S.T. Yau High School Science Award

Research Report

The Team

Name of team member:Zirui Wang
School:St. Princeton International School of Mathematics and Science
City, Country:Princeton,USA

Name of supervising teacher:Philip Tang
Job Title:Science teacher
School/Institution:Princeton International School of Mathematics and Science
City, Country:Princeton,USA

Title of Research Report

[Optimization Application of PPO Algorithm in Reinforcement Learning in Drone Attitude Balance]

Date

[23/8/2025]

Optimization Application of PPO Algorithm in Reinforcement Learning in Drone Attitude Balance

Abstract

Robust attitude and position control remain critical challenges for consumer drones. This studies evaluates the performance of a Reinforcement Leaning algorithm (PPO) against traditional control algorithms on drone's stability in both computer simulation and real life situations. Reinforcement Learning is trained with common parameters and rewards for small attitude error, lower angular rates, and efficient control effort. Performance were measured across level 0 to 5 wind in simulation and level 0 to 3 in real life experimentation. Results showed that PPO out-performed traditional PID controller in both computer simulation and real life experimentation. PPO showed better stability than PID with reasonable actuation. These findings indicate that PPO can produce more robust, precise control than fixed-gain controllers.

keywords: PPO; Reinforcement Learning; Drone; Attitude Control; Stability

Acknowledgements

I sincerely appreciate Mr Philip Tang and my school Princeton International School of Mathematics and Science for the guidance and support throughout the project.

I would like to also thank Mr Gang Hao for additional guidance on application of Reinforcement Learning.

After all, I would like to also thank my friends for their support over the difficult times during this research.

Declaration of Academic Integrity

The participating team declares that the paper submitted is comprised of original research and results obtained under the guidance of the instructor. To the team's best knowledge, the paper does not contain research results, published or not, from a person who is not a team member, except for the content listed in the references and the acknowledgment. If there is any misinformation, we are willing to take all the related responsibilities.

Names of team members Zirui Wang

Signatures of team members Zirui Wang

Name of the instructor Philip Tang

Signature of the instructor P Tang

Date 8/24/2025

Contents

1 - Introduction

1.1 - Background and Significances

1.1.1 - Development and Application fields of Drone Technology

Drones have rapidly evolved from niche gadgets to mainstream tools in the last decade as the manufacturing costs have decreased. Alongside the lower costs, many drone-related technologies have also improved, such as sensors. To adapt to the many different environments in which drones are being used today, flight controllers must be well-tested and exhibit desirable control even under extreme conditions.

Figure 1: DJI M30 in Extreme Conditions [1]

Drones typically consist of an inner loop for stability and control, and an outer loop for high-level tasks, such as navigation. Many algorithms are used to achieve these abilities, such as computer vision-based obstacle avoidance, and AI models for path planning and controls. The inner loop is implemented predominantly with PID. Despite the great performance of PID in stable environments, a different algorithm is definitely needed to navigate a harsher and more unpredictable environment.

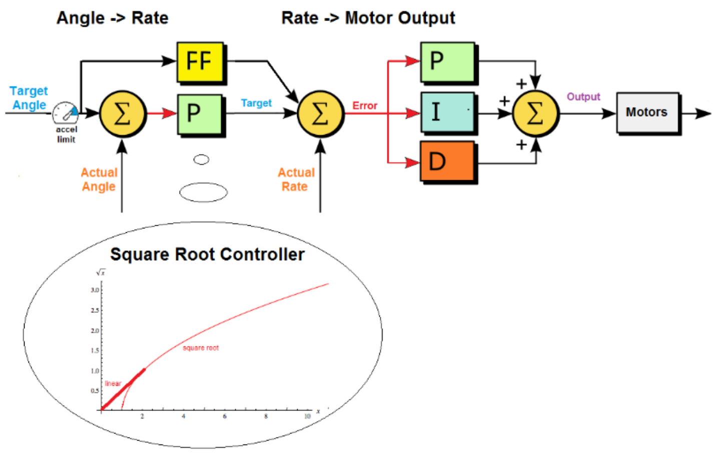


Figure 2: PID Control

1.1.2 - Key Role of Attitude Control

The attitude of a drone (its orientation in terms of roll, pitch, and yaw) is fundamentally crucial to its flight stability, navigation, and controllability. Maintaining the correct flight posture keeps a drone upright and airborne. Unlike airplanes that naturally stabilize along their forward motion (under ideal conditions), a drone is an inherently unstable system that must constantly correct its attitude to resist tilting. Additionally, turbulence (both externally or currents deflected off the ground) makes it even more difficult. In essence, attitude stabilization is the inner loop that guarantees smooth flight and stability. Without it, the drone will flip out of control.

Attitude control is also tightly linked to navigation. For a drone to move in any direction, it must first tilt towards that direction (so the thrust has that horizontal portion). Thus, precise attitude control allows the drone to accelerate, turn, and brake along a desired path. For example, to fly forward, a drone must pitch down a few degrees; to hover against the wind, it must roll slightly towards the wind. Accurate attitude control ensures these maneuvers happen with the right magnitude and timing, so that the drone's trajectory follows the pilot's intent (may it be automatic or artificial).

From a user's perspective, a better attitude controller translates to a "better" flying experience, as they don't have to constantly adjust the drone. Most drones include an automatic stabilization mode where the flight controller will self-level the drone. This makes the drone easier to fly, as the pilot doesn't have to constantly compensate for the drift.

1.1.3 - Potential of Reinforcement Learning in Attitude Control

Reinforcement learning (RL) treats control as a sequential decision-making problem, where an agent (the drone's controller) learns a policy by interacting with the environment and receiving feedback in the form of rewards. Unlike traditional model-based approaches, RL is model-free: it does not require an explicit dynamics model. Instead, the system learns an optimal control policy through trial-and-error, optimizing behavior to maximize cumulative rewards. Modern deep RL algorithms use neural networks to handle high-dimensional state inputs and continuous action outputs, making them well-suited for complex drone dynamics.

1.2 - Purpose and Problem

1.2.1 - Limitations of Traditional Controllers

Modern drones typically use a combination of hardware sensors (inertial sensors) and algorithms (PID).

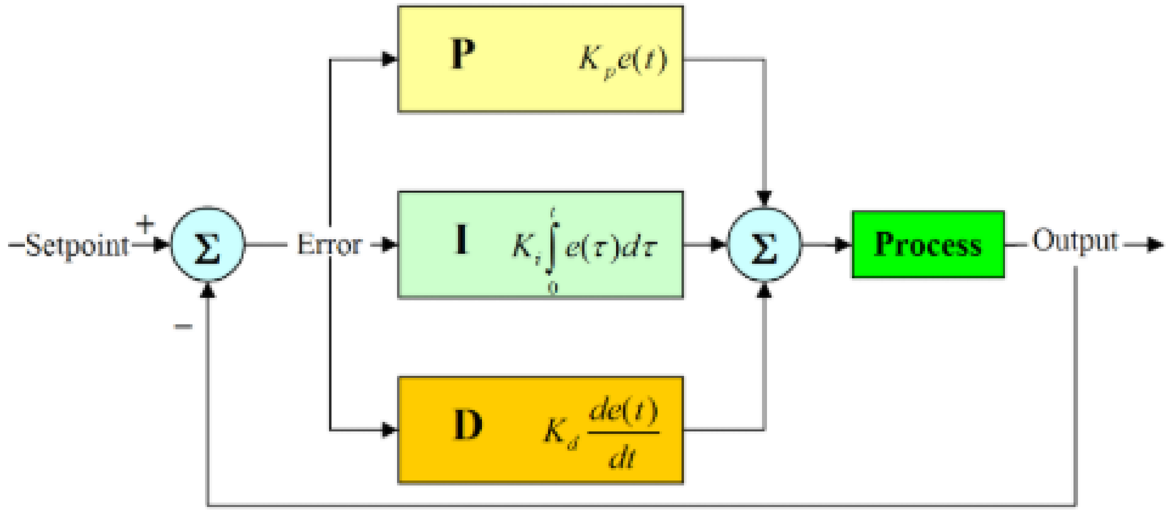


Figure 3: Simple PID Diagram

Nearly all drone flight controllers rely on an Inertial Measurement Unit (IMU), which contains a 3-axis gyroscope and accelerometer to sense the drone's motion and orientation. The gyroscope measures the angular velocity around the roll, pitch, yaw axis, while accelerometers measure linear accelerations to infer tilt. Together they provide measurement for the flight controller to maintain stability. However, raw IMU readings are noisy and prone to drift. To address this, flight controllers use sensor fusion filters to combine gyro and accelerometer data into a clean estimate of the drone's attitude. [2]

Proportional-Integral-Derivative (PID) controller has long been used for attitude control in drones. PID controllers are conceptually simple yet powerful, using three terms to correct error between desired state and actual state. In a drone a PID loop can adjust the motor outputs based on the attitude error or angular rate error, thereby compensating for deviations. Most drones use a cascaded PID control architecture, where the motors are regulated using an inner PID loop and an outer loop maintaining the angles.

Another category of control solution is the use of optimal control theory, such as Linear Quadratic Regulator (LQR), or more generally Model Predictive Control (MPC). These approaches leverage a mathematical model of the drone's dynamics to compute control inputs. LQR controllers have been designed for drone attitude stabilization with good results, as it can offer fast response and inherent balancing of trade-offs (like between response speed and control effort).

