4 Results

4.1 Comparison of Controllers

Nominal Conditions (No Wind). Fig. 6 and 7 illustrate the position tracking and velocity control performance of PID and Neural PID controllers in a disturbance-free scenario.

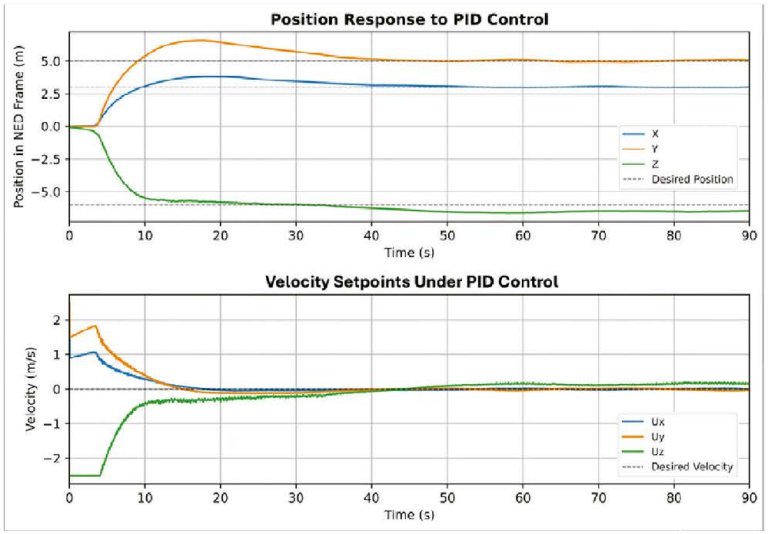

Fig. 6. Results Nominal Conditions PID.

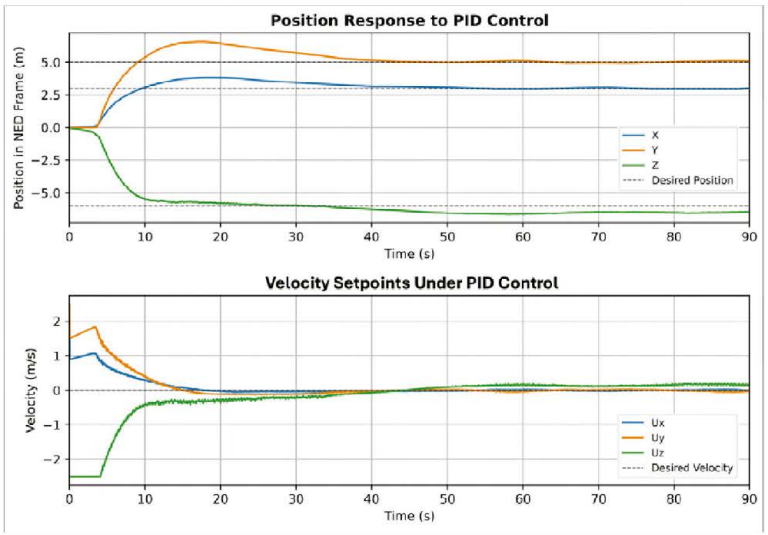

Fig. 7. Results Nominal Conditions Neural PID.

Neural PID demonstrates improved stability and reduced tracking error, particularly in the Z-axis, where settling time is significantly shorter.

Table 3. Performance Metrics Under Nominal Conditions.

Metric	X		Y		Z	
	PID	Neural PID	PID	Neural PID	PID	Neural PID
MSE	0.5789	**0.4174**	1.7062	**1.4859**	2.0594	**1.3221**
Settling Time (s)	9.33	**5.96**	8.97	**8.15**	25.70	**7.49**
Overshoot (%)	27.57	**11.65**	31.29	**25.04**	10.74	**9.12**
IAE	35.2736	**27.0838**	59.2087	**55.0637**	66.4549	**53.3511**
STD	0.7609	**0.6459**	1.3062	**1.2189**	1.4275	**1.1495**

Moderate Wind (Beaufort 4, 6–8 m/s). Fig. 8 and 9 depict the controllers responses under moderate wind conditions.

