PID Tuning for a UAV using Reinforcement Learning

Grande Alessandro Francesco, Truppi Federico

September 30, 2025

Contents

1 Introduction

In the article [3], the tuning of PID gains was carried out using a reinforcement learning-based approach using the Deep Deterministic Policy Gradient (DDPG) algorithm. The goal of the system is to dynamically tune PID's controller gains for Roll, Pitch, and Yaw, improving quadrotor stability and response under different operating conditions.

This year's theme focuses on the analysis of adaptive tuning of a controller based on Reinforcement Learning (RL), applied not to the internal loop [3], but to the external loop controlling the position of an Unmanned Aerial Vehicle (UAV). The objective is to dynamically and online optimize the controller's gains to improve trajectory tracking performance in realistic operational scenarios.

We begin with the presentation of the mathematical model of the drone, in order to clearly define the state space and the structure of the controller, characterized by a two-loop architecture with negative feedback: an internal loop for attitude control and an external loop for position control.

Several simulations will then be presented in which the UAV is required to follow a reference helical trajectory, first in ideal conditions, then in the presence of sensory noise, and finally with the introduction of a wind disturbance model, simulated as the sum of Gaussian components translated over time.

The results obtained demonstrate that the controller parameters optimized via RL adapt dynamically during flight, leading to a significant reduction in position errors and an improvement in tracking performance compared to a controller manually tuned with classical methods.

Finally, a systematic comparison will be conducted between three advanced RL algorithms - SAC (Soft Actor-Critic), TD3 (Twin Delayed DDPG), and DDPG (Deep Deterministic Policy Gradient) - in order to identify which learning strategy is more effective, stable, and performant for UAV position control in disturbed environments.

2 Methodology: Quadcopter Mathematical Model

A UAV (Unmanned Aerial Vehicle) is an aircraft characterized by the absence of a human pilot on board. Their use has always been widespread in the military field, but is now also present in civilian applications such as aerial surveillance, remote sensing, and research. The main advantage of UAVs is their use in high-risk areas, i.e. places and territories where operating conditions are quite hostile or inaccessible.

According to the articles [2] and [1] the dynamic behavior of a quadcopter drone can be described by a set of non-linear differential equations that account for both translational dynamics (motion in space) and rotational dynamics (orientation). This type of system allows for greater flexibility and controllability, given by the arbitrary choice of control input based on the model chosen to describe the drone's dynamics.

2.1 Reference Systems

The first step to reach the mathematical model is to consider the quadcopter as a rigid body moving in the space. In this way, the importance of setting up reference systems capable of describing the dynamics of such a rigid body is evident. The first one is the Earth-fixed (IF) reference system, that is, a non-rotating inertial system with Z-axis pointing towards the south pole. The second one is the Body fixed (BF) reference system, that is, a reference system translating with the body itself and centered on the center of mass of the quadcopter. Thanks to them, it is possible to trace the translational coordinates and to know the dynamics of the attitude, respectively.

2.2 Angular Velocities

In order to understand the rotational dynamic of the BF relative to the IF, Euler angles (roll, pitch, yaw angles) were used. In practice, the BF reference system can be obtained from the IF thanks to three rotations: the first around the z axes of the IF of an angle ϕ; the second around the new y axes of an angle θ; the last occurs around the x axes of the BF of an angle ψ. Each of these rotations can be mathematically expressed by their rotational matrix:

$$R_{XBS}(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & \sin\phi \\ 0 & -\sin\phi & \cos\phi \end{bmatrix}$$

$$R_{y'}(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & \sin\theta \\ 0 & -\sin\theta & \cos\theta \end{bmatrix}$$

$$R_{ZIF}(\psi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\psi & \sin\psi \\ 0 & -\sin\psi & \cos\psi \end{bmatrix}$$

Applying the multiplication between matrices, we obtain the rotational matrix $R_{IF\to BF}$:

$$R_{IF\to BF} = \begin{bmatrix} \cos\theta\cos\psi & \cos\theta\sin\psi & -\sin\psi \\ \sin\theta\sin\phi\cos\psi - \cos\phi\sin\psi & \sin\theta\sin\phi\sin\psi + \cos\phi\cos\psi & \sin\phi\cos\theta \\ \cos\phi\sin\theta\cos\psi + \sin\phi\sin\psi & \cos\phi\sin\theta\sin\psi - \sin\phi\cos\psi & \cos\phi\cos\theta \end{bmatrix}$$

Since the inverse of this matrix corresponds to the transposed, the matrix from BF to IF is:

$$R_{BF\to IF} = \begin{bmatrix} \cos\theta\cos\psi & \sin\theta\sin\phi\cos\psi - \cos\phi\sin\psi & \cos\phi\sin\theta\cos\psi + \sin\phi\sin\psi \\ \cos\theta\sin\psi & \sin\theta\sin\phi\sin\psi + \cos\phi\cos\psi & \cos\phi\sin\theta\sin\psi - \sin\phi\cos\psi \\ -\sin\psi & \sin\phi\cos\theta & \cos\phi\cos\theta \end{bmatrix}$$

It is now possible to express the angular velocities of BF in the IF system:

$$\vec{\omega} = \begin{pmatrix} p \\ q \\ r \end{pmatrix}^{BF} = \begin{pmatrix} \dot{\phi} \\ 0 \\ 0 \end{pmatrix}^{BF} + R_{XBF}(\phi) \begin{pmatrix} 0 \\ \dot{\theta} \\ 0 \end{pmatrix}^{BF'} + R_{XBF}R_{y'}(\phi) \begin{pmatrix} 0 \\ 0 \\ \dot{\psi} \end{pmatrix}^{IF}$$

From this equation, it is possible to obtain the relationship between the angular velocities and roll,pitch,yaw angular variations:

$$\begin{pmatrix} p \\ q \\ r \end{pmatrix}^{BF} = \begin{pmatrix} 1 & 0 & -\sin\theta \\ 0 & \cos\phi & \sin\phi\cos\theta \\ 0 & -\sin\phi & \cos\phi\cos\theta \end{pmatrix}_{IF}^{BF} \begin{pmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{pmatrix}^{IF}$$

As last step, the following system is obtained:

$$\begin{cases} \dot{\phi} = p + (q\sin\phi + r\cos\phi)\tan\theta \\ \dot{\theta} = q\cos\phi - r\sin\phi \\ \dot{\psi} = (q\sin\phi + r\cos\phi)\frac{1}{\cos\theta} \end{cases}$$

2.3 Translational Velocities and Accelerations

Since BF is a rotational system, it is known from rational mechanics that the derivative of a vector (the acceleration in this case) can be expressed using the angular velocity defined in the previous section. So, if translational velocity is defined in the BF system as the following vector:

$$\vec{v}^{BF} = \begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{pmatrix}^{BF} = \begin{pmatrix} u \\ v \\ w \end{pmatrix}^{BF}$$

its derivative with respect to time is:

$$\vec{a}^{BF} = \dot{\vec{v}}^{BF} + \vec{\omega} \wedge \vec{v}^{BF}$$

In order to better track the position of the drone, it is useful to use the acceleration expressed in the IF, that is:

$$\vec{a}^{IF} = [\ddot{x}, \ddot{y}, \ddot{z}]^{IF}$$

2.4 Translational Dynamics

Now that we know how to express velocities and accelerations, we can analyze forces and torques acting externally on the quadcopter. Obviously, all of these must be expressed in the IF reference system. In terms of forces, there are three types:

1. Gravitational force, acting parallel to the Z-axis (remind that the IF has the Z-axis points towards the south pole, so it will agree with the sign of the acceleration g);
2. Aerodynamic forces: it is a friction force acting in the opposite direction to the motion of the drone;
3. Thrust: this force can be model as a linear combination of the quadratic velocities of the four rotors.

The three forces are in vector form below:

$$\vec{F}_g^{IF} = \begin{pmatrix} 0 \\ 0 \\ mg \end{pmatrix}^{IF}$$

where m is the total mass of the drone;

$$\vec{F}_{aero}^{IF} = \begin{pmatrix} A_x\dot{x} \\ A_y\dot{y} \\ A_z\dot{z} \end{pmatrix}^{IF}$$

A_i is the friction coefficient with respect to the axis i;

$$\vec{T}^{BF} = b\begin{pmatrix} 0 \\ 0 \\ \Omega_1^2+\Omega_2^2+\Omega_3^2+\Omega_4^2 \end{pmatrix}^{BF} = \begin{pmatrix} 0 \\ 0 \\ U_1 \end{pmatrix}^{BF}$$

b represents the rotor constant, Ω_i is the rotor velocity i.

As mentioned above, it is necessary to have the three forces in the IF reference inertial system. In order to write the thrust in the IF reference system, the rotational matrix $R_{BF\to IF}$ is used:

$$\vec{T}^{IF} = R_{BF\to IF}\begin{pmatrix} 0 \\ 0 \\ U_1 \end{pmatrix}^{BF} = \begin{pmatrix} (\cos\phi\sin\theta\cos\psi+\sin\phi\sin\psi)U_1 \\ (\cos\phi\sin\theta\sin\psi-\sin\phi\cos\psi)U_1 \\ (\cos\phi\cos\theta)U_1 \end{pmatrix}^{IF}$$

It is now possible to combine all the contributions of the forces in the different axes to obtain the translational dynamics:

$$\begin{pmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{z} \end{pmatrix}^{IF} = \begin{pmatrix} \frac{A_x}{m}\dot{x}+(\cos\phi\sin\theta\cos\psi+\sin\phi\sin\psi)U_1 \\ \frac{A_y}{m}\dot{y}+(\cos\phi\sin\theta\sin\psi-\sin\phi\cos\psi)U_1 \\ g+\frac{A_z}{m}\dot{z}+(\cos\phi\cos\theta)U_1 \end{pmatrix}^{IF}$$

2.5 Rotational Dynamics

In order to achieve the rotational dynamics, the first step is to define the torques of the quadcopter. There are three of them: roll, pitch and yaw.