Besides the aforementioned algorithms, many modern drones also augment their spatial awareness using cameras and other sensors. They use the SLAM (Simultaneous Localization and Mapping) algorithm. The SLAM algorithm enables a drone to build a map of its environment and localize itself with that map in real time. In the context of consumer drones, SLAM is used for positioning and obstacle avoidance, which indirectly improves attitude control during autonomous flight.

1.2.2 - Applicability of PPO in Drone Attitude Control

PPO (Proximal Policy Optimization) is a learning-based controller. Instead of hand-tuning gains or relying on a fixed model, PPO learns a policy—a mapping from the drone's state (angles, rates, etc.) to motor commands—by practice (usually in simulation first). During training, it tries many actions, sees what works, and improves step by step. PPO adds a simple safety idea to training: only allow small, careful policy updates each round, which keeps learning stable.

It's particularly suitable for drone attitude control for the following reasons:

1. The drone's true behavior changes with battery voltage, prop damage, wind, and payload. PPO doesn't need a perfect model; it learns how to react from experience, including tricky cases (e.g., wind, ground effect), so it can handle nonlinear and changing dynamics naturally.
2. Instead of tuning KP, KI, KD (the values for a PID controller) by hand, you design a reward that encodes what you care about (small angle error, low rates, smoothness, low power). PPO then tunes itself to balance those goals. That makes it easier to target multiple objectives at once.
3. When you train in simulation with varied conditions (different masses, winds, sensor noise), PPO learns a single policy that works across them. This "practice on many scenarios" often yields robust behavior without per-scenario gain tables.
4. PPO can take in many signals at once (attitude, rates, motor temps, vibration metrics). Classical loops typically use a few signals and assume the rest is constant/insignificant. PPO can learn to weigh these inputs to make better choices.

1.2.3 - Points of Innovation

First, I adopted Reinforcement Learning algorithm (PPO) in drone attitude control to achieve better stability than traditional PID and LQR controllers, without spending too much extra computation budget.

Second, rather than report error only, I measured actuator command distribution and second-moment statistics to show that PPO achieves improved stability without unreasonable actuation spikes (smooth overall action instead of sudden twitches that minimizes error).

2 - Basics of Attitude Control

2.1 - Drone Dynamics Modeling

To model a drone's motion, I derive six coupled, nonlinear equations—three governing translation and three governing rotation. These arise from Newton's second law and Euler's rigid-body equations, framed in two coordinate systems: an Earth-fixed (inertial) frame and a body-fixed frame.

We define:

- **Inertial frame** $O_I(X_I, Y_I, Z_I)$: Origin at take-off point, with X_I and Y_I spanning the horizontal plane and Z_I pointing upward.
- **Body frame** $O_b(X_b, Y_b, Z_b)$: Origin at the drone's center of mass; X_b points forward, Y_b to the right wing, and Z_b upward through the frame.

Also, the vehicle's orientation is described by the Euler angles φ (roll about X_b), θ (pitch about Y_b), and ψ (yaw about Z_b).

We label the four rotors (each with a fixed-pitch propeller) as follows, viewed from above:

- Rotor 1 at the front ($+X_b$).
- Rotor 2 on the right ($+Y_b$).
- Rotor 3 at the rear ($-X_b$).
- Rotor 4 on the left ($-Y_b$).

Adjacent rotors spin in opposite directions to balance net reaction torque:

- Rotors 1 & 3 spin clockwise (CW).
- Rotors 2 & 4 spin counterclockwise (CCW).

2.1.1 - Attitude Representation

To express a vector v_b in body coordinates into inertial coordinates v_I, we use the rotation matrix

$$R(\varphi, \theta, \psi) = R_z(\psi) R_y(\theta) R_x(\varphi)$$

where, for example,

$$R_x(\varphi) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\varphi) & -\sin(\varphi) \\ 0 & \sin(\varphi) & \cos(\varphi) \end{pmatrix}$$

and similarly for R_y and R_z. Thus

$$v_I = R(\varphi, \theta, \psi) v_b$$

2.1.2 - Translational Dynamics

Applying Newton's second law in the inertial frame,

$$m\ddot{r} = \Sigma F$$

where $r = \begin{pmatrix} x \\ y \\ z \end{pmatrix}$ is the the position, m is the mass, and the external forces are:

1. Thrust F_T: All four rotors produce upwards thrust $T_i = k\omega_i^2$. Summed as $U_1 = k \sum_{i=1}^{4} \omega_i^2$m, this force acts along Z_b and in inertial coordinates becomes

$$F_T = R(\varphi, \theta, \psi) \begin{pmatrix} 0 \\ 0 \\ U_1 \end{pmatrix}$$

2. Gravity F_g: The gravitational force, where in inertial coordinates is

$$F_g = \begin{pmatrix} 0 \\ 0 \\ -mg \end{pmatrix}$$

3. Aerodynamic drag F_D: Using a linearized model that is valid at moderate speeds, Aerodynamic drag can be modelled as

$$F_D \approx \begin{pmatrix} c_x \dot{x} \\ c_y \dot{y} \\ c_z \dot{z} \end{pmatrix}$$

Combining these yields component wise equations:

$$\begin{aligned} \ddot{x} &= \frac{U_1}{m}(\cos(\varphi)\sin(\theta)\cos(\psi) + \sin(\varphi)\sin(\psi)) - \frac{c_x}{m}\dot{x} \\ \ddot{y} &= \frac{U_1}{m}(\cos(\varphi)\sin(\theta)\cos(\psi) + \sin(\varphi)\sin(\psi)) - \frac{c_y}{m}\dot{y} \\ \ddot{z} &= \frac{U_1}{m}(\cos(\varphi)\cos(\theta)) - g - \frac{c_z}{m}\dot{z} \end{aligned}$$

These capture how tilting the drone (through φ, θ) redirects thrust into horizontal motion, and how drag and gravity oppose its movement.

2.1.3 - Rotational Dynamics

Euler's equations for rotation about the center of mass are

$$M = I\dot{\omega} + \omega \times (I\omega)$$

where $\omega = \begin{pmatrix} p \\ q \\ r \end{pmatrix}$ is the angular velocity in body aces ($p = \dot{\varphi}$), and $I = \begin{pmatrix} I_x & 0 & 0 \\ 0 & I_y & 0 \\ 0 & 0 & I_z \end{pmatrix}$ is the inertia tensor.

The total moment $M = \begin{pmatrix} M_\varphi \\ M_\theta \\ M_\psi \end{pmatrix}$ has three contributions:

1. Different thrust between rotor pairs creates roll, pitch, and yaw moments::
   - Roll moment M_φ (about X_b) from right vs. left rotors:

$$U_2 = M_\varphi = l(k\omega_4^2 - k\omega_2^2) = lk(\omega_4^2 - \omega_2^2)$$

   - Pitch moment M_θ (about Y_b) from front vs. back rotors:

$$U_3 = M_\theta = l(k\omega_3^2 - k\omega_1^2) = lk(\omega_3^2 - \omega_1^2)$$

   - Yaw moment M_ψ (about Z_b) from net rotor drag torque difference:

$$U_4 = M_\psi = k_{d[(\omega_1^2 + \omega_3^2) - (\omega_2^2 + \omega_4^2)]}$$

where k_d ($\frac{N \cdot m \cdot s^2}{\text{rad}^2}$) is the rotor's drag torque coefficient.

2. Gyroscopic torques When the body angular rates p, q tilt the spinning rotors, conservation of rotor angular momentum produces gyroscopic moments:

$$M_{\text{gyro}} = J_r \omega \times \begin{pmatrix} 0 \\ 0 \\ \Omega \end{pmatrix}, \qquad \Omega = (\omega_2 + \omega_4) - (\omega_1 + \omega_3)$$

This expands to torques about X_b and Y_b:

$$M_{\text{gyro}} = J_r \Omega \begin{pmatrix} -q \\ p \\ 0 \end{pmatrix}$$

3. Rotational damping A linear viscous damping opposing angular rates:

$$M_d = -\begin{pmatrix} d_\varphi p \\ d_\theta q \\ d_\psi r \end{pmatrix}$$

Finally, to obtain the roll,pitch, yaw equations, we substitute $M = \begin{pmatrix} U_2 \\ U_3 \\ U_4 \end{pmatrix} + M_{\text{gyro}} + M_D$ into Euler's equation to obtain:

$$\begin{aligned}
\ddot{\varphi} &= \frac{I_y - I_z}{I_x} qr - \frac{J_r \Omega}{I_x} q + \frac{U_2}{I_x} - \frac{d_\varphi}{I_x} p \\
\ddot{\theta} &= \frac{I_z - I_x}{I_y} pr - \frac{J_r \Omega}{I_y} p + \frac{U_3}{I_y} - \frac{d_\theta}{I_y} q \\
\ddot{\psi} &= \frac{I_x - I_y}{I_z} pq + \frac{U_4}{I_z} - \frac{d_\psi}{I_z} r
\end{aligned}$$

Each term's origin is now clear:
- Inertia coupling ($\frac{I_y - I_z}{I_x} qr$): arises because I is anisotropic.
- Gyro torque ($\frac{J_r \Omega}{I_x} q$) from spinning rotors reacting to pitch/roll.
- Control input ($\frac{U_2}{I_x}$): direct moment from thrust difference.
- Damping ($\frac{d_\varphi}{I_x} p$): viscous resistance to roll rate.

