Intelligent Control - Team Project
Learning Agile Maneuvers for a 3D Quadrotor

Members
Konstantinos Skantzis - up1092806@ac.upatras.gr
Georgios Tarnaras - up1092751@ac.upatras.gr

ECE UPatras - 2025

1 Introduction - Goals

Our goal for this project is to use **reinforcement learning** to train a simulated **quadrotor** to perform agile maneuvers such as **flips**, **dives**, or **slalom navigation**. Quadrotor helicopters are an emerging rotorcraft concept for unmanned aerial vehicle (UAV) platforms. The vehicle consists of four rotors in total, with two pairs of counter-rotating, fixed-pitch blades located at the four corners of the aircraft. Due to its specific capabilities, the use of autonomous quadrotor vehicles has been envisaged for a variety of applications both as individual vehicles and in multiple vehicle teams, including surveillance, search and rescue and mobile sensor networks.

Figure 1: A real quadrotor made by 3D Robotics Company

2 Quadrotor dynamics

2.1 Rigid body dynamics

The first step before control development is a good modeling of the dynamic system. For simplicity, we will ignore friction. The derivation of the equations of motion for a quadrotor requires two reference frames: the earth-fixed frame and the body-fixed frame (see Figure 1). The world-fixed frame consists of the axes X, Y and Z with Z pointing upward. The body-fixed frame is attached to the center of gravity of the quadrotor with x pointing to rotor 1, y pointing to rotor 4 and z perpendicular to the plane of rotors pointing upward in the hoverstate. Z–X–Y Euler angle sequence (ψ, ϕ, θ), referred to as yaw, roll, and pitch, respectively, is used to model the orientation of the quadrotor in the earth-fixed frame. Each of the four rotors generates a thrust F_i and a torque T_i, which are perpendicular to their plane. The Newton-Euler equations for linear and angular motion can be written as follows:

$$ma = \begin{bmatrix} 0 \\ 0 \\ mg \end{bmatrix} + R \begin{bmatrix} 0 \\ 0 \\ \sum F_i \end{bmatrix} \tag{1}$$

$$I\dot{\omega} = I \begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} = \begin{bmatrix} l(F_2 - F_4) \\ l(F_3 - F_1) \\ -T_1 + T_2 - T_3 + T_4 \end{bmatrix} + \begin{bmatrix} p \\ q \\ r \end{bmatrix} \times I \begin{bmatrix} p \\ q \\ r \end{bmatrix} \tag{2}$$

Where:

- m: mass of the quadrotor
- l: length between rotors and z-axis or length between rotors
- g: acceleration due to gravity
- a: acceleration of the quadrotor
- I: diag(I_{xx}, I_{yy}, I_{zz})
- R: Rotation matrix from the body-fixed frame to the earth-fixed frame
- p,q,r: The components of angular body rates these are related to the derivatives of Euler angles

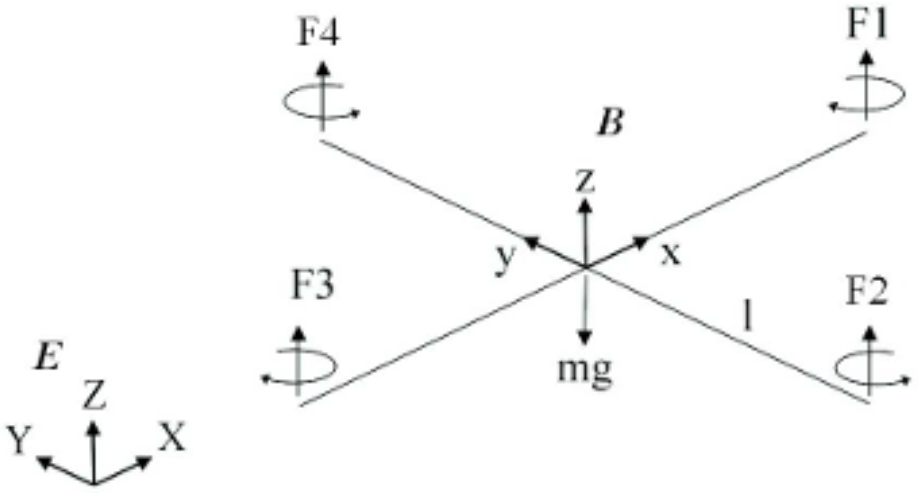

Figure 2: Frame system with a body fixed frame B and the inertial frame E.