- The rolling moment around the X-axis is generated by varying the speed of rotors 2 and 4;
- the pitching moment around the Y-axis is generated by varying the speed of rotors 1 and 3;

- the yawing moment around the Z-axis is obtained by appropriately combining the speeds of all the rotors.

In this way, we obtain the three torques:

$$M_x = U_2 = l * b(\Omega_4^2 - \Omega_2^2)$$

$$M_y = U_3 = l * b(\Omega_3^2 - \Omega_1^2)$$

$$M_z = U_4 = l * d(\Omega_4^2 + \Omega_2^2 - \Omega_1^2 - \Omega_3^2)$$

where l is the length of the drone arm, measured in the BF from its center of mass B.

Now, for a rigid body we can express the angular moment with respect to the center of mass B in the non-inertial system BF as follows:

$$\vec{K}_B^{BF} = \sum_{i=1}^{n} (P_i - B) \wedge m_i \vec{v_i}$$

From mechanics theory it is possible to write the angular moment in terms of the inertia tensor:

$$\vec{K}_B^{BF} = \sum_{i=1}^{n} (P_i - B) \wedge m_i [\vec{v_B} + \vec{\omega} \wedge (P_i - B)] =$$

$$= \sum_{i=1}^{n} (P_i - B) \wedge m_i \vec{v_B} + \sum_{i=1}^{n} (P_i - B) m_i \wedge [\vec{\omega} \wedge (P_i - B)] = \vec{I_B} \vec{\omega}$$

Since the BF is a principal inertial reference frame (or central inertia reference frame), that is, a reference frame in which the centrifugal moments are zero with respect to the reference axes, it is possible to simplify the tensor inertia matrix as a diagonal matrix:

$$\vec{I_B} = \begin{pmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{pmatrix}^{BF}$$

Deriving the angular moment vector, we obtain the angular moment of the external forces applied to the drone:

$$\vec{M}_B^{BF} = \frac{d}{dt} \sum_{j=1}^{3} K_j \hat{b}_j = \frac{d}{dt} \sum_{j=1}^{3} I_{jj} \omega_j \hat{b}_j =$$

$$= \frac{d}{dt} \sum_{j=1}^{3} I_{jj} \dot{\omega}_j \hat{b}_j + \vec{\omega} \wedge \sum_{j=1}^{3} I_{jj} \omega_j \hat{b}_j = \vec{I_B} \dot{\vec{\omega}} + \vec{\omega} \wedge \vec{I_B} \vec{\omega}$$

Recalling that $\vec{\omega} = \begin{pmatrix} p & q & r \end{pmatrix}'$, the last form of the derivative is:

$$\vec{M}_B^{BF} = \begin{pmatrix} U_2 \\ U_3 \\ U_4 \end{pmatrix}^{BF} = \begin{pmatrix} I_{xx}\dot{p} + qr(I_{zz} - I_{yy}) \\ I_{yy}\dot{q} + pr(I_{xx} - I_{zz}) \\ I_{zz}\dot{r} + pq(I_{yy} - I_{xx}) \end{pmatrix}^{BF}$$

2.6 Final Model

The complete model is given below:

$$\begin{cases} \ddot{x} = \frac{A_x}{m} \cdot \dot{x} + (\cos\phi \sin\theta \cos\psi + \sin\phi \sin\psi)\frac{U_1}{m} \\ \ddot{y} = \frac{A_y}{m} \cdot \dot{y} + (\cos\phi \sin\theta \sin\psi - \sin\phi \cos\psi)\frac{U_1}{m} \\ \ddot{z} = g + \frac{A_z}{m} \cdot \dot{z} + (\cos\phi \cos\theta)\frac{U_1}{m} \\ U_2 = I_{xx}\dot{p} + qr(I_{zz} - I_{yy}) \\ U_3 = I_{yy}\dot{q} + pr(I_{xx} - I_{zz}) \\ U_4 = I_{zz}\dot{r} + pq(I_{yy} - I_{xx}) \end{cases} \tag{1}$$

where:

- x, y, z: coordinates of the drone position in the inertial reference system;
- $\dot{x}, \dot{y}, \dot{z}$: linear velocities along the axes x, y, z;
- ϕ: Roll angle (rotation about the x axis of the body);
- θ: Pitch angle (rotation about the y axis of the body);
- ψ: Yaw angle (rotation about the z axis of the body);
- p, q, r: angular velocities around the x, y, z axes of the body;
- m: drone mass;
- I_{xx}, I_{yy}, I_{zz}: moments of inertia with respect to the x, y, z axes;
- A_x, A_y, A_z: coefficients of friction (linear) in the respective axes;
- U_1: total thrust force generated by the propellers;
- U_2: torque around the x axis (roll);
- U_3: torque around the y axis (pitching);
- U_4: torque around the z axis (yaw).

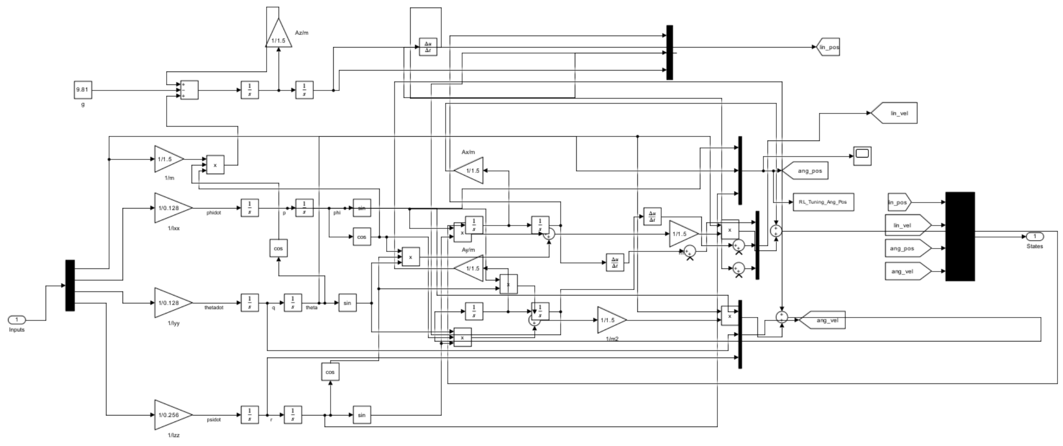

Figure 1: Block scheme of UAV

3 Drone Control Law

3.1 Proportional-Integral-Derivative Control

The PID controller is one of the most widespread and versatile control systems based on negative feedback. The controller acquires an input value from a process and compares it with a reference value. The difference is then used to determine the value of the controller's output variable, which is the process variable that can be manipulated. The PID regulates the output based on:

- the value of the error signal (proportional action);
- the past values of the error signal (integral action);
- how quickly the error signal changes (derivative action).

The three actions of a PID are calculated separately and simply added algebraically:

$$u(t) = u_p(t) + u_I(t) + u_D(t)$$

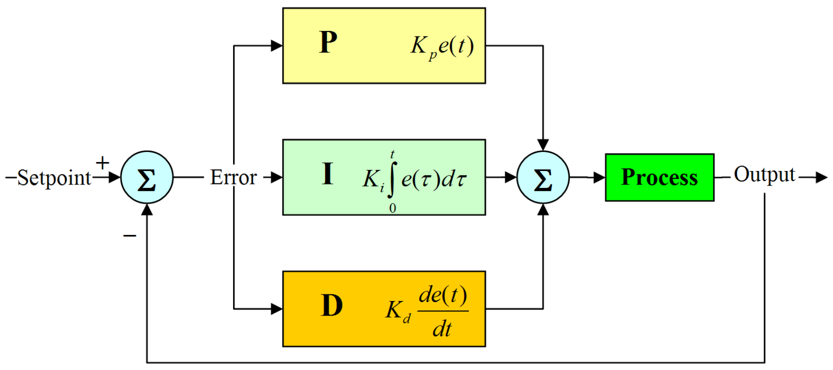

Figure 2: Block diagram of a PID

The proportional action is obtained by multiplying the error signal e with an appropriate constant:

$$u_p(t) = K_p e(t)$$

It represents a virtual spring force. It is possible to regulate a process with such a controller, which, in some simple cases, is even capable of stabilizing unstable processes.

The integral action is proportional to the time integral of the error signal e, multiplied by the constant K_I

$$u_I(t) = K_I \int_0^t e(\tau)d\tau$$

This definition of the integral action causes the controller to have memory of the past values of the error signal.

The derivative action is proportional to the time derivative of the error signal e, multiplied by the constant K_P.

$$u_D(t) = K_D \frac{de(t)}{dt}$$

It represents a virtual damping force. The idea is to quickly compensate for variations in the error signal.

Although implementing a PID controller at the programmatic level is conceptually simple, the optimal selection of the gains K_P, K_I, and K_D often represents one of the most delicate and complex phases of the entire control system design process. PID gains are not universal parameters: they depend heavily on the physical system to be controlled (inertia, friction, delays, nonlinearity) and the design specifications (settling time, overshoot, steady-state error, robustness to disturbances).

3.2 Proportional-Derivative Control of UAV

In the control of unmanned aerial vehicles (UAVs), it is essential to design control laws that can ensure the achievement and maintenance of a desired trajectory in 3D space. The proposed control is a cascaded control with a double Proportional-Derivative (PD) control loop:

- External loop (position control): receives the desired position and velocity as input and generates an acceleration reference.
- Inner loop (attitude control): takes the desired acceleration and current orientation as input and generates control moments for roll, pitch, and yaw.

This separation is justified by the hypothesis that the orientation dynamics is faster than the position dynamics, allowing a simplified synthesis of the two controllers.