2.2 - Traditional Control Algorithms

2.2.1 - PID Control and LQR Control

A proportional-integral-derivative (PID) controller is one of the most commonly used control algorithms in industrial systems, designed to maintain a desired output by continuously adjusting the inputs. It works by calculating the error between a desired set point and the actual process value, then using three terms—proportional, integral, and derivative—to correct the system. Together, these terms allow for precise and stable control in a wide variety of systems, from temperature regulation to motor speed control.\

In the following paragraphs, I will introduce the transform functions of PID. Here, $e(t)$ means the error at t in the time domain.

- The proportional term reacts to the current error:

Within the time domain, we can describe this part with the formula: $p(t) = K_p e(t)$

- The integral term accumulates past errors to eliminate steady-state bias:

Within the time domain, we can describe this part with the formula: $i(t) = K_i \int_0^{\{t\}} e(\tau) d\tau$

- The derivative term anticipates future errors based on the current rate of change.

Within the time domain, we can describe this part with the formula: $d(t) = K_d \frac{d}{dt} e(t)$

Combining the terms together, we will have our full PID:

$$y(t) = K_p \ e(t) + K_i \int_0^t e(\tau) d\tau + K_d \frac{d}{dt} e(t)$$

Where $y(t)$ is our output.

Whereas PID uses local error feedback with heuristic tuning, Linear–Quadratic Regulator (LQR) exploit an explicit model of the dynamics.

For drones, one linearizes the dynamics about hover to obtain

$$\dot{\delta x} = A \delta x + B \delta u,$$

where δx stacks small deviations in angles, rates (and possibly position/velocity), and δu are small control increments. LQR selects a **full-state feedback**

$$\delta u = -K \delta x$$

that minimizes the infinite-horizon quadratic cost

$$J = \int_0^\infty (\delta x^T Q \delta x + \delta u^T R \delta u) dt$$

In practice:

- LQR handles axis coupling naturally (one gain matrix for all channels).
- Tuning via Q, R is systematic, trading agility against effort.
- Runtime cost is light (a matrix–vector multiply), suiting embedded controllers.
- For unmeasured states, one pairs LQR with a state estimator (e.g., Kalman filter), yielding LQG; integral augmentation (LQR-I) removes steady-state offsets.

However, because LQR relies on a linearized model, performance is best for small deviations about hover; large angles, aggressive maneuvers, prop-wash, or parameter shifts (battery, payload) degrade the model fit. Common remedies include gain scheduling and feedforward terms. [3], [4], [5]

2.2.2 - Limitations of Traditional Controller in Nonlinear Systems

Despite the widespread use of PID, LQR and other traditional controllers, these methods encounter significant limitations when applied to the highly nonlinear and coupled dynamics of a drone. A drone is an inherently multivariable, under-actuated system with strong coupling between translational and rotational motions. Classical controllers, especially those based on linearization or fixed gains, struggle to maintain performance across the full flight. Key challenges associated with traditional control approaches include:

- Modelling Inaccuracies: Classical designs often assume a reasonably accurate mathematical model of the drone. In reality, factors like unmodeled aerodynamic effects, parameter uncertainties (mass distribution, motor thrust), and time varying mechanics (battery voltage dropping) can all cause the true system to deviate from mathematical model. A controller tuned on an approximate or simplified model may thus exhibit degraded performance or even instability when these discrepancies arise. For example, feedback linearization can in theory cancel nonlinearities exactly, but it requires an exact model; if the actual quadrotor differs from the model, precision is lost and the controller is no longer effective [6]
- Disturbance Rejection and Robustness: Handling external disturbances (wind, ground effects) and unmodeled dynamics is a persistent difficulty for traditional controllers. A basic PID has no explicit mechanism to reject unseen disturbances. Similarly, an optimal LQR regulator designed for a specific operating point can perform poorly if the system is pushed outside that regime or if specific disturbances push the state into nonlinear region.
- Gain Tuning and Stability Margins: Practical implementation of PID controllers on a quadrotor requires careful tuning of gains (PID variables) to achieve a balance between responsiveness and stability. This tuning process is labor-intensive and often needed to be repeat multiple times for different conditions or platform. A PID tuned for hover, may not work optimally for fast forward flight or aggressive maneuvers, leading to oscillations or slow response if not re-tuned. Furthermore, there is systematic way to tune PIDs for multi-axis coupling; engineers often resort to trial-and-error or heuristic methods which do not guarantee optimal performance.

In light of these issues, many recent studies have critically evaluated classicl control methods on drones and often concluded that while PID, LQR, and other control methods work well in basic conditions, they struggle in advanced or unanticipated situations. [6]

2.3 - Basic Theory of Reinforcement Learning

2.3.1 - Markov Decision Process (MDP) Theoretical Framework

A Markov Decision Process (MDP) is a mathematical framework used in reinforcement learning to model and analyze an agent's decision-making process within an environment. MDPs provide a formal means of describing the interaction between agent and environment, and furnish the theoretical foundation for the design and evaluation of reinforcement learning algorithms.

Within an MDP, the agent's goal is to identify a policy π, which prescribes for each state the action that maximizes the expected cumulative discounted reward starting from that state. The cumulative discounted reward (often called the return) G_t is defined as

$$G_t = \sum_{k=0}^{\infty} \gamma^k R(s_t, a_t, s_{t+1})$$

where s_t and a_t represent the state and action at time t respectively. S denotes the state space, i.e., the set of all possible states. Each state $s \in S$ describes a specific configuration of the environment. A denotes the action space, i.e., the set of all possible actions. Each action $a \in A$ is an operation that the agent can execute in a given state.

$P(s' \mid s, a)$ denotes the state-transition probability: given the current state s and action a, it is the probability of transitioning into the next state s'.

$R(s, a, s')$ denotes the reward function: the immediate reward the agent receives after taking action a in state s and transitioning to state s'.

γ is the discount factor, a number $\in [0, 1]$ that determines the present value of future rewards. $\gamma = 0$ means only immediate rewards matter, while $\gamma = 1$ treats future rewards as equally important as immediate rewards.

To evaluate a policy, we introduce the state-value function and the action-value function:

- State-value function $V^{\pi(s)}$: under policy π, the expected cumulative discounted return starting from state s.
- Action-value function $Q^{\pi(s,a)}$: under policy π, the expected cumulative discounted return starting from state s, taking action a, and then following π.

These two functions satisfy the following Bellman equations:

$$V^{\pi(s)} = \max_a \left[R(s, a, s') + \gamma \sum_{s'} P(s' \mid s, a) V^{\pi(s')} \right]$$
$$Q^{\pi(s,a)} = R(s, a, s') + \gamma \sum_{s'} P(s' \mid s, a) V^{\pi(s')}$$

Here, $\sum_{s'}$ denotes summation over all possible next states s'.

An MDP (Markov Decision Process) provides a theoretical foundation for reinforcement learning by formalizing the agent's decision-making problem. By specifying states, actions, transition probabilities, and rewards, one can construct models to evaluate and optimize policies to maximize cumulative return.

2.3.2 - Core Concept of Reinforcement Learning

Reinforcement Learning (RL). Reinforcement learning is a machine-learning paradigm that studies how an agent should act in an environment in order to maximize its cumulative reward. Its core idea is to learn an optimal policy through trial-and-error, enabling the agent to make optimal decisions in the environment.

Agent and Environment. In this project, the agent is the drone attitude-control system, whose objective is to regulate motor outputs to maintain or change the vehicle's attitude for balance. The environment comprises the drone's physical hardware, sensors (e.g., IMU), actuators (e.g., motors), and the external operating conditions (e.g., airflow, wind speed).

State. A state represents the drone's instantaneous information at a given time, including attitude angles (roll φ, pitch θ, yaw ψ), angular rates (p, q, r), as well as position and linear velocity, etc. These quantities are provided by the IMU and other onboard sensors.

Action. An action is the control applied by the agent to the environment. In this drone setting, actions correspond to adjusting the rotational speeds of the four motors to generate different lift and torques, thereby changing or maintaining the drone's attitude.

Reward. The reward *r* r is feedback from the environment in response to the agent's behavior. For an attitude-stabilization task, the reward function can be designed using factors such as attitude error, angular-rate magnitude, and motor power. For example, higher rewards can be given when the attitude is close to the target and angular rates are small; conversely, large attitude errors or excessive angular rates can incur lower rewards or penalties.

Policy. A policy specifies the rule by which the agent selects actions given a state. In this project, the policy is represented by a deep neural network that takes the current drone state as input and outputs the corresponding action (motor-speed adjustments). Using a reinforcement-learning algorithm (e.g., PPO), the policy is iteratively updated and optimized to increase cumulative reward.

Value Function. The value function evaluates the expected long-term cumulative reward of being in a particular state under the current policy. It helps the agent assess state quality and thus guides the selection of better actions.

Action-Value Function. The action-value function (or Q-function) evaluates the expected long-term cumulative reward of taking a particular action in a given state and then following the current policy thereafter. This helps the agent choose optimal actions for specific states.