Because the 4 propellers can be arranged in an X or + configuration, the torques along the x and y axes can be calculated in different ways. During simulation, we assumed the X configuration, instead of the + configuration shown in the equations, and l as the length between the rotors.

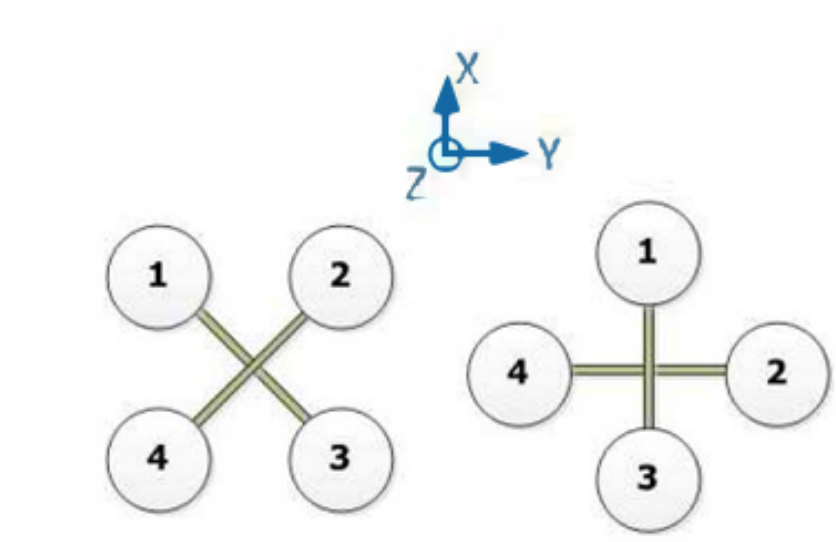

Figure 3: X and + configuration

2.2 Motor Configuration and Torque Calculation for X-Configuration

For the quadrotor in an X-configuration, the motors are arranged as follows:

- **Motor 1:** Front-Left (FL), Counter-Clockwise (CCW)
- **Motor 2:** Front-Right (FR), Clockwise (CW)
- **Motor 3:** Rear-Right (RR), Counter-Clockwise (CCW)

- **Motor 4:** Rear-Left (RL), Clockwise (CW)

The torques in the body frame are calculated using the standard quadrotor dynamics equations:
$\tau_x = \frac{L}{2}(F_{RL} + F_{FL} - F_{FR} - F_{RR})$

$\tau_y = \frac{L}{2}(F_{RL} + F_{RR} - F_{FR} - F_{FL})$

$\tau_z = k(F_{FL} + F_{RR} - F_{FR} - F_{RL})$
where L m is the length between the rotors and k is the torque-to-thrust ratio.

2.3 Quaternion Dynamics

To represent the quadrotor's orientation and avoid gimbal lock issues, we use unit quaternions for the slalom and dive tasks:

$$q = \begin{bmatrix} w \\ x \\ y \\ z \end{bmatrix}.$$

The time derivative of the quaternion is governed by the quaternion kinematic equation:

$$\dot{q} = \frac{1}{2} q \otimes \begin{bmatrix} 0 \\ \omega_x \\ \omega_y \\ \omega_z \end{bmatrix}, \tag{3}$$

where $\otimes$ denotes quaternion multiplication and

$$\boldsymbol{\omega} = \begin{bmatrix} \omega_x & \omega_y & \omega_z \end{bmatrix}^T$$

is the angular velocity vector expressed in the body-fixed frame. For the flip task we used euler angles.

2.4 Motor dynamics

Each motor with angular speed Ω_i, produces a thrust F_i and a torque T_i.

$$F_i = k_T \Omega_i^2 \tag{4}$$

$$T_i = k_Q \Omega_i^2 \tag{5}$$

where k_T and k_Q are the thrust and drag coefficient. For the purpose of this project we will assume the **thrusts** are our actions, which would be controlled directly by a controller changing the rotors speeds in a real environment.

2.5 Parameters used during simulation

The parameters used in our model correspond to real-world values and can accurately represent a quadrotor during simulation.