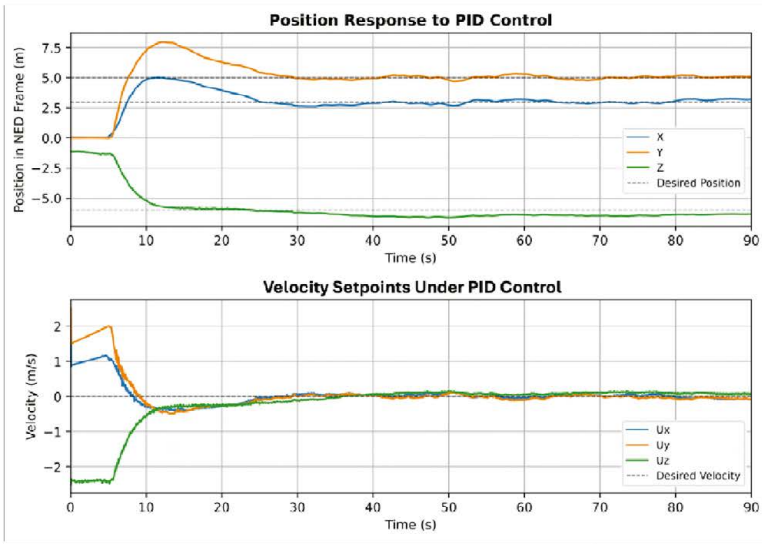

Fig. 8. Results Moderate Wind PID.

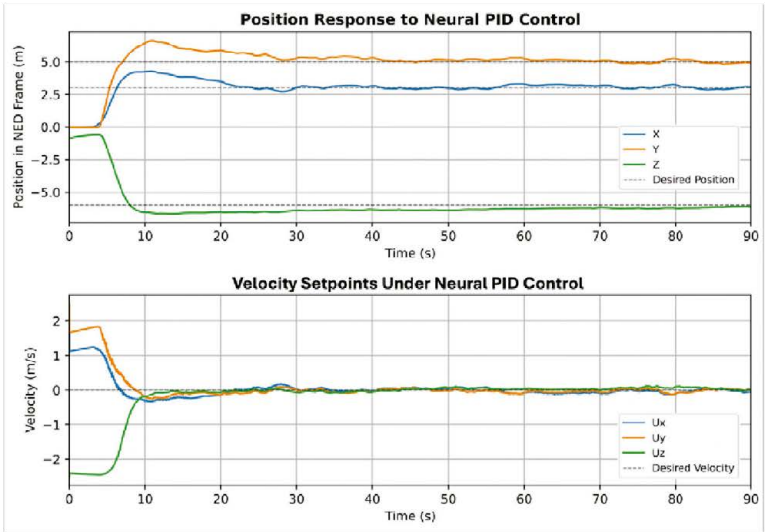

Fig. 9. Results Moderate Wind Neural PID.

Neural PID exhibits lower overshoot and faster stabilization, particularly in the X and Y axes, which are more affected by lateral wind disturbances.

Table 4. Performance Metrics Under Moderate Wind Conditions.

Metric	X		Y		Z	
	PID	**Neural PID**	**PID**	**Neural PID**	**PID**	**Neural PID**
MSE	0.9850	**0.6224**	2.4178	**1.5760**	1.8667	**1.8419**
Settling Time (s)	7.49	**6.24**	7.59	**7.00**	22.96	**8.00**
Overshoot (%)	67.14	**43.12**	59.73	**32.58**	**10.16**	10.74
IAE	50.8625	**36.0855**	70.7562	**54.5496**	65.2699	**60.1176**
STD	0.9925	**0.7888**	1.5548	**1.2554**	1.3582	**1.3565**

Gusty Wind (Beaufort 6, 8–14 m/s). Fig. 10 and 11 illustrate the controllers responses under strong and intermittent wind bursts.