Discount Factor. The discount factor γ balances immediate and future rewards. In attitude stabilization, choosing an appropriate γ allows the agent to prioritize immediate stability while also accounting for longer-term flight performance.

Exploration and Exploitation. Exploration refers to trying novel actions to discover control strategies that may yield higher rewards; exploitation refers to leveraging the current best-known strategy to maximize cumulative reward.

3 - PPO Algorithm

3.1 - Introduction

Proximal Policy Optimization (PPO) is an optimization algorithm for training reinforcement-learning models that aims to improve the stability and reliability of policy updates. In standard policy-gradient methods, an update can push the policy distribution too far, which destabilizes training. PPO introduces a constraint that keeps each update within a bounded range, thereby improving both training stability and sample efficiency.

The motivation for PPO is the difficulty of controlling the update magnitude in policy-gradient methods. In traditional approaches, the step size critically affects stability: a step that is too large causes excessive policy shifts and instability, while a step that is too small leads to slow convergence. PPO incorporates the notion of a trust region, constraining the difference between the new and old policies during each update (e.g., via clipping or a KL-penalty), thus avoiding these problems.

3.1.1 - Background and Motivation of PPO

In reinforcement learning, taking unconstrained policy gradient steps can lead to overshooting – excessively large updates that destabilize training [7]. Trust Region Policy Optimization (TRPO) (Schulman et al., 2015) addressed this by enforcing a constraint on the Kullback–Leibler (KL) divergence between the new and old policies, thereby limiting how far the policy can move in a single update.

The motivation for PPO is the difficulty of controlling the update magnitude in policy-gradient methods. In traditional approaches, the step size critically affects stability: a step that is too large causes excessive policy shifts and instability, while a step that is too small leads to slow convergence. PPO incorporates the notion of a trust region, constraining the difference between the new and old policies during each update (e.g., via clipping or a KL-penalty), thus avoiding these problems. In theory one could add a large penalty for KL divergence to achieve the same effect, but the theoretically recommended penalty coefficient would force impractically small step sizes [8]. TRPO's hard KL constraint provides a reliable way to take larger steps without overshooting, ensuring each update improves performance monotonically under a trust region limit.

Proximal Policy Optimization, introduced by Schulman et al. in 2017, is a first-order optimization approach that retains the benefits of TRPO's trust region method while being much simpler to implement. PPO achieves performance on par with or better than TRPO in many tasks (e.g. Atari games), yet it avoids TRPO's complex second-order optimization, making it computationally more efficient [9]. The core idea of PPO is to keep policy updates within a safe region around the old policy (hence "proximal") using a modified objective function that can be optimized with standard stochastic gradient descent techniques (e.g. Adam). Schulman et al. proposed two main variants of PPO's surrogate objective: one that uses an adaptive KL penalty and one that uses a clipped probability ratio.

3.1.2 - Core Principals and Mathematical Derivation of PPO

TRPO derives a theory-backed update rule that contains each policy step using a KL-divergence-based trust region, yielding conservative updates with monotonic improvement guarantees. In practice, however, directly penalizing the KL term can be overly stringent and lead to very small steps unless the penalty weight is carefully tuned. Moreover, a single penalty coefficient that works across tasks, or even across phases of training on the same task, can be hard to pick. [8]

A useful way to describe the KL-penalized step is:

$$\Delta\theta^* = \arg\max_{\Delta\theta} L_{\theta+\Delta\theta} - \beta D_{\mathrm{KL}}(\pi_\theta \parallel \pi_{\theta+\Delta\theta})$$

where L is the policy gradient term and $\beta > 0$ is the KL penalty coefficient. The central difficulty is choosing β so that updates are neither too timid nor aggressive. [8]

With PPO, one widely used variant (PPO Clip) replaces the explicit KL penalty with a clipped probability-ratio objective. Let

$$r_{t(\theta)} = \frac{\pi_{\theta(a_t \mid s_t)}}{\pi_{\theta_{\mathrm{old}}}(a_t \mid s_t)}, \hat{A}_t \text{ an advantge estimate}$$

and choose a small clipping hyper-parameter ε. The clipped surrogate is

$$L_{\mathrm{clip}}(\theta) = \mathbb{E}\Big[\min\Big(r_{t(\theta)}\hat{A}_t,\quad \mathrm{clip}\Big(r_{t(\theta)}, 1-\varepsilon, 1+\varepsilon\Big)\hat{A}_t\Big)\Big]$$

if r_t tries to move outside of $[1-\varepsilon, 1+\varepsilon]$, the term is surrogated, so pushing further gives no extra improvement in the objective (the gradient contribution vanishes in those regions). Intuitively, positive advantage actions can only be up-weighted to about $1+\varepsilon$ times their old probability, and negative actions can only be down-weighted to about $1-\varepsilon$. This prevents oversize policy jumps while still permitting meaningful changes. The original PPO paper shows that clipping yields a conservative lower bound on the unconstrained objective.

PPO clipped is generally preferred as it's much easier to implement while still maintaining trust region like behavior. The original study also found PPO clipped to be more stable and generally better-performing than PPO penalty (another variant that relies on an adaptive KL penalty, but due to relevance, I will not go too deep into how it works) across multiple benchmarks.

3.1.3 - Construction and Optimization of Objective Function

Within PPO, the objective function is commonly a clipped function, to control the steps of policy updates.

$$L_{\mathrm{clip}}(\theta) = \mathbb{E}\Big[\min\Big(r_{t(\theta)}\hat{A}_t,\quad \mathrm{clip}\Big(r_{t(\theta)}, 1-\varepsilon, 1+\varepsilon\Big)\hat{A}_t\Big)\Big]$$

where, $r_{t(\theta)} = \frac{\pi_{\theta(a_t \mid s_t)}}{\pi_{\theta_{\mathrm{old}}}(a_t \mid s_t)}$ is the probability, $\hat{A}_t$ is the advantage estimate, ε is the super parameter for the clipping region. This objective function guarantees the steps taken won't be overly aggressive, which improves stability.

Update the policy parameters θ by performing gradient ascent to maximize the clipped objective. At each update, compute the policy loss $L_{\mathrm{clip}}(\theta)$, then use backpropagation to obtain the gradients and apply an optimizer to update the parameters of the policy network.

Objective Function Construction and Optimization process for Value Function Update

The goal of updating the value function is to make its predictions as close as possible to the ground truth values. This is typically achieved by minimizing the mean squared error between the predicted value and the actual target. Specifically, the optimization objective for the value function is:

$$L_{\text{VE}}(\theta) = \mathbb{E}_{t\left[(V_\theta(s_t) - V_{\text{target}}(s_t))^2\right]}$$

where $V_{\text{target}}(s_t)$ is the target value, often computed using a smoothed advantage estimate. The value-function parameters φ are then updated by gradient descent to minimize this mean squared error loss at every step.

4 - Application and Design of PPO in Drone Attitude Control

4.1 - Experimental Envrionment

In this project, I have chosen the CQ230 Open Source Drone Development Kit as our platform. This kit combines a Raspberry Pi 4B with a Pixhawk 2.4.8 flight controller (FC) to form a compact quadrotor. The Pixhawk FC runs the open source ArduPilot firmware, while the Raspberry Pi hosts development tools. Together, these features make the CQ230 kite ideal for my project.

4.1.1 - Overall Setup

The drone design comprises two main aspects: the structural design and the electrical design. The CQ230 kit provides all necessary hardware components, allowing us to focus primarily on algorithm implementation and system integration.

4.1.2 - Structural Design

The CQ230 drone kit uses a custom designed anti-collision frame as its structural backbone.

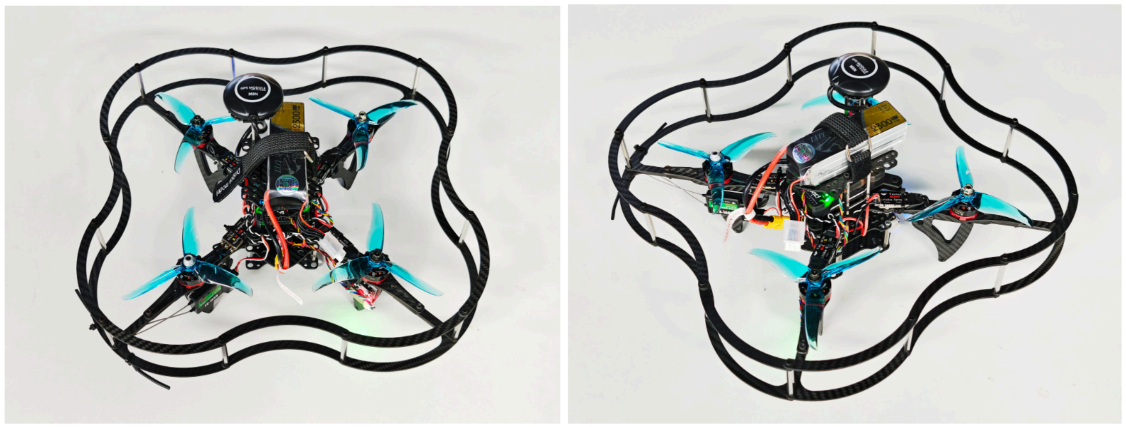
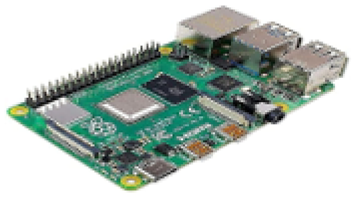
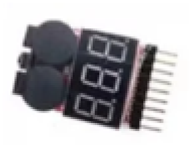


Figure 4: Top down view of the drone

The frame has a 230 mm diagonal motor spacing in a symmetric quadcopter layout. The overall dimensions are approximately 350 by 360 by 300 mm, and the fully assembled weight is around 612 grams. This compact, lightweight design enables operation in confined, indoor spaces. In summary, the CQ230's structural design balances miniaturization and robustness, providing a reliable hardware foundation for our experiments.