Position and velocity errors are defined as:

$$e_p = p_{des} - p$$

$$e_v = v_{des} - v$$

where e_p e_v are 3x1 matrices having to control the position and velocity of the UAV in 3D space

The desired acceleration is calculated using a PD control law:

$$a_{des} = K_P e_p + K_D e_v \tag{2}$$

where K_P and K_D are diagonal matrices whose elements define the gains of the three PD controllers for the position.

The vertical component of the total force to be applied along the Z-axis of the UAV is given by the following:

$$F_Z = m(a_{des,z} + g)$$

where m is the mass of the drone and g is the gravitational acceleration. This force represents the overall thrust generated by the motors to counteract gravity and achieve the desired vertical acceleration.

To obtain the desired horizontal accelerations $a_{des,x}$ and $a_{des,y}$, the drone must tilt appropriately. Under the assumption of small angles and almost-steady-state dynamics, the roll (ϕ_{des}) and pitch (θ_{des}) references are obtained through kinematic inversion:

$$\phi_{des} = \frac{1}{g}(a_{des,x} sin\psi - a_{des,y} cos\psi)$$

$$\theta_{des} = \frac{1}{g}(a_{des,x} cos\psi + a_{des,y} sin\psi)$$

where ψ is the current yaw angle. The desired yaw is assumed constant and zero for simplicity, which implies that the drone does not rotate during flight.

We define:

- $\mu = [\phi, \theta, \psi]'$
- $\mu_{des} = [\phi_{des}, \theta_{des}, 0]'$
- $\omega = [p, q, r]'$

The attitude and angular velocity errors are the following:

$$e_\mu = \mu_{des} - \mu$$

$$e_\omega = -\omega$$

where e_μ e_ω are 3x1 matrices to control the orientation and angular velocity of the UAV in the 3D space. A zero reference is assumed for the desired angular velocity.

The motor torques are generated by the second PD controller:

$$\tau = K_P e_\mu + K_D e_\omega \tag{3}$$

where K_P and K_D are diagonal matrices whose elements define the gains of the three PD controllers for the orientation.

```
function [Fz, tau, e_ang] = PID_controller(t, x, des_pos, des_vel,
   gains)

    g = 9.81;  % gravitational acceleration [m/s^2]
    m = 1.15;  % UAV mass [kg]

    % Outer-loop position controller gains (diagonal matrices)
    Kp = diag([gains(1), gains(3), gains(5)]);   % position gains
       for [x, y, z]
    Kd = diag([gains(2), gains(4), gains(6)]);   % velocity gains
       for [x, y, z]

    % Inner-loop attitude controller gains (fixed, empirically
       tuned)
    Kp_ang = diag([150, 150, 100]);   % proportional gains for [phi
       , theta, psi]
    Kd_ang = diag([50, 50, 30]);      % derivative gains for [phi,
       theta, psi]

    % Extract states
    pos = x(1:3);       % [x, y, z]
    vel = x(4:6);       % [vx, vy, vz]
    phi = x(7);         % roll
    theta = x(8);       % pitch
    psi = x(9);         % yaw
    omega = x(10:12);   % [p, q, r] body angular rates

    % === Outer Loop: Position Control ===
    e_pos = des_pos - pos;          % position error
    e_vel = des_vel - vel;          % velocity error
    acc_des = Kp * e_pos + Kd * e_vel;  % desired acceleration from
        PD control

    % Total thrust: compensate gravity + track vertical
       acceleration
    Fz = m * (acc_des(3) + g);

    % === Inner Loop: Attitude Control ===
    % Map desired horizontal acceleration to desired roll & pitch (
       small angle approx.)
    phi_des   = (1/g) * (acc_des(1)*sin(psi) - acc_des(2)*cos(psi))
       ;
    theta_des = (1/g) * (acc_des(1)*cos(psi) + acc_des(2)*sin(psi))
       ;
    psi_des   = 0;  % desired yaw = 0 (simplified)

    % Current and desired attitude vectors
    ang_des = [phi_des; theta_des; psi_des];
    ang     = [phi; theta; psi];
    e_ang   = ang_des - ang;          % attitude angle error

    % Angular rate error: assume desired rate = 0
    e_omega = -omega;

    % Compute control torques
    tau = Kp_ang * e_ang + Kd_ang * e_omega;
end
```

Listing 1: PD controller for UAV position and attitude control.

4 Reinforcement Learning

Reinforcement learning (RL) is an interdisciplinary area of machine learning and optimal control that is concerned with how an intelligent agent should take actions in a dynamic environment to maximize a reward signal through trial and error. It is a type of learning where an agent interacts with its environment, taking actions and receiving feedback in the form of rewards or penalties, and uses this feedback to improve its decision-making process over time. Moreover, the actions taken may affect not only the immediate reward, but also the next state of the system and, as a consequence, the subsequent rewards. The agent creates a set of if-then rules or policies which help it to decide which action to take next to achieve the optimal cumulative reward. The agent must also choose between further exploring the environment to learn new state-action rewards or selecting known high-reward actions from a given state, the so-called exploration-exploitation trade-off. We can then define the principle elements present in reinforcement learning:

- Agent: the entity that learns and takes actions within the environment;
- Environment: the context in which the agent operates, providing feedback based on the agent's actions;
- Policy: the strategy or the mapping that dictates how an agent should act in a given state;
- Action: a decision made by the agent within the environment.
- Reward: feedback provided by the environment indicating the quality of the agent's action;
- Goal: the objective of the agent, typically to maximize the cumulative reward over time.

4.1 DDPG

The deep deterministic policy gradient (DDPG) algorithm is an off-policy actor-critic method for environments with a continuous action-space. It combines ideas from DPG (Deterministic Policy Gradient) and DQN (Deep Q-Network). In other words, DDPG is a technique consisting of two models (networks):

- Actor: the network that decides which action should be taken;
- Critic: the network that informs the actor how good was the action and how it should adjust, using a Q-function.

4.1.1 Why actor-critic?

The actor-critic is used to take advantage of both value-based and policy-based methods.

4.1.2 Why deterministic?

It is deterministic because the actor is a policy network that takes the state as input and outputs the exact action, instead of a probability distribution over actions such as DQN.

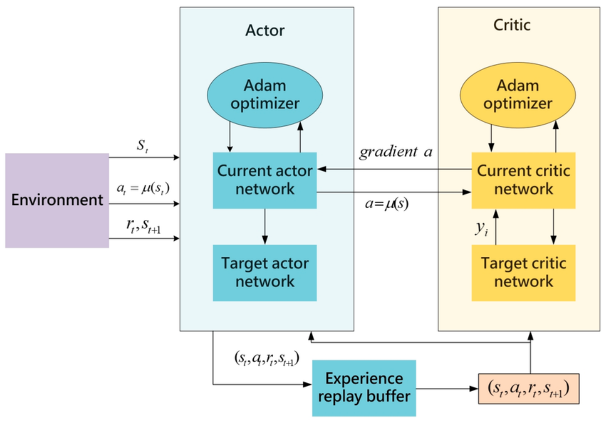

Figure 3: Diagram of the Reinforcement Learning algorithm based on DDPG

4.1.3 Why off-topic?

Off-policy learning is a method that allows an agent to learn a policy that is different from the policy used to generate the data. This allowed the method to leverage historical data and be more data-efficient. Also, since the data might not perfectly reflect the current policy, off-policy methods can be more prone to instability or bias.

4.2 TD3

TD3 stands for Twin Delayed Deep Deterministic Policy Gradient. It is an off-policy actor-critic method for environments with a continuous action-space. As the name suggests, it derives directly from the DDPG method and is presented as its evolution.

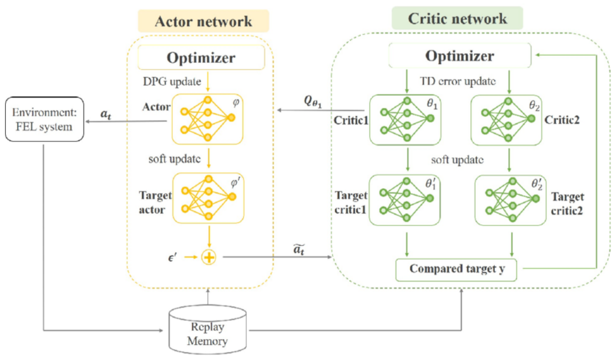

Figure 4: Structure of TD3

The main improvements are the following.

1. Twin Q networks (Twin Critics):
   - D3 uses two separate Q-networks (critics) instead of one;
   - The minimum of the two Q-values is used for the target:
   $$y = r + \gamma min(Q_1(s', a'), Q_2(s', a'))$$
   - This reduces the overestimation bias found in DDPG.
2. Delayed Policy Updates:
   - The actor (policy network) is updated less frequently than the critic (e.g., every 2 or more critic updates);
   - This prevents the policy from chasing noisy value estimates.
3. Target Policy Smoothing:
   - A small amount of noise is added to the target action when computing the target Q-value;
   - This prevents the policy from exploiting sharp peaks in the Q-function that may be artifacts of function approximation;
   - Clipped noise:
   $$a' = \pi(s') + clip(\epsilon, -c, c), \epsilon \sim N(0, \sigma)$$

This allows TD3 to be more stable than DDPG, have better performance in continuous control tasks, and avoid the overestimation of the Q-value function.

4.3 SAC

Soft Actor–Critic (SAC) is an off-policy actor–critic (2 Q-networks critic as TD3 agent) algorithm designed for a continuous action space. In this case, unlike standard reinforcement learning, the agent will not try to maximize just the expected reward, but also a policy entropy. This lets the actor output a probability distribution (usually a Gaussian distribution) over actions, and makes the agent prefer exploratory actions instead of deterministic greedy ones. The entropy weight is automatically adjusted by optimizing towards desired target entropy, improving adaptability.