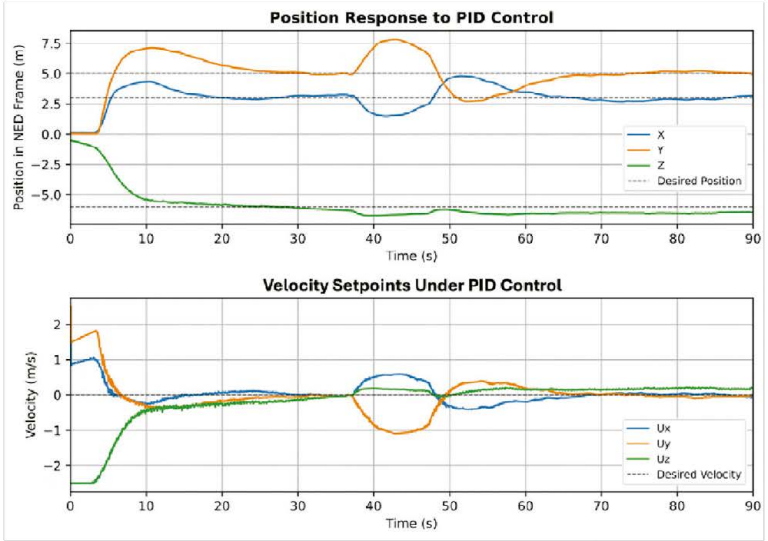

Fig. 10. Results Gusty Wind PID.

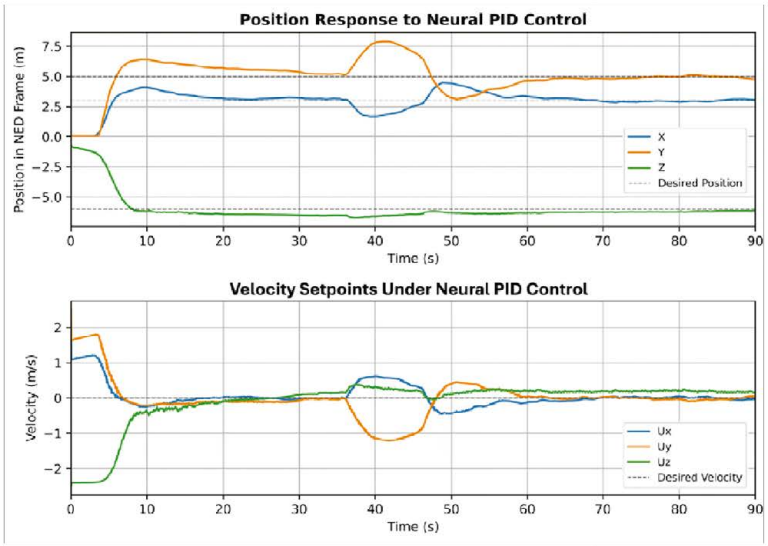

Fig. 11. Results Gusty Wind Neural PID.

The Neural PID controller demonstrates superior performance in the Y-axis, effectively reducing deviations caused by wind bursts. However, in the X and Z axes, the PID controller exhibits lower MSE and standard deviation STD, suggesting a more stable response in these directions. While the Neural PID reduces oscillations, it does not significantly enhance overall stability under extreme wind conditions.

Table 5. Performance Metrics Under Gusty Wind Conditions.

Metric	X		Y		Z	
	PID	**Neural PID**	**PID**	**Neural PID**	**PID**	**Neural PID**
MSE	**0.8541**	0.8993	4.0517	**2.5956**	1.4405	**1.7393**
STD	**0.9241**	0.9483	2.0129	**1.6110**	1.1963	**1.3157**

4.2 Discussion

Under nominal conditions, Neural PID exhibits superior performance in reducing settling time, tracking error, and overshoot. In moderate wind conditions, it provides faster convergence and lower oscillations, improving stability. In gusty wind scenarios, the Neural PID controller shows enhanced resilience, minimizing trajectory deviations and maintaining flight stability despite external disturbances.

5 Conclusions and Future Work

This study demonstrates that an Adaptive Neural PID controller trained with EKF improves quadcopter trajectory tracking and stability under varying wind conditions. In nominal and moderate wind scenarios, it outperforms conventional PID by reducing overshoot, improving settling time, and minimizing tracking error. Under gusty wind conditions, the Neural PID controller enhances stability in the Y-axis but does not significantly surpass the PID controller in other axes.

Future work will focus on extending this method to incorporate fault diagnosis modules (e.g., actuator or sensor failures) and exploring additional non-linear control strategies such as sliding-mode or backstepping with neural adaptation. Additionally, more extensive hardware flight tests beyond HIL will be conducted to validate these findings in real-world operational environments.