4.1.3 - Hardwares

Name	Image	Description
Pixhawk 2.4.8 Flight Controller		Acts as the drone's brain. It features a 32-bit STM32F427 processor, an onboard BMP561 barometer, and multiple I/O ports.
Raspberry Pi 5B		Serves as the companion computer, running Ubuntu with pre-installed libraries. It handles high-level tasks such as computer vision, path planning, and network communication.
Brushless Motors (2205)		Four 2205-series brushless motors paired with 5045 propellers deliver thrust and maneuverability.
Electronic Speed Controllers (30 A ESC)		Each motor is controlled by a 20 A ESC, which receives PWM signals from the Pixhawk and precisely regulates motor RPM.
Battery (4S, 16.8V 2300 mAh)		A single 4S battery provides enough power to sustain 7 minutes of flight.

Name	Image	Description
Power Module (Ledi Mini Pix)		Distributes battery power to the FC and provides voltage/current telemetry back to the FC for low-voltage warnings and failsafe logic.
Optical Flow Sensor (MF-01)		Combines a downward-facing optical flow camera with a laser/ultrasonic rangefinder to provide position and height feedback.
GPS Module (Ublox M8N)		Provides outdoor GNSS positioning with 2-3 m horizontal accuracy.
Telemetry Radio (CUAV Wi-Fi Module)		Establishes a Wi-Fi link between the FC and ground station.
RC Transmitter/Receiver (FlySky FS-i6)		A 6-channel 2.4 GHz transmitter and receiver allow manual control during testing and a range up to 700 meters.
Buzzer (BB Alarm Buzzer)		Signals various flight events through audible alerts.

4.1.4 - Software Environment and Simulation Setup

Operating system: On the drone, a Raspberry Pi 5B is used, with Ubuntu 24 installed, combined with Pixhawk to realize the control of the drone. Ground station uses: Windows + Anaconda + PyCharm/ VScode. Programming language and toolchain: Python is used for the development of the PPO algorithm, and reinforcement learning libraries such as Stable-Baselines3 are used to accelerate implementation.

Simulation platform: A UAV dynamics simulation environment based on PyBullet is constructed for algorithm training and preliminary verification.

4.2 - Reinforcement Learning Framework Design

4.2.1 - Action Space and State Space Definition

The action space A is defined as:

$$A = [\Delta\omega_1, \Delta\omega_2, \Delta\omega_3, \Delta\omega_4]$$

where:

- $\Delta\omega_1, \Delta\omega_2, \Delta\omega_3, \Delta\omega_4$ represent the variations in the rotational speeds of the four motors.
- The range of rotational speed variation is $[-0.08, 0.08]$, expressed in terms of relative changes in rotational speed. This formulation facilitates exploration and learning for the reinforcement learning algorithm.

```
def __init__(self):
    super(DroneGymEnv, self).__init__()

    # Define State Space
    self.state_dim = 15
    self.state_space = Box(low=-np.inf, high=np.inf, shape=(self.state_dim,),
dtype=np.float32)

    # Define Action Space
    self.action_dim = 4
    self.action_space = Box(low=-0.08, high=0.08, shape=(self.action_dim,),
dtype=np.float32)
```

4.2.2 - Reward Function

The design of the reward function is crucial for the performance of reinforcement learning algorithms. In the task of attitude stabilization control of drones, the reward function R is defined as:

$$R = \alpha R_1 + \beta R_2 + \gamma R_3 + \delta R_4 + \varepsilon R_5$$

- R_1: Positional deviation reward: $R_1 = -\ |x - x_{\text{target}}| - |y - y_{\text{target}}| - |z - z_{\text{target}}|$
- R_2: Velocity reward: $R_2 = -\ |v_x| - |v_y| - |v_z|$
- R_3: Attitude reward: $R_3 = -\ |\theta| - |\varphi| - |\psi|$
- R_4: Angular velocity reward: $R_4 = -\ |\omega_x| - |\omega_y| - |\omega_z|$
- R_5: Control effort reward: $R_5 = -\ |\Delta F_1| - |\Delta F_2| - |\Delta F_3| - |\Delta F_4|$

Here, $\alpha, \beta, \gamma, \delta, \varepsilon$ are the weighting coefficients. Their relative weighting is set such that:

$$\gamma > \alpha, \beta, \delta > \varepsilon$$

which ensures attitude stability is prioritized.

```
def _calculate_reward(self, state, action):
    # Extract information
    x, y, z, v_x, v_y, v_z, pitch, roll, yaw, omega_x, omega_y, omega_z, wind_x, wind_y,
wind_z = state
    delta_omega1, delta_omega2, delta_omega3, delta_omega4 = action  # 动作

    # Components
    r_position = - (abs(x) + abs(y) + abs(z))
    r_velocity = - (abs(v_x) + abs(v_y) + abs(v_z))
    r_attitude = - (abs(pitch) + abs(roll) + abs(yaw))
    r_angular_velocity = - (abs(omega_x) + abs(omega_y) + abs(omega_z))
    r_action = - (abs(delta_omega1) + abs(delta_omega2) + abs(delta_omega3) +
abs(delta_omega4))

    # Overall
    reward = 0.1 * r_position + 0.2 * r_velocity + 0.3 * r_attitude + 0.3 *
r_angular_velocity + 0.1 * r_action

    return reward
```

In the experiments, the weighting parameters will be tuned according to the specific tasks and environmental conditions of the drone, in order to achieve optimal control performance. The reward function is designed to comprehensively account for the drone's position, velocity, attitude, angular velocity, and control effort, thereby guiding the drone toward stable flight in complex environments.

4.2.3 - Model Evaluation

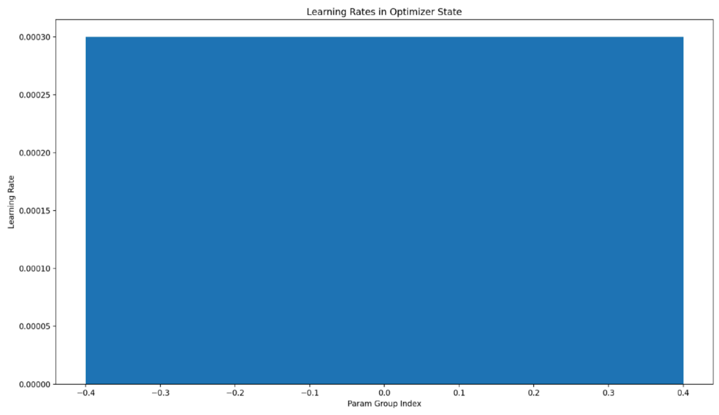


Figure 5: Learning rate distribution for different parameter groups

This Figure illustrates the distribution of learning rates across different parameter groups in the optimizer. It can be observed that all parameter groups share a constant learning rate of 0.0003, indicating that the optimizer adopts a unified learning rate strategy. Such a configuration helps maintain convergence and stability during training, preventing instability that may arise from large discrepancies in learning rates. The choice of 0.0003 was based on preliminary hyper parameter tuning experiments, where this value consistently delivered good performance—striking a balance between rapid convergence and avoiding oscillations caused by excessively large update steps. Future work may explore dynamic learning rate adjustment strategies, such as learning rate decay or adaptive learning rate methods, to achieve better performance across different training stages.

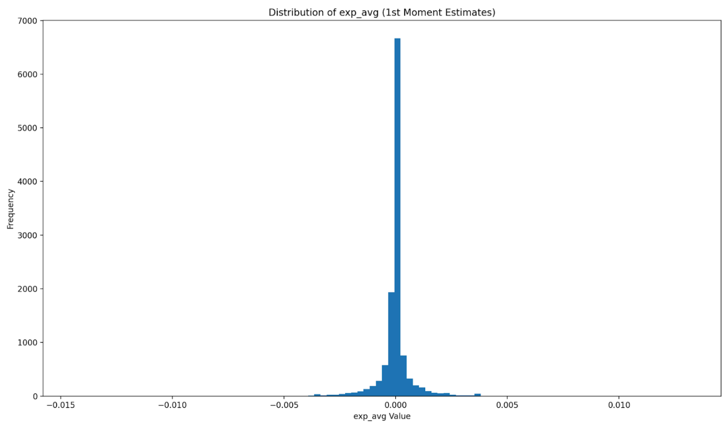


Figure 6: First moment estimation (exp_avg)

This figure shows the histogram of the frequency distribution of `exp_avg` values, where the horizontal axis represents the `exp_avg` values and the vertical axis represents frequency. The results indicate that most `exp_avg` values are concentrated around zero, forming a sharp peak. This distribution suggests that the algorithm tends to reduce the magnitude of policy updates during training, thereby promoting more stable policy improvement. The sharpness of the distribution also implies relatively low variance in updates, which benefits learning efficiency and stability. However, such a distribution may limit the algorithm's ability to explore new strategies, since most update steps are small. Future research may focus on balancing exploration and exploitation to enhance adaptability and flexibility in more complex environments.