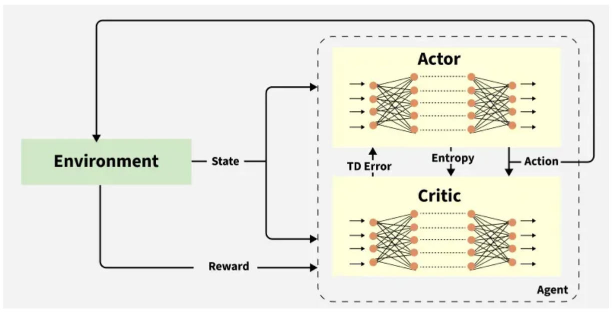

Figure 5: Structure of SAC

4.4 Methods Confrontation

Feature /Algorithm	DDPG	TD3	SAC
Policy Type	Deterministic	Deterministic	Stochastic
Learning Style	Off-policy	Off-policy	Off-policy
Critic Network	1 Q-network	2 Q-network (clipped min to reduce overestimation)	2 Q-network (clipped min for stability)
Actor Updates	Every step	Delayed	Every Step
Objective Function	Maximize Q-value	Maximize Q-value	Maximize Q-value, Entropy
Stability	Can diverge	Much more stable than DDPG	Very stable, robust across tasks
Best Suited For	Simple continuous control	Continuous control tasks needing stable training	Continuous control, tasks requires stochastic sampling
Weaknesses	Unstable, sensitive to hyperparameters	Still deterministic	More compute-intensive
Sample Efficiency	High	High	High

Table 1: Methods Comparison Table

5 Agent Environment Setup for Controller on Simulink

5.1 Implementation on Simulink

Taking into account the final model [1] and the control laws [2] and [3], the system was implemented in Simulink (Figure 6). The trajectory reference used is a helical curve (Figure 8). Before starting the RL training phase, the control was calibrated with the classic PD calibration method.

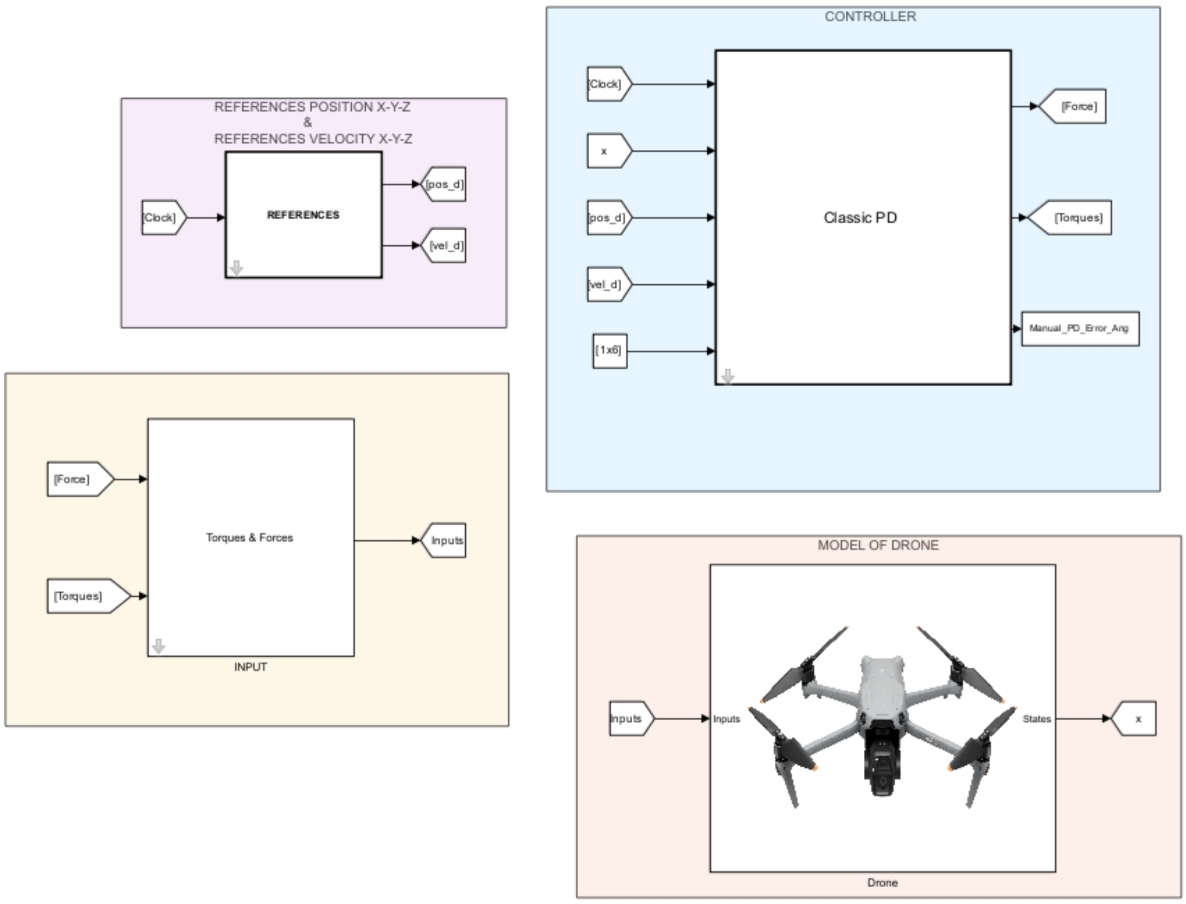

Figure 6: Simulink model of the controller and the UAV

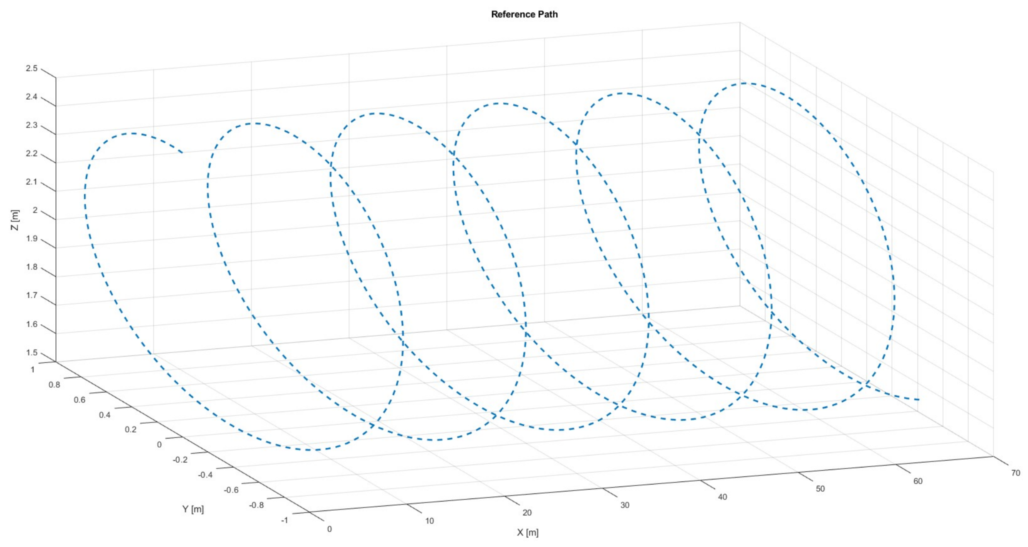

Figure 7: Path Reference

5.2 Reinforcement Learning on the Gains of a PD Controller

In MATLAB & Simulink, you can create the reinforcement learning environment from the Simulink model using the RL Agent block. The agent receives the drone's state space as observations and produces $\eta \in [-1, 1]$ as an action. Referring to article [1], starting from the initial tuning values, the control gains are computed adaptively. By heuristically assigning a value of $a = 0.5$ to (4), the following updating law is selected:

$$\begin{cases} K_P x^{\text{new}} = K_P x \left(1 + \alpha \cdot \eta_{K_P x}\right) \\ K_D x^{\text{new}} = K_D x \left(1 + \alpha \cdot \eta_{K_D x}\right) \\ K_P y^{\text{new}} = K_P y \left(1 + \alpha \cdot \eta_{K_P y}\right) \\ K_D y^{\text{new}} = K_D y \left(1 + \alpha \cdot \eta_{K_D y}\right) \\ K_P z^{\text{new}} = K_P z \left(1 + \alpha \cdot \eta_{K_P z}\right) \\ K_D z^{\text{new}} = K_D z \left(1 + \alpha \cdot \eta_{K_D z}\right) \end{cases} \tag{4}$$

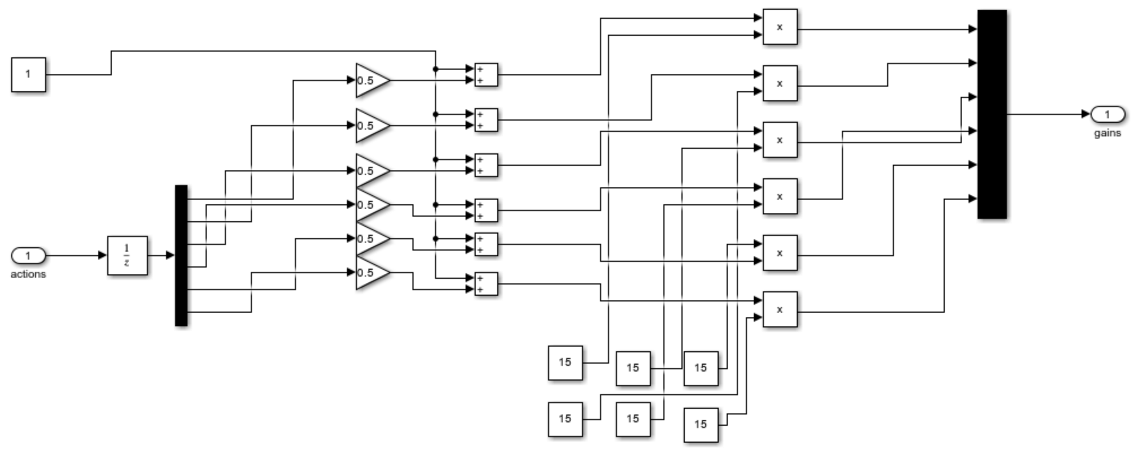

Figure 8: Block Scheme of New Gains

In the inner loop, the feedback for orientation control, the gains are left constant.