Parameter	Value
mass	0.5 kg
g	9.81 ms^{-2}
I_{xx}	0.0075 kgm^2
I_{yy}	0.0075 kgm^2
I_{zz}	0.013 kgm^2
k_Q/k_T	0.02 m
L	0.15 m

Table 1: Parameters used during our simulation in python

3 Reinforcement Learning

Reinforcement learning (RL) is a type of machine learning process that focuses on decision making by autonomous agents. An autonomous agent is any system that can make decisions and act in response to its environment independent of direct instruction by a human user. In reinforcement learning, an autonomous agent learns to perform a task by trial and error in the absence of any guidance from a human user. By carefully designing reward functions we can train our quadrotor to perform different tasks.

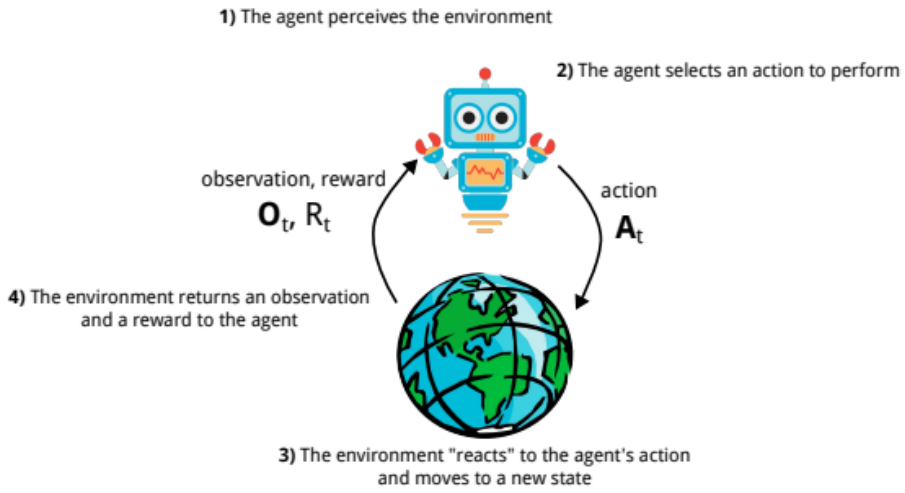

Figure 4: Environment Model

3.1 PPO

PPO is a popular policy gradient method in reinforcement learning designed to improve training stability and sample efficiency. It works by updating the policy in a way that prevents large, destructive updates, using a clipped objective. We will use PPO to train our quadrotor to perform each task. We chose to use PPO as our training algorithm because through trial and error we found it worked best for our tasks. More about PPO can be read here.

4 Simulation in Python

We will use OpenAI's Gymnasium to simulate the quadrotor environment we want to train our agent in. Gymnasium is an open source Python library for developing and comparing reinforcement learning algorithms by providing a standard API to communicate between learning algorithms and environments, as well as a standard set of environments compliant with that API. For our training we will use the PPO algorithm implemented in stable-baselines 3. Stable Baselines3 (SB3) provides us with a set of reliable implementations of reinforcement learning algorithms in PyTorch. The PPO agent was trained using a multilayer perceptron (MLP) policy network. The learning rate was set to 3×10^{-4}, and the rollout length (n_{steps}) was 2048 steps. Mini-batch updates were performed with a batch size of 64. A discount factor $\gamma = 0.99$ and a GAE $\lambda = 0.95$ were used to compute the advantage estimates. The entropy coefficient was set to 0.0, effectively disabling entropy regularization. Training logs were recorded via TensorBoard for monitoring purposes. In addition to these, we will use: numpy (for computations), pybullet (for visualization), matplotlib (for plotting). We start by creating a parent class called QuadRotor. QuadRotor will inherit from gym.Env which is the main Gymnasium class for implementing Reinforcement Learning Agents environments. We initialize our parameters in the __init__ method, as well as our action space

$$\text{action} = [F_{FR}, F_{RL}, F_{FL}, F_{RR}] \in \mathbb{R}^4$$

where the thrust range for each motor is $[0, 3 \times F_{hover}]$, with $F_{hover} = \frac{mg}{4}$ being the thrust required per motor for hovering.. In addition, we implement the methods used to update the dynamics during simulation _get_rotation_matrix, _euler_rates_to_body_rates, _body_rates_to_euler_rates, quaternion_multiply, quaternion_to_rotation_matrix. These methods will later be used by the subclasses in their step method to update the dynamics at each single timestep. As their names suggest these methods, in order: return the rotation matrix from the euler angles, convert euler angle rates to body angular velocities and convert body angular velocities to euler angle rates, perform quaternion multiplication and convert the quaternion to the rotation matrix. We need to follow Gym's expected API structure. This means we need to implement the methods: **reset(), step(), render() and methods for observation and info** in the subclasses used. Their implementations can be seen in the python files and are for the most part standard. For the purpose of this report we will focus on the steps we took to create the reward functions correctly.