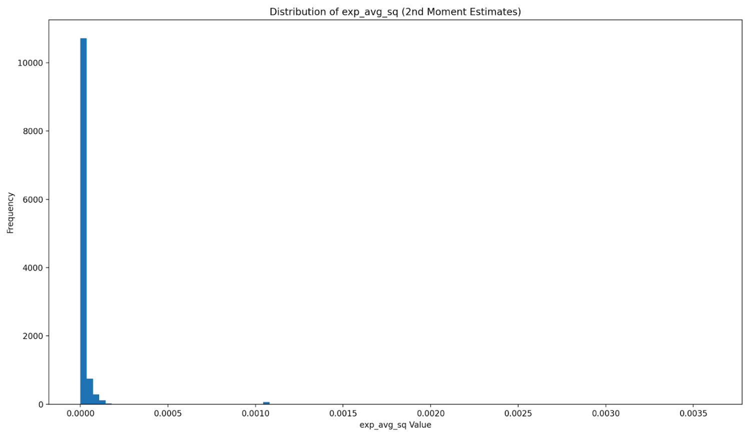


Figure 7: Frequency distribution histogram of exp_avg_sq values

This figure presents the histogram of the frequency distribution of `exp_avg_sq` values. The results show an even sharper concentration around zero, with nearly all estimates clustered extremely close to zero. This highly concentrated distribution indicates effective control over the second-moment estimates of policy updates, which likely helps reduce variance in updates and further improves training stability.

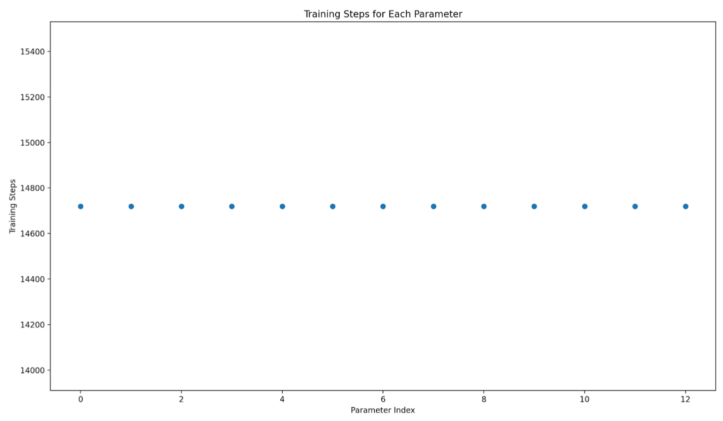


Figure 8: The number of training steps corresponding to each parameter index

This figure depicts the number of training steps corresponding to each parameter index. The data reveal that all parameters were updated at a relatively stable rate of approximately 14700 steps, suggesting balanced updates across all model parameters. Such uniformity ensures that every parameter receives sufficient training, thereby preventing under-trained parameters from degrading overall model performance.

In this setup, the total number of time steps was set to 95000, with a fixed learning rate of 0.0003 to ensure stable learning. The discount factor was set to 0.99 to balance the importance of immediate and future rewards. For variance reduction and bias improvement, we adopted the Generalized Advantage Estimator (GAE) with $\lambda = 0.95$. The entropy coefficient was set to 0 to avoid encouraging randomness during training, while the value function coefficient was set to 0.5 to balance updates between policy and value function. To prevent gradient explosion, the maximum gradient norm was clipped at 0.5. Each training iteration used a batch size of 64, with 10 epochs per update. Finally, the clipping range of PPO was set to 0.2, limiting policy update magnitude to further enhance training stability.

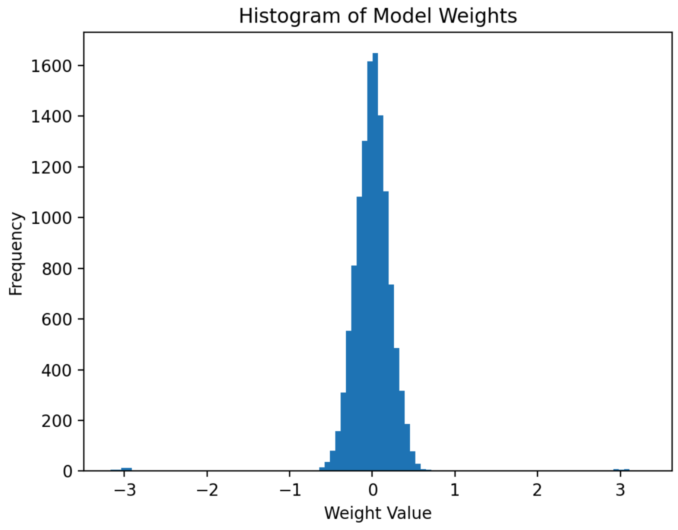


Figure 9: Histogram of Model Weights

This figure reveals that the weight values are primarily concentrated around zero and approximately follow a normal distribution, with the highest frequency occurring between -1 and 1. This distribution indicates that the model weights were effectively regularized during training, which helps prevent overfitting and improves generalization. Moreover, maintaining a well-balanced weight distribution is crucial for model performance and stability, since excessively large or small weights can lead to overfitting or underfitting.

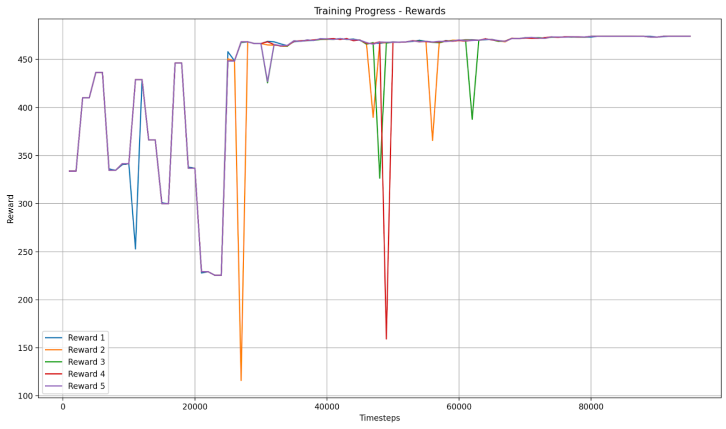


Figure 10: Training Progress - Rewards

PPO training exhibits an early exploratory phase followed by convergence; Reward 1 and Reward 5 dominate in magnitude and stability (≈ 450) in later training, indicating their primary role in optimizing drone attitude stabilization, while other reward terms contribute less.

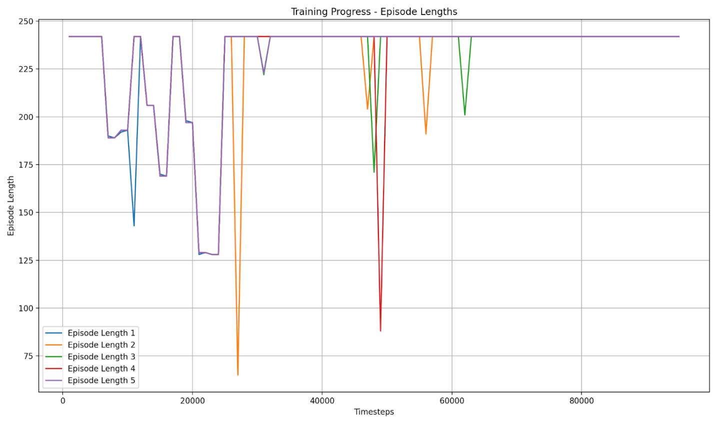


Figure 11: Training Progress - Episode Length

Episode lengths show high variability during exploration but converge to stable values across all five episode types as training proceeds, indicating policy maturation and consistent task performance.

4.3 - Experimentation

4.3.1 - Simulation Verification

Wind Level	**Wind Speed** (unit: `m/s`)
level 0	0
level 1	[0.3, 1.5]
level 2	[1.6, 3.3]
level 3	[3.4, 5.4]
level 4	[5.5, 7.9]
level 5	[8.0, 10.7]

* There are more levels, but for the purposes of this research, up to level 5 is enough.

Table 1: Wind Levels up to 5

To comprehensively evaluate the performance of Proximal Policy Optimization (PPO) versus a conventional PID controller for drone attitude stabilization, we conducted a set of simulation experiments in the PyBullet environment. Simulations covered a range of wind conditions from Beaufort-scale equivalent level 0 to level 5 (approximately 0 to 8.0 m/s). For each wind condition we recorded key performance metrics, including attitude angles (pitch, roll, and yaw), position error, angular velocities, and control command magnitudes. These measurements were used to quantify stability, tracking accuracy, control effort, and robustness to aerodynamic disturbance.