The reward function was defined as a weighted sum of the position, velocity, and attitude error norms.

$$\sigma = 0.60\|e_p\| + 0.20\|e_v\| + 0.20\|e_\mu\|$$

A higher weight has been assigned to the position error. However, for dynamic system control, it is also essential to limit velocity and attitude errors, since excessive deviations in these states can compromise stability.

The reward function is set as the following piece-wise function:

$$R\left(\sum_{i=0}^{t}\sigma_i\right) = \begin{cases} -25, & 11.5 \leq \sigma \\ -15, & 10.5 \leq \sigma < 11.5 \\ -10, & 9.5 \leq \sigma < 10.5 \\ -5, & 8.5 \leq \sigma < 9.5 \\ -1, & 7.5 \leq \sigma < 8.5 \\ 10, & 6.5 \leq \sigma < 7.5 \\ 15, & \sigma \leq 6.5 \end{cases} \tag{5}$$

where t is the simulation time.

5.3 Training phase

The training phase is carried out in Matlab/Simulink using the Deep Learning and Reinforcement Learning Toolbox, the simulation plant is modeled according to the figure 9, and the selected training hyperparameters are found heuristically and displayed below[3].

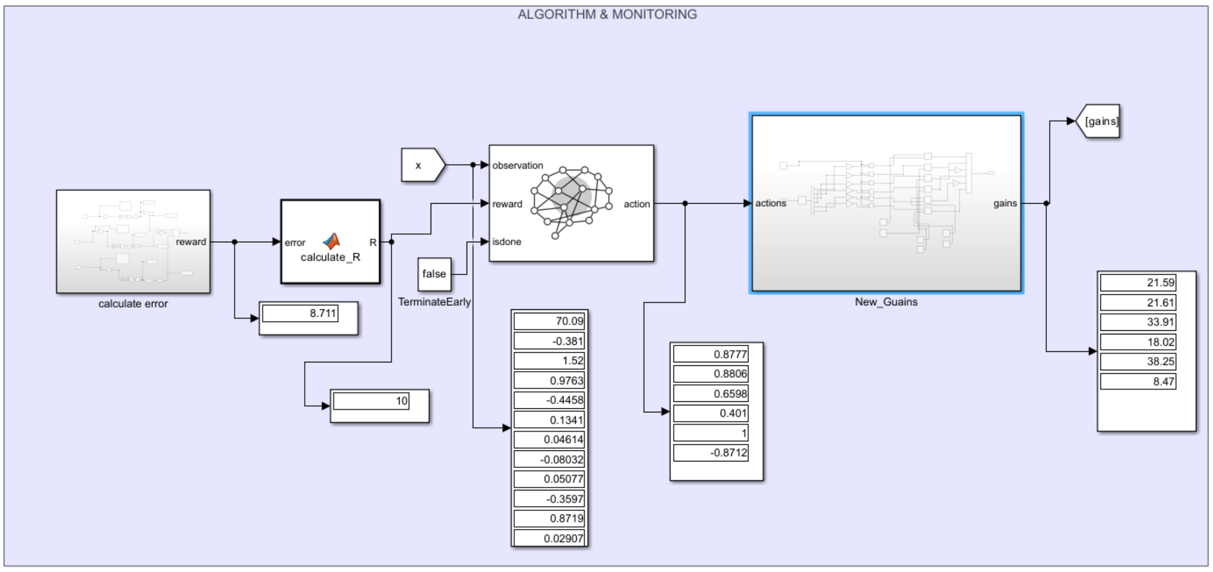

Figure 9: RL training algorithm & monitoring

Table 2: Hyperparameters

Parameters	Tuning
Sampling time	0.05
Reward discount factor γ	0.99
Learning rate for actor	10^{-3}
Learning rate for critic	10^{-3}
L_2 Regularization factor	10^{-5}
Optimizer parameter ϵ	10^{-8}
Minimum batch size	1024
Experience buffer length	10^{6}

6 Results

6.1 Simulations under ideal conditions

It is essential to carefully tune the controller parameters as they directly influence the system performance. In the case of a PD controller, too high a proportional gain can cause overshoot, while too high a derivative gain can lead to instability. Therefore, it is crucial to find an optimal balance between the two gains. With the initial tuning values, as can be seen in Figure 10, the trajectory teaching specifications, that is, $\lim_{t\to\infty} e_p = 0$ for each axis, are not fully satisfied. In fact, the steady-state position error is limited to a non-negligible range.

As for the velocity error with the classic PD, oscillations are obtained on the Y-axis without ever settling at a zero error value. A similar situation occurs on the Z-axis, albeit with minimal oscillations.(Figure 11).

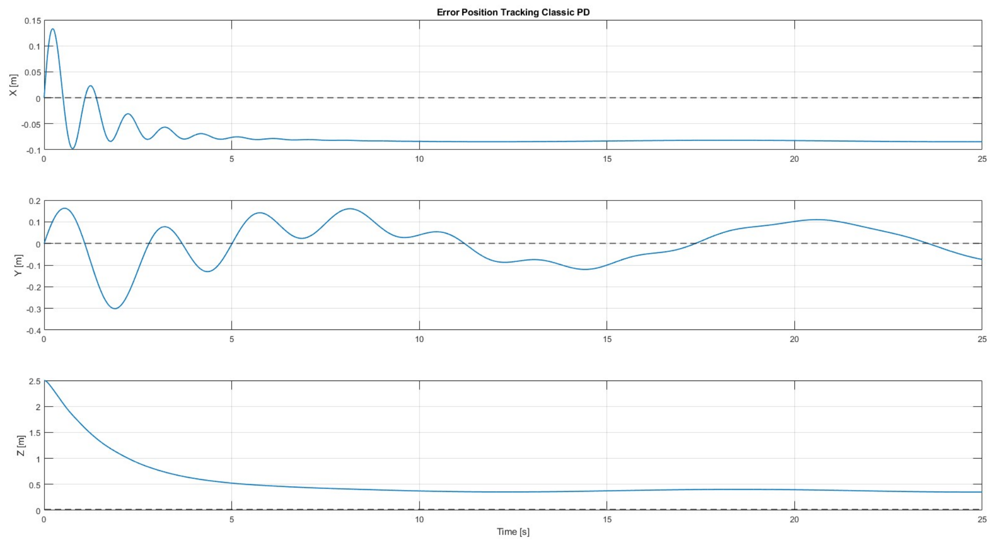

Figure 10: Error Position Tracking Classic PD

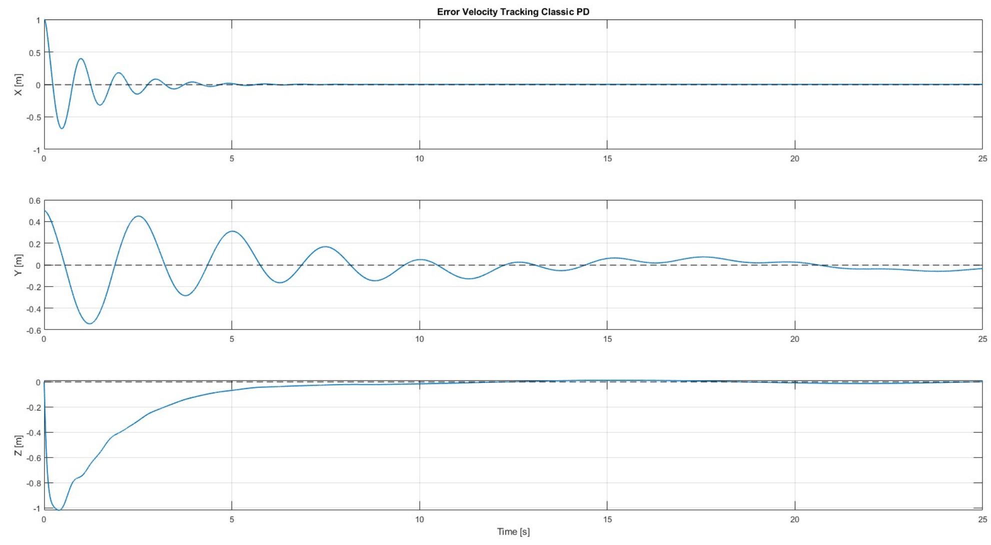

Figure 11: Error Velocity Tracking Classic PD

Once the training phase is completed, the RL agent-based controller is tested through several simulations. The results demonstrate that applying the RL agent-based method allows the system to reach steady-state conditions more quickly while reducing errors on all three axes. (Figure 12 13).