4.1 Flip

QuadRotorFlip is the subclass that inherits from QuadRotor and implements a realistic environment for learning aerial flips. Our target is to make the quadrotor flip around the x axis, which means we want our roll angle to change from 0° to 360°. We initialize our state variables *state* = [position$\in \mathbf{R}^3$, velocity$\in \mathbf{R}^3$, euler angles $\in \mathbf{R}^3$, angular velocity$\in \mathbf{R}^3$, motor forces$\in \mathbf{R}^3$,angular velocity$\in \mathbf{R}^3$, roll ϕ, roll progress $\frac{\phi}{360°}$] and different lists which will store values for visualization purposes. To properly train our quadrotor to do flips we reward it when it makes progress towards its goal. We define progress as the ratio of the total roll angle and the target roll angle $\frac{\phi}{360°}$. We also reward it if maintains a good altitude. A good altitude is defined as close to 2 meters, it's starting position. Finally we give a big reward if it successfully completes a flip.

Flip Rewards	Definitions
Progress Reward	$100\sqrt{\frac{\phi}{360°}}$
Altitude Reward	5 if $\lvert z-2\rvert < 0.25$
Flip Reward	500 if $\phi > 360°$

Table 2: Positive Rewards for the Flip Task

The rewards constants were empirically selected by trial and error, based on the agent performance observed during training. The non linear progress reward encourages the agent to start moving in the right direction, but doesn't overly reward big leaps. We don't want our agent to move aggressively. Additionally, we apply penalties when our agent doesn't behave the way we want.

Flip Penalties	Definitions
Position Penalty	$-100\lvert z-2\rvert$
Velocity Penalty	$-0.5\lVert u_{xyz}\rVert$
Angular Velocity Penalty	$-2\lVert \omega_{yz}\rVert$
Efficiency Penalty	$-0.1Var(Forces)$
Orientation Penalty	$-10(\lvert\theta\rvert + \lvert\psi\rvert)$

Table 3: Penaltiy Terms for the Flip Task

We penalize our agent when it deviates from 2 meters and the center with a very large penalty. We also penalize large linear velocities and angular velocities along the axis we don't care about. We apply an efficiency penalty to encourage energy-efficient behavior and discourage erratic control inputs. Finally, to maintain flight stability and reduce oscillations or erratic maneuvers, we applied a penalty based on the absolute pitch and yaw angles. The episode may terminate if the agent crashes, reaches a max height, develops excessive velocities, or succeeds in doing its task. We define the max steps per episode as 400, with a dt of 0.05 secs, and train our agent for a total of 5 million timesteps. Our pitch and yaw angles stay almost constant and near zero, while our roll angle reaches 360°. The quadrotor maintains a good height and doesn't fall close to the ground.

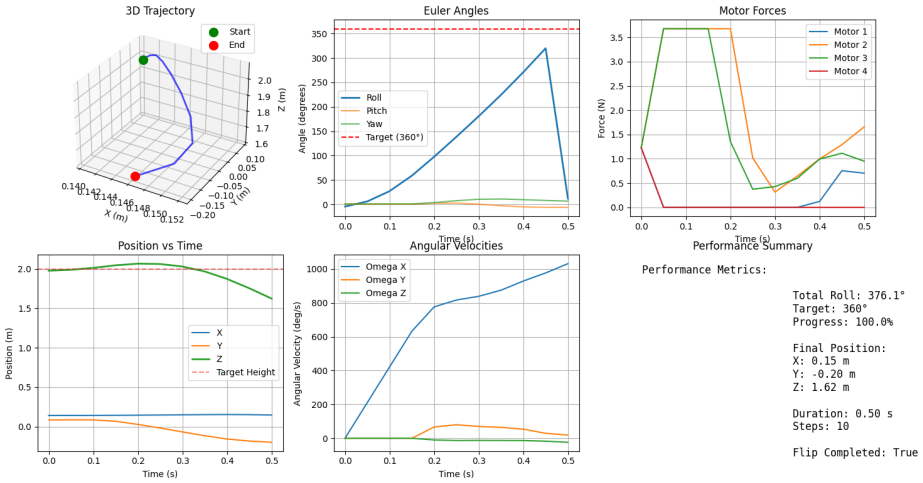

Figure 5: Flip results after training for 5 million timesteps

We train our model for an additional 5 million steps. After 10 million steps we see

that the quadrotor succesfully performs a flip while maintaing a Z position closer to its initial this time.