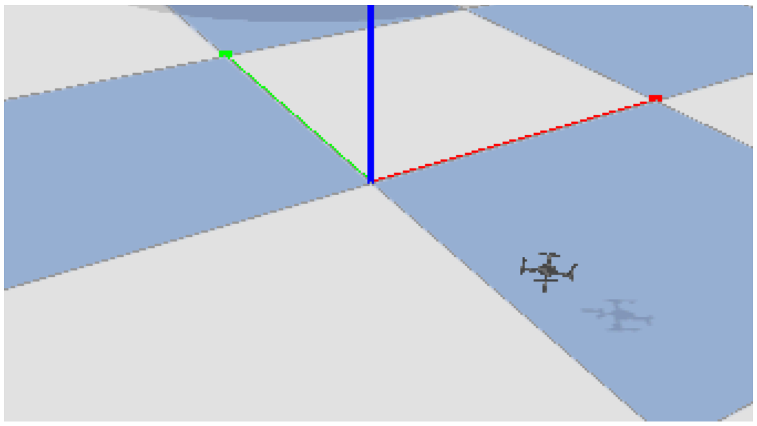


Figure 12: PyBullet Simulation Demonstration

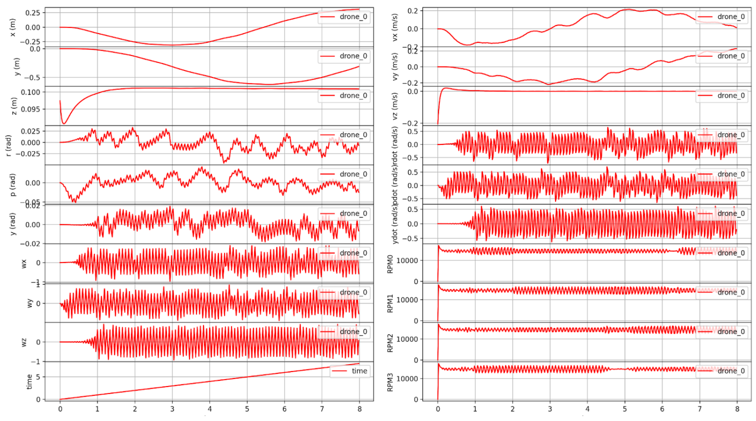


Figure 13: PyBullet Simulation with 0 Wind Speed

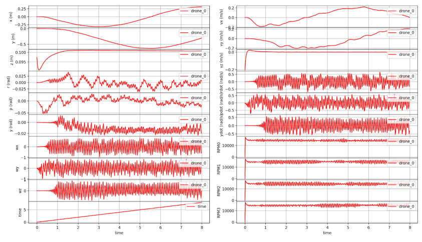


Figure 14: PID at 0.3m/s

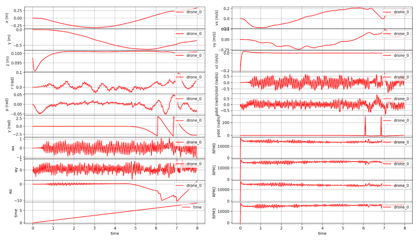


Figure 15: PID at 3.4m/s

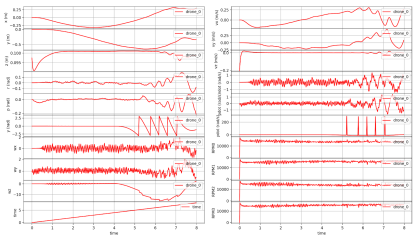


Figure 16: PID at 8.0m/s

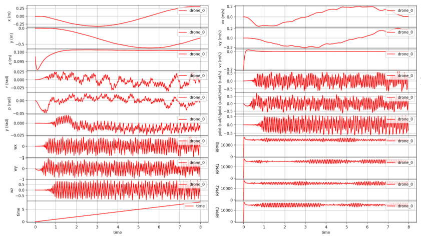


Figure 17: PPO at 0.3m/s

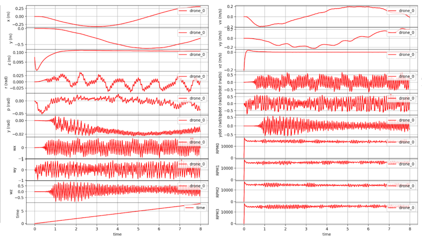


Figure 18: PPO at 3.4m/s

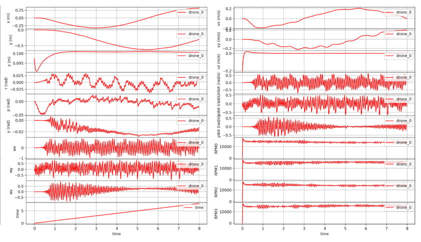


Figure 19: PPO at 8.0m/s

In simulations with progressively increasing wind speed, the PPO controller exhibited superior attitude stability. Even under strong winds equivalent to Beaufort scale 5 ($\approx$ 8.0 m/s), attitude deviations remained within $\pm 5^{\circ}$, whereas the PID controller's attitude excursions under the same conditions expanded to approximately ±15°. This indicates that PPO substantially improves disturbance rejection and reduces the accumulation of attitude error in adverse aerodynamic conditions.

Position-error measurements further confirm PPO's advantage. Across wind levels, the PPO-controlled vehicle maintained lower position deviations. For example, under level-3 winds ($\approx$ 3.4– 5.4 m/s), the mean position error with PPO was only 0.8m, compared with 1.5m for PID control. This improvement is attributable to PPO's capacity for timely corrective actions and more precise compensation for wind disturbances, enabling the vehicle to track the target position more closely during gusty flight.pics/ experiment_materials.png

4.3.2 - Physical Experimentation

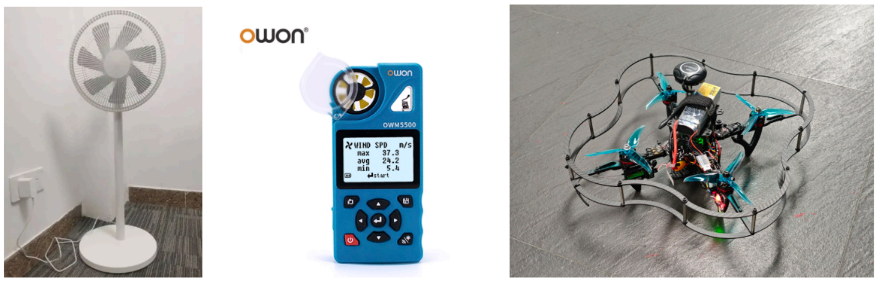


Figure 20: Materials

Figure 21: PID at Level 3 Wind

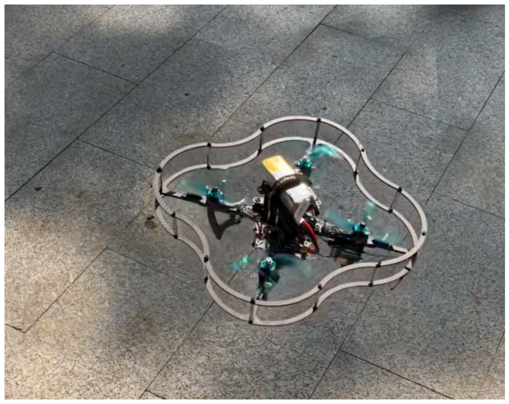


Figure 22: PPO at level 3 Wind

Due to environmental constraints and the power limitations of household fans, the UAV state under level-3 wind conditions.

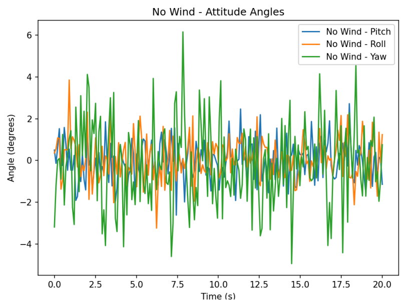


Figure 23: Attitude Angle of PID at Level 0 Wind

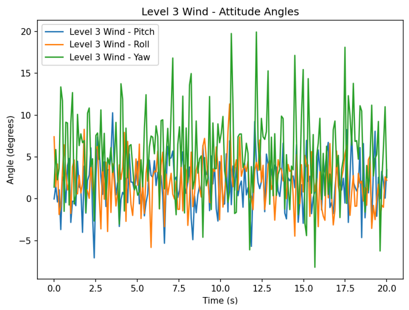


Figure 24: Attitude Angle of PID at Level 3 Wind

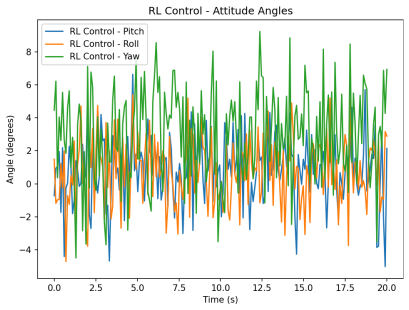


Figure 25: Attitude Angle of PPO at level 3 Wind

Under no-wind conditions, the drone's horizontal displacements in both the X and Y axes were tightly regulated, with mean values effectively zero and a standard deviation of approximately 0.10m. Altitude was held stably at the 1.0m setpoint with only minor fluctuations (standard deviation $\approx$ 0.5m). Under natural level-3 winds, the horizontal displacement variability increased markedly: the standard deviation in X and Y rose to $\approx$ 0.3m, indicating degraded position stability; altitude remained centered on 1.0 m but with an increased standard deviation of $\approx$ 0.10 m. When controlled by the reinforcement-learning (PPO) controller, however, the drone's positional performance under level-3 winds improved substantially: horizontal displacement standard deviations were reduced to $\approx$ 0.15m in both X and Y, and altitude fluctuation was constrained to a standard deviation of $\approx$ 0.07m. These results demonstrate the superiority of the learned controller in maintaining precise position and altitude in moderately windy conditions, reinforcing its suitability for disturbance-robust drone operation.