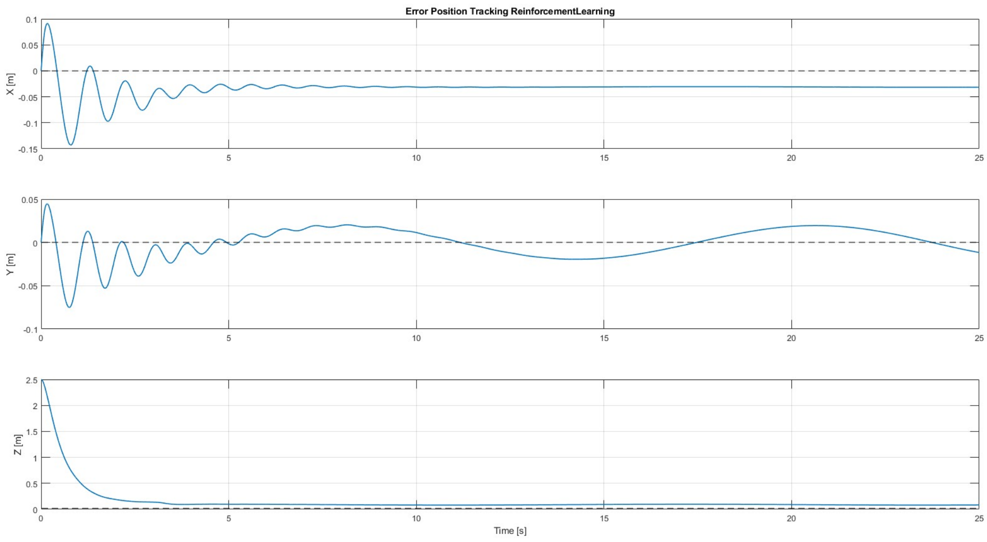

Figure 12: Error Position Tracking RL

By using Reinforcement Learning all errors of position in the three axes are considerably reduced, in particular the error on the x and z axes decreases the offset with respect to the reference, and the error on the Y-axis reduces the amplitude of its oscillations.

With Reinforcement Learning a lower value of the velocity error was obtained, minimizing the oscillations on the Y-axis and reducing the settling time on the Z-axis.

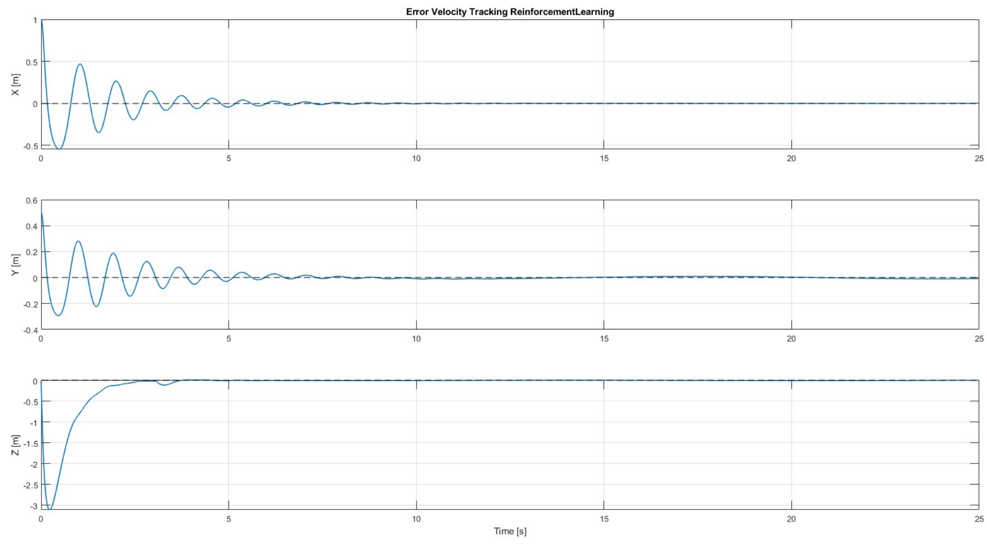

Figure 13: Error Velocity Tracking RL

Below are the two previous tracking errors compared.

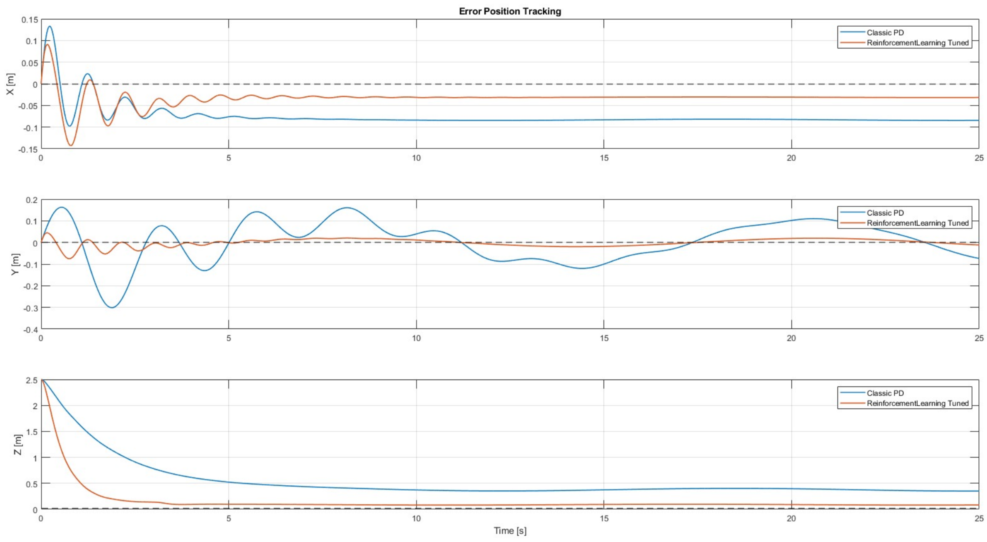

Figure 14: Error Position Tracking

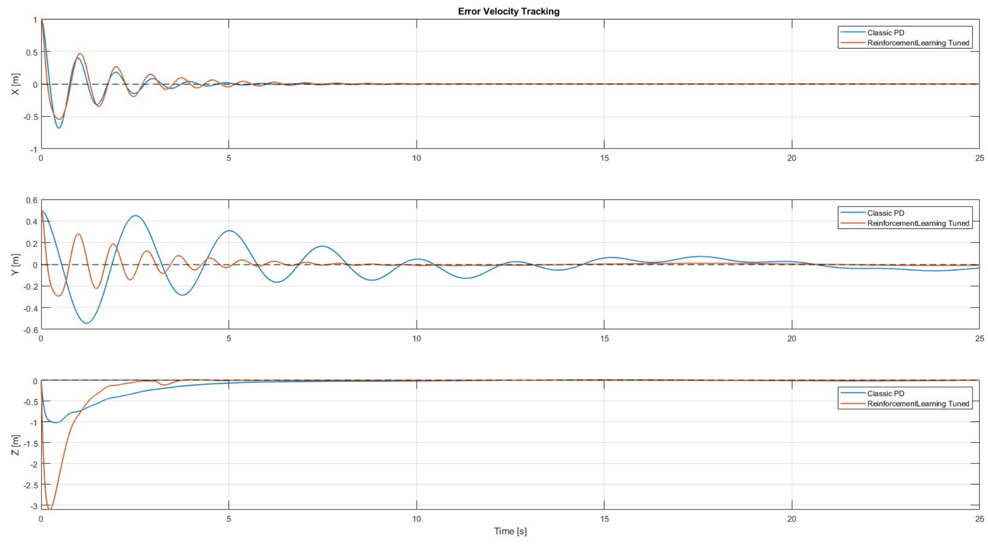

Figure 15: Error Velocity Tracking

6.2 Simulations with measurement error

Typically, sensors such as encoders, gyroscopes, cameras, etc., are used to measure the states of a UAV. The control loop does not use the actual motor states, but a measurement of these states, which is often subject to noise that is difficult to model. To model the measurement error, white noise was inserted into the Simulink feedback model.

By introducing measurement error in the two cases examined, we can observe the improved performance of reinforcement learning compared to classic PD. Indeed, we can observe the larger initial oscillations of classic PD on the Y-axis and the greater offset on the Z-axis compared to the reference.

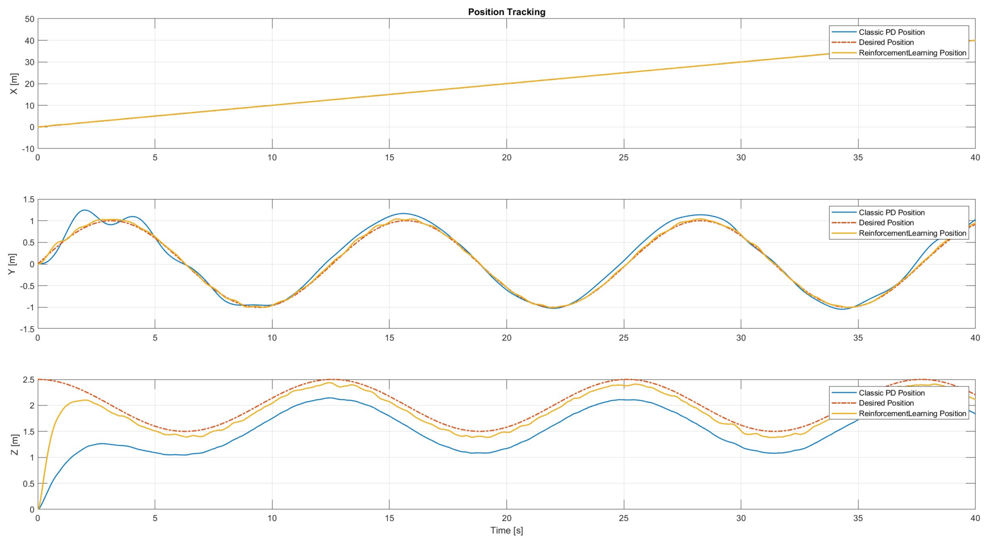

Figure 16: Position Tracking With Measurement Error

Below, the position errors with measurement error of classic PD and Reinforcement Learning are compared.