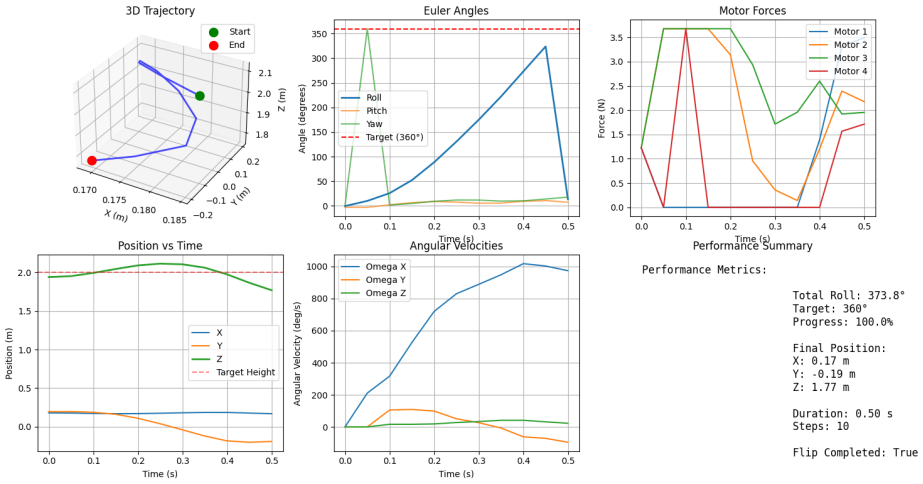

Figure 6: Flip results after training for 10 million timesteps

4.2 Dive

For the dive maneuver, we create a QuadRotorDive subclass that implements a challenging diving task from high altitude to a specific target zone. The objective is to train the quadrotor to descend from an initial altitude of approximately 20 meters to a target altitude of 1 meter while maintaining positional accuracy and stability. Our state space for the dive task is expanded to include relative positioning information: $state = [\text{position} \in \mathbf{R}^3, \text{velocity} \in \mathbf{R}^3, \text{quaternion} \in \mathbf{R}^4, \text{angular velocity} \in \mathbf{R}^3, \text{relative position to target} \in \mathbf{R}^3]$. The **Dive Environment** implements several key features designed to simulate realistic agile maneuvers:

- **Initial Conditions:** The quadrotor starts at an altitude $z_0 \sim \mathcal{N}(20, 3^2)$ meters, with a small random displacement in the x and y directions.
- **Target Zone:** The dive target is located at coordinates:

$$(x_t, y_t, z_t) = (0, 0, 1) \text{ meters.}$$

- **Success Criteria:** A dive is considered successful if the following conditions are simultaneously satisfied:

$$\begin{aligned} |z - z_t| &< 0.5 \text{ m} \quad \text{(Altitude error)} \\ \sqrt{(x - x_t)^2 + (y - y_t)^2} &< 1.5 \text{ m} \quad \text{(Horizontal error)} \\ |v_z| &< 1.0 \text{ m/s} \quad \text{(Vertical velocity)} \end{aligned}$$

The reward structure for the dive task is carefully designed to encourage a controlled descent while maintaining stability and accuracy.

Here:

- $d_z = |z - z_t|$ denotes the altitude error,

Dive Rewards	Definitions
Altitude Progress	$1.5 \times (d_{z,prev} - d_{z,current})$
Completion Bonus	+250 points upon successful dive completion

Table 4: Positive Rewards for the Dive Task

Dive Penalties	Definitions
Horizontal Drift	$-0.15 \times \|\|(x, y) - (x_t, y_t)\|\|$
Angular Velocity Magnitude	$-0.01 \times \|\|\boldsymbol{\omega}\|\|$
Excessive Tilt	$-10 \times \big(\cos\theta_{\max} - \cos\theta\big)$, if $\theta > \theta_{\max}$
Inverted Flight	-150, if $\cos\theta < 0$
Ground Collision	-150, if $z < 0.5$ m
Time Penalty	-0.1 per timestep

Table 5: Penalty Terms for the Dive Task

- $\theta_{\max} = 60°$ is the maximum allowable tilt angle,
- θ is the current tilt angle relative to vertical.