References

1. Mohsan, S.A.H., Othman, N.Q.H., Li, Y., et al.: Unmanned aerial vehicles (UAVs): practical aspects, applications, open challenges, security issues, and future trends. *Intelligent Service Robotics* **16**, 109–137 (2023).

2. Gomez-Avila, J., Villaseñor, C., Hernandez-Barragan, J., Arana-Daniel, N., Alanis, A., Lopez-Franco, C.: Neural PD Controller for an Unmanned Aerial Vehicle Trained with Extended Kalman Filter. *Algorithms*, **13**(2), 1–16 (2020).
3. Abbas, N., Abbas, Z., Zafar, D.S., Ahmed, N., Liu, X., Khan, S., Foster, E., Larkin, S.: Survey of Advanced Nonlinear Control Strategies for UAVs: Integration of Sensors and Hybrid Techniques. *Sensors* **24**, 13286 (2024).
4. Borase, R., Maghade, D., Sondkar, S., Pawar, S.: A review of PID control, tuning methods and applications. *International Journal of Dynamics and Control* **9**, (2021).
5. Nguyen, V.-C., Nguyen, M., Tran, H., Mien, T., La, H., Nguyen, H.: Trajectory Tracking Control for a Quadcopter under External Disturbances. *Engineering, Technology & Applied Science Research* **14**, 17620–17628 (2024).
6. Derafa, L., Benallegue, A., Fridman, L.: Super twisting control algorithm for the attitude tracking of a four rotors UAV. *Journal of the Franklin Institute*, **349**(2), 685–699 (2012).
7. Li, J., Wan, L., Li, J., Hou, K.: Adaptive Backstepping Control of Quadrotor UAVs with Output Constraints and Input Saturation. *Applied Sciences* **13**, 8710 (2023).
8. Antonio-Toledo, M., Sanchez, E. N., Alanis, A. Y.: Neural Inverse Optimal Control Applied to Quadrotor UAV. In: *2018 IEEE Latin American Conference on Computational Intelligence (LA-CCI)*, Guadalajara, México, pp. 1–8 (2018).
9. Sanchez, E., Alanis, A.: Redes Neuronales: Conceptos Fundamentales y Aplicaciones a Control Automático. Pearson Educación, Madrid, Spain (2006).
10. Haykin, S.: Kalman Filtering and Neural Networks. In: Adaptive and Cognitive Dynamic Systems: Signal Processing, Learning, Communications and Control. Wiley, Hoboken, NJ, USA (2004).
11. Hernandez-Barragan, J., Rios, J., Alanis, A., Lopez-Franco, C., Gomez-Avila, J., Arana-Daniel, N.: Adaptive Single Neuron Anti-Windup PID Controller Based on the Extended Kalman Filter Algorithm. *Electronics*, **9**(4), 636 (2020).
12. QGroundControl Documentation. `https://qgroundcontrol.com` last accessed: February 21, 2025.
13. MAVLink Protocol. `https://mavlink.io/en/` last accessed: February 21, 2025.
14. PX4 Autopilot Documentation. `https://docs.px4.io/main/en/`, last accessed: February 21, 2025.
15. MAVLink Protocol: Common Message Set. `https://mavlink.io/en/messages/common.html`, last accessed: February 21, 2025.
16. Rao, J., Li, B., Zhang, Z., Chen, D., Giernacki, W.: Position Control of Quadrotor UAV Based on Cascade Fuzzy Neural Network. *Energies* **15**(5), 1763 (2022).
17. Karimoddini, A., Lin, H., Chen, B., Lee, T.: Hierarchical hybrid modelling and control of an unmanned helicopter. *International Journal of Control* **87**, 10–1080 (2014).
18. Cai, G., Chen, B.M., Lee, T.H.: Coordinate Systems and Transformations. In: *Unmanned Rotorcraft Systems*, pp. 23–34. Springer, London (2011).
19. Reyes Cortés, F., Cid Monjaraz, J.: *Drones: Cinemática, Dinámica y Control de Cuadricópteros*. Alfaomega Grupo Editor, Ciudad de México (2019).
20. National Weather Service (NOAA): Official Website. `https://www.weather.gov`, last accessed: February 21, 2025.
21. Royal Meteorological Society: Beaufort Wind Scale. `https://www.rmets.org/metmatters/beaufort-wind-scale`, last accessed: February 21, 2025.