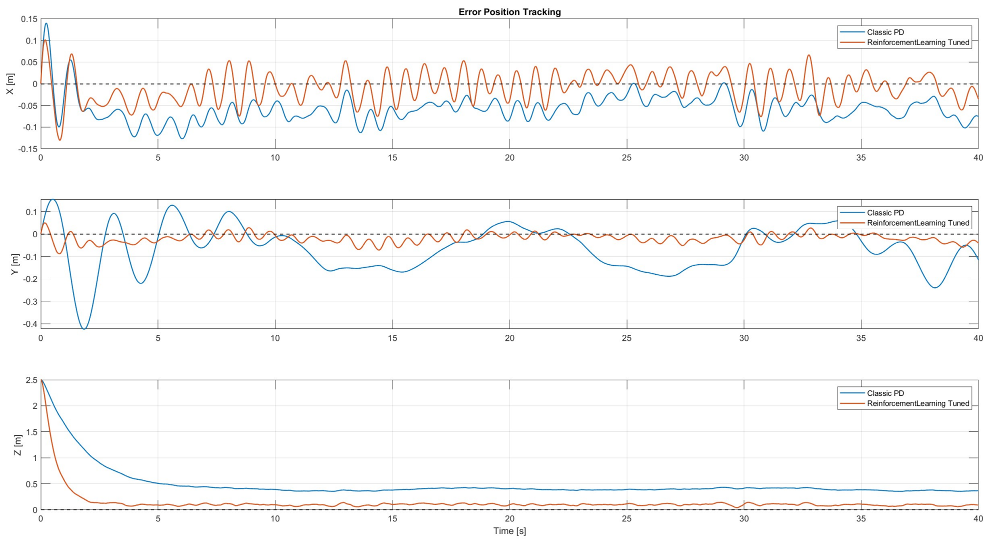

Figure 17: Error Position Tracking With Measurement Error

It is also interesting to evaluate how the PD controller gains change over time. The RL agent sets the controller gains differently from the initial values manually adjusted at the beginning of the simulation. Given the nature of a particularly difficult disturbance for the PD, the newly optimized gains remain constant and limited throughout the simulation. (Figure 18).

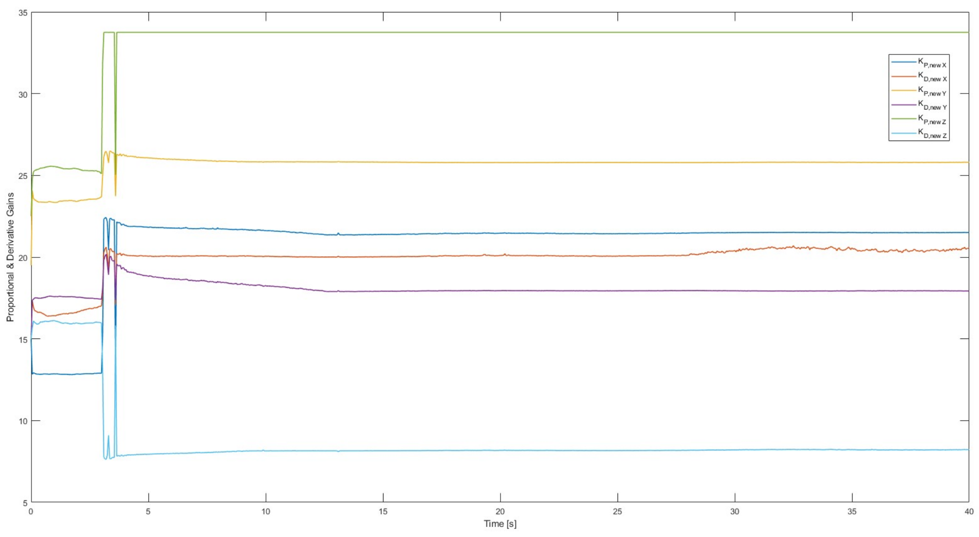

Figure 18: The outer loop controller gains of RL-based controller

To provide a clearer and more objective quantitative comparison between the performance of the manually tuned controller and the one optimized using RL, two standard metrics were adopted in addition to the norms: the *Root Mean Square Error* (RMSE) and the *Mean Absolute Error* (MAE). The results of this comparison are reported in Table 3, highlighting a clear improvement in trajectory tracking performance thanks to the RL-based approach.

Table 3: Comparison between Classic PD and RL-PD (with measurement error)

Method	**Err. X**	**Err. Y**	**Err. Z**	**RMSE**	**MAE**
Classic PD	8.02	14.58	71.19	0.347	0.215
RL PD	4.24	3.40	33.51	0.159	0.063

6.3 Wind

In order to provide a more realistic representation of the environment in which the UAV operates, an external disturbance, such as wind, was introduced into the system.

The wind profile was modeled by superimposing time-shifted Gaussian functions, each with a distinct variance, to reproduce a realistic turbulent behavior. The disturbance is applied to the X and Y components of the dynamical system.

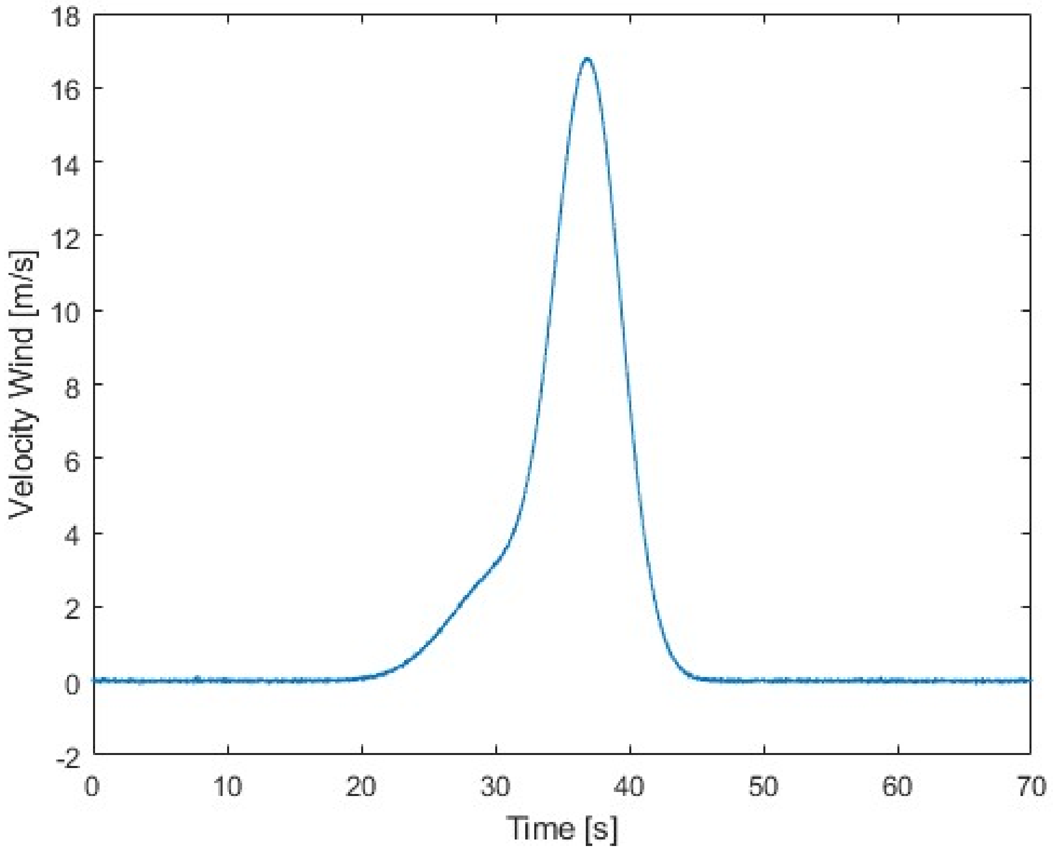

Figure 19: Example of wind representation

Figure 20 shows the position tracking behavior of a UAV subjected to atmospheric disturbances (wind), comparing two control strategies. The classic PD controller exhibits significant deviations where it accumulates lag with respect to the reference due to the cumulative effect of the wind. In contrast, the RL controller maintains more precise tracking, with limited oscillations and faster convergence to the desired trajectory.

1

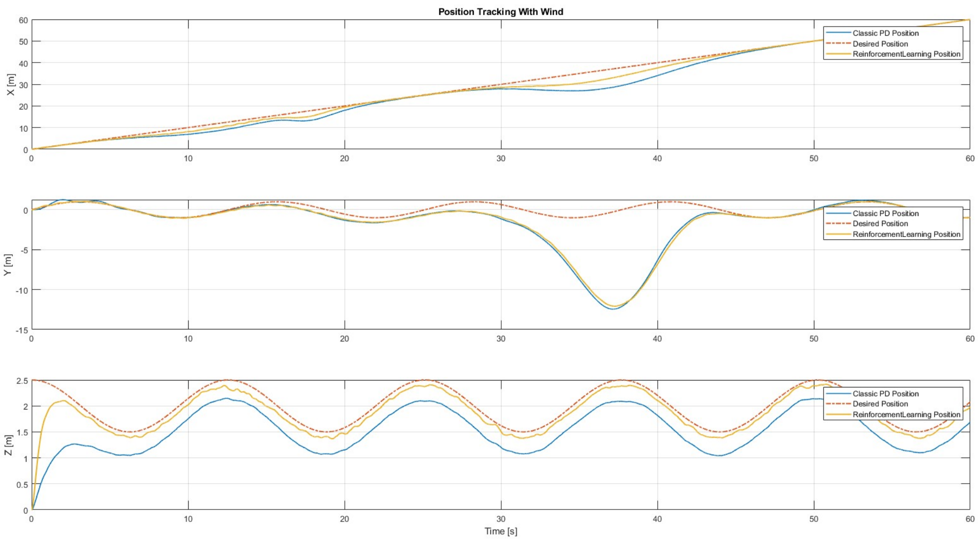

Figure 20: Position Tracking With Wind

To further highlight the differences between classic PD and Reinforcement Learning, the position errors in the presence of wind are compared below.