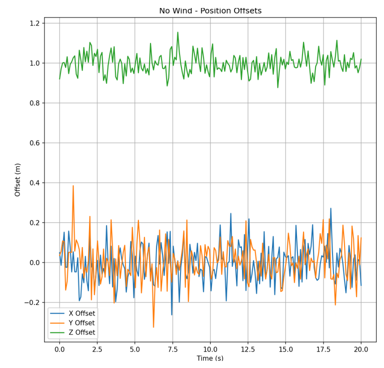


Figure 26: Positional Offset of PID at Level 0 Wind

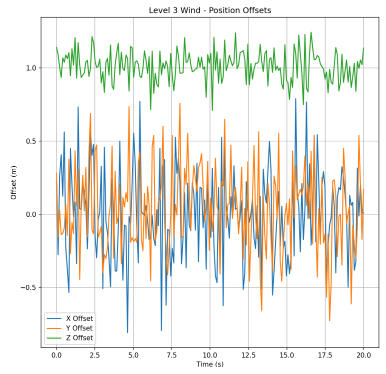


Figure 27: Positional Offset of PID at Level 3 Wind

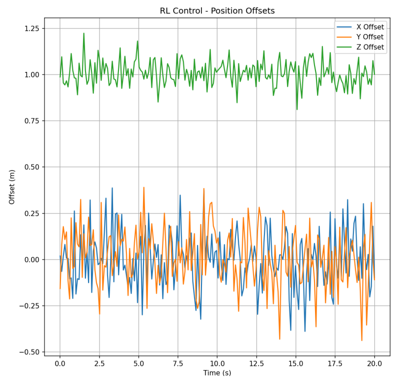


Figure 28: Positional Offset of PPO at level 3 Wind

With no wind, the drone's x and y displacements were tightly regulated, with mean values effectively 0 and a standard deviation of approximately 0.1m. Altitude was held stably at the 1.0m setpoint with minor fluctuations (standard deviation ≈ 0.5m). When exposed to natural level-3 wind, horizontal variability increased. The standard deviation in x and y rose to around 0.3m, indicating degraded lateral position stability; altitude remained centered on 1.0m, but with a increased standard deviation of about 0.1m. By contrast, the reinforcement-learning controller substantially mitigated these disturbances. Under the same level-3 wind, x and y standard deviation were reduced to approximately 0.15m, and altitude variability was constrained to a standard deviation of 0.07m. These results highlight the learned controller's superior ability to provide precise positional regulation in moderately windy conditions and demonstrate its potential to improve disturbance-robust operation.

5 - Conclusion and Outlook

5.1 - Summary of Research Results

This study compared a learned controller (PPO) against a conventional cascaded PID baseline for attitude stabilization and position holding on a small quadrotor platform (CQ230-style hardware in simulation). Key findings are:

- Attitude stability under wind. In progressively stronger simulated winds PPO maintained attitude excursions within $\pm 5^{\circ}$ at ≈ 8.0m/s, whereas the tuned PID's excursions grew to roughly $\pm 15^{\circ}$ under the same conditions.
- Position accuracy. Under level-3 winds ($\approx 3.4 - 5.4$m/s) the PPO policy produced a mean position error of ~ 0.8m versus ~ 1.5 m for PID; horizontal displacement standard deviations improved from ~ 0.30 m under PID to ~ 0.15m under PPO. Altitude variability also reduced from ≈ 0.10m with PID to $\approx$ 0.07m with PPO.
- Practical observation. PPO's advantage stems from its ability to (1) implicitly learn nonlinear compensation for wind and actuator effects, (2) optimize multi-objective trade-offs encoded in the reward, and (3) generalize across a range of simulated disturbances when trained with sufficient scenario variety.

These results demonstrate that reinforcement learning, when carefully trained, can improve robustness and stability for small consumer drones relative to traditional control tuned around a single operating point.

5.2 - Deficiencies and Future Work

All experiments used simulated wind (PyBullet) and household-fan test with limited uniformity and power. The learned policy may not directly transfer to physical hardware without domain randomization, system identification, or real-world fine tuning. Also, the simulator and the household wind setup do not fully reproduce turbulent, spatially varying wind fields, ground effect, or detailed propeller aerodynamics. These unmodeled effects can degrade real-world performances of PPO.

Also, due to using Reinforcement Learning Algorithms, important safety behaviors require additional verification, fallback controllers (which still have to be tuned), or runtime monitors. Also, training required many interactions: on-board learning or frequent re-training is impractical unless sample efficiency improves or efficient online adaptation schemes are implemented. Additionally, deploying neural network policies onboard requires attention to latency, quantization, and computational power on embedded platforms. Performance depends strongly on reward engineering. Learned behaviors can be brittle if reward terms are misaligned. Interpreting why a learned policy behaves a certain way also remains difficult. Experiments covered a finite set of wind magnitudes and conditions. Generalization to all other payloads, motor faults, and extreme maneuvers was not evaluated, and therefore lead to degraded performances in those scenarios.

Therefore, some suggested next steps includes:

- Add domain randomization (mass, sensor noise, delay, wind patterns) and system identification to narrow simulation-to-real gap.
- Explore more hybrid architectures (such as PPO with PID/LQR as fallback) in the case of PPO failure.
- Investigate explainability tools and systematic ablation studies to better understand reward-term contributions.

References

[1] T. D. Company, “No Fear of Storms: New DJI M30 Enterprise Can Operate in Heavy Weather [Image].” 2025.

[2] P. Gui, L. Tang, and S. C. Mukhopadhyay, “MEMS Based IMU for Tilting Measurement: Comparison of Complementary and Kalman Filter Based Data Fusion,” in *Proceedings of the 2015 IEEE 10th Conference on Industrial Electronics and Applications (ICIEA)*, Auckland, New Zealand, 2015, pp. 2004–2009. doi: 10.1109/ICIEA.2015.7334442.

[3] S. Bouabdallah, A. Noth, and R. Siegwart, “PID vs LQ Control Techniques Applied to an Indoor Micro Quadrotor,” in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2004. doi: 10.3929/ethz-a-010085491.

[4] M. Okasha, J. Kralev, and M. Islam, “Design and Experimental Comparison of PID, LQR and MPC Stabilizing Controllers for Parrot Mambo Mini-Drone,” *Aerospace*, vol. 9, no. 6, p. 298, 2022, doi: 10.3390/aerospace9060298.

[5] J. Peksa and D. Mamchur, “A Review on the State of the Art in Copter Drones and Flight Control Systems,” *Sensors*, vol. 24, no. 11, p. 3349, 2024, doi: 10.3390/s24113349.

[6] A. Zulu and S. John, “A Review of Control Algorithms for Autonomous Quadrotors,” *Open Journal of Applied Sciences*, vol. 4, no. 14, pp. 547–556, 2014, doi: 10.4236/ojapps.2014.414053.

[7] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and OpenAI, “Proximal Policy Optimization Algorithms,” 2017.

[8] J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel, “Trust Region Policy Optimization,” *arXiv preprint arXiv:1502.05477*, 2015, [Online]. Available: https://arxiv.org/abs/1502.05477

[9] W. Chen, K. K. L. Wong, S. Long, and Z. Sun, “Relative Entropy of Correct Proximal Policy Optimization Algorithms with Modified Penalty Factor in Complex Environment,” *Entropy*, vol. 24, no. 4, p. 440, 2022, doi: 10.3390/e24040440.

[10] P.-J. Bristeau, F. Callou, D. Vissière, and N. Petit, “The Navigation and Control Technology Inside the AR.Drone Micro UAV,” in *Preprints of the 18th IFAC World Congress*, Milano, Italy, 2011, pp. 1477–1484. [Online]. Available: https://www.asprom.com/drone/PJB.pdf

[11] O. A. Dhewa, F. Arifin, A. S. Priambodo, A. Winursito, and Y. M. Mustafah, “Attitude UAV Stability Control Using Linear Quadratic Regulator-Neural Network (LQR-NN),” *IIUM Engineering Journal*, vol. 25, no. 2, pp. 246–265, 2024, doi: 10.31436/iiumej.v25i2.3119.

[12] W. Koch, R. Mancuso, R. West, and A. Bestavros, “Reinforcement Learning for UAV Attitude Control,” *ACM Transactions on Cyber-Physical Systems*, vol. 3, no. 2, pp. 1–21, 2019, doi: 10.1145/3301273.

[13] F. Santoso, M. A. Garratt, and S. G. Anavatti, “State-of-the-Art Intelligent Flight Control Systems in Unmanned Aerial Vehicles,” *IEEE Transactions on Automation Science and Engineering*, vol. 15, no. 2, pp. 613–627, 2018, doi: 10.1109/TASE.2017.2651109.

[14] J. Panerati, H. Zheng, S. Zhou, J. Xu, A. Prorok, and A. P. Schoellig, “Learning to Fly—a Gym Environment with PyBullet Physics for Reinforcement Learning of Multi-agent Quadcopter Control,” in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2021, pp. 7512–7519. doi: 10.1109/IROS51168.2021.9635857.