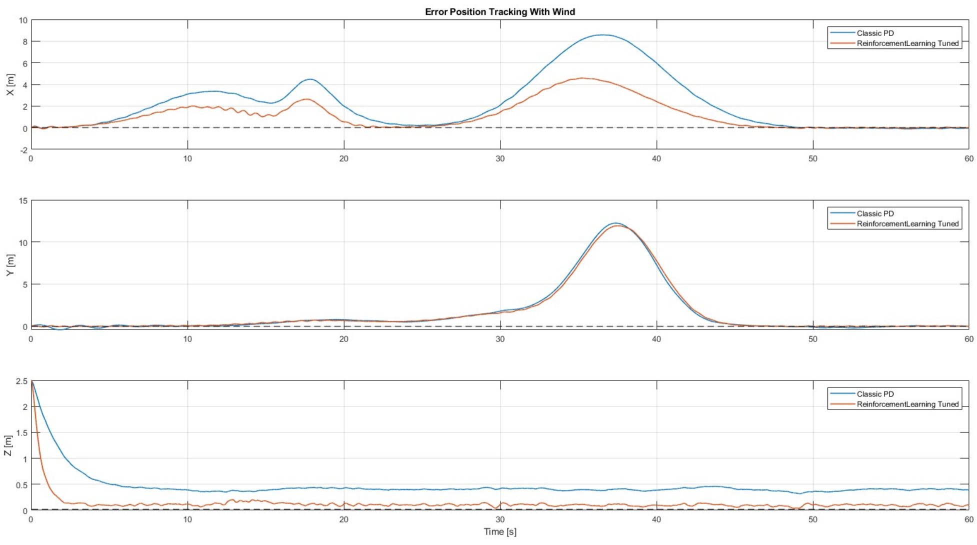

Figure 21: Error Position Tracking With Wind

As in the previous subsection, the different metrics were evaluated:

Table 4: Comparison between Classic PD and RL-PD (with measurement error and disturbance)

Method	**Err. X**	**Err. Y**	**Err. Z**	**RMSE**	**MAE**
Classic PD	492.8518	510.1828	80.2392	2.7433	1.4133
RL PD	256.5636	495.3493	35.3312	2.1679	0.9366

In conclusion, we can state that the results demonstrate that the application of adaptive tuning reduces error peaks, while allowing the system to reach stability conditions more quickly.

This improvement is attributed to the RL agent, which optimizes the controller by generating a refined set of gains compared to manually adjusted values.

The optimized gains result in better signal tracking and improved overall system response compared to manually adjusted gains, demonstrating the effectiveness of the RL-based approach.

1

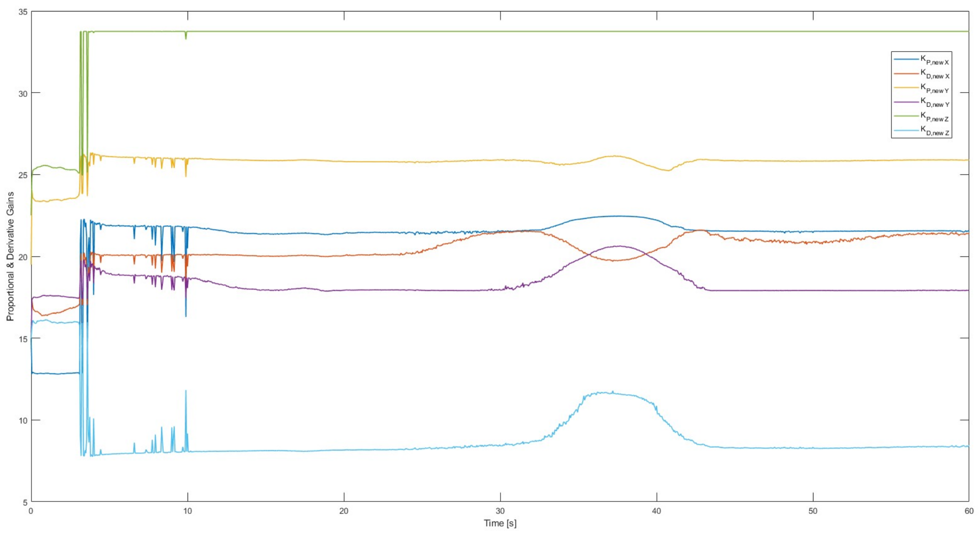

Figure 22: The outer loop controller gains of RL-based controller with disturbance of wind

6.4 Comparison with different algorithms

In the context of Reinforcement Learning (RL), the choice of algorithm is strongly influenced by the nature of the reward function. The reward structure in 5 introduces significant challenges for gradient-dependent algorithms.

The Deep Learning and Reinforcement Learning Toolbox provides several training algorithms. Performance was evaluated using three algorithms: DDPG (Deep Deterministic Policy Gradient), TD3 (Twin Delayed DDPG), and SAC (Soft Actor-Critic). To ensure a fair comparison, all algorithms were trained with the same set of hyperparameters, reported in Table 2. Training lasted 1000 episodes for each algorithm, without early stopping and without accounting for the noise added to the system [3], in order to evaluate final performance and convergence behavior under identical conditions. Furthermore, if the error diverges during an episode, the episode is terminated to accelerate convergence.

The Figure 23 shows that convergence during the training phase is faster in SAC and exhibits greater stability.

The structure of the reward function has a decisive impact on RL algorithm performance. In particular, the presence of discontinuities and plateaus favors algorithms that incorporate intrinsic exploration mechanisms (such as SAC) and penalizes those that rely solely on continuous gradients (such as DDPG).

1

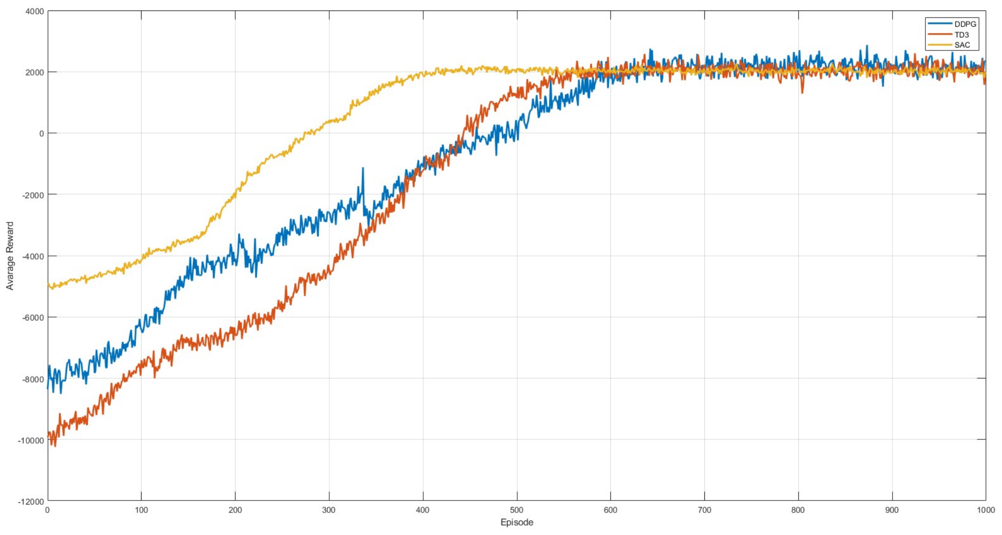

Figure 23: Comparison of the Training Phase of the DDPG, TD3, and SAC Algorithms

7 Conclusions

The results obtained highlight both the significant potential and the inherent challenges in applying reinforcement learning (RL) to fine-tune the parameters of PD controllers. In particular, the DDPG algorithm (an off-policy, actor-critic method) proved effective in optimizing the controller's outer-loop gains, enabling a robust and adaptive solution capable of accurately tracking complex trajectories even in the presence of nonlinear dynamics.

It is important to emphasize that during the training phase, no external disturbances or measurement noise were introduced, in order to isolate and evaluate the RL agent's intrinsic ability to learn an optimal control policy under ideal conditions. These perturbations (wind gusts along the Y-axis and sensory noise) were instead reintroduced exclusively during the testing phase, allowing for a controlled and targeted evaluation of the RL controller's robustness and adaptability in realistic scenarios.

Despite the absence of disturbances during training, the RL-optimized controller consistently outperformed the manually tuned controller in simulation tests, demonstrating increased trajectory tracking accuracy, superior adaptability to dynamic changes, and significantly improved robustness to environmental disturbances.

Including disturbances such as wind gusts and sensory noise directly in the training phase could further increase the agent's adaptability to real-world conditions. This approach would create a more realistic simulation environment, allowing the RL agent to develop strategies that take these factors into account from the outset. This improved training process could lead to an even greater reduction in errors and improved performance in outdoor scenarios.

Extending this control approach to the inner loop, using a multi-agent architecture, could offer important insights into the scalability and versatility of Reinforcement Learning-based strategies. A multi-agent system would allow for coordinated and distributed optimization of both the outer and inner loop parameters, paving the way for complete and hierarchical adaptive controllers. This configuration would not only allow for evaluating the interaction between different levels of learned control, but would also provide valuable insights into the ability of Reinforcement Learning to manage complex systems.

References

[1] Gabriel Hoffmann, Haomiao Huang, Steven Waslander, and Claire Tomlin. Quadrotor helicopter flight dynamics and control: Theory and experiment. 08 2007.

[2] Teppo Luukkonen. Modelling and control of quadcopter. *Independent research project in applied mathematics, Espoo*, 22(22):1–24, 2011.

[3] Serhat Sönmez, Luca Montecchio, Simone Martini, Matthew J. Rutherford, Alessandro Rizzo, Margareta Stefanovic, and Kimon P. Valavanis. Reinforcement learning based prediction of pid controller gains for quadrotor uavs, 2025.