The dive task training uses a maximum episode length of 600 steps (corresponding to 30 seconds with a timestep $dt = 0.05$ s) and a total of 4 million training timesteps.

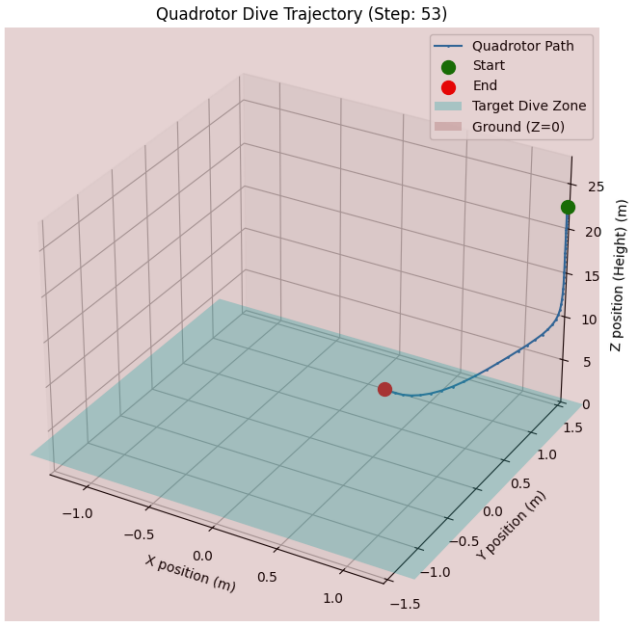

Figure 7: Dive trajectory after training for 4 million timesteps

4.3 Slalom

For the slalom maneuver, we create a QuadRotorSlalom subclass that implements a challenging slalom navigation task. The objective is to train the quadrotor to navigate

through a series of gates arranged in a slalom pattern, requiring precise control and agile maneuvering capabilities. The slalom environment is designed with the following key characteristics:

- **Gate Configuration:** The course consists of 5 gates positioned alternately to the left and right, creating a zigzag pattern
- **Gate Spacing:** Gates are spaced 15 meters apart in the Z-direction (forward direction)
- **Gate Size:** Each gate has an opening of 2.0 × 2.0 meters
- **Course Width:** The slalom pattern spans 10 meters in the X-direction
- **Gate Positioning:** Gates alternate between $x = +5$ meters and $x = -5$ meters

Our state space for the slalom task includes comprehensive information about the quadrotor's state and its relationship to the target gate: $state$ = [position$\in \mathbf{R}^3$, velocity$\in \mathbf{R}^3$, quaternion $\in \mathbf{R}^4$, angular velocity$\in \mathbf{R}^3$, relative position to next gate$\in \mathbf{R}^3$]. The slalom task implements several critical features:

- **Gate Detection:** The system tracks which gate is the current target and detects when the quadrotor passes through a gate by monitoring Z-plane crossings
- **Collision Detection:** The environment detects collisions with gate structures and ground impacts
- **Boundary Enforcement:** Flight boundaries prevent the quadrotor from flying too far outside the course area
- **Progressive Difficulty:** The alternating gate positions require rapid direction changes and precise control

The reward function for the slalom task is designed to encourage both progress through the course and flight stability:

Slalom Rewards	**Definitions**
Approach Reward	$1.0 \times (d_{prev} - d_{current})$
Gate Passage Bonus	+100 points for successfully passing through a gate
Course Completion Bonus	+200 points for completing all gates

Table 6: Positive Rewards for the Slalom Task

Slalom Penalties	**Definitions**
Distance Penalty	$-0.1\times$ distance to target gate
Gate Miss Penalty	-50 points for crossing gate plane but missing opening
Collision Penalties	-100 points for ground or structure collision
Boundary Violation	-100 points for flying out of bounds
Crash Penalty	-50 points for any terminating event

Table 7: Penalty Terms for the Slalom Task

We define the max steps per episode as 400, with a dt of 0.05 secs, and train our agent for a total of 5 million timesteps. The slalom maneuver represents the most complex of the three tasks, as it requires the quadrotor to demonstrate sustained agile flight with rapid direction changes, precise positioning, and obstacle avoidance. The alternating gate positions force the agent to learn dynamic maneuvering strategies while maintaining flight stability throughout the course. Our quadrotor succesfully passes through each gate as shown in the figure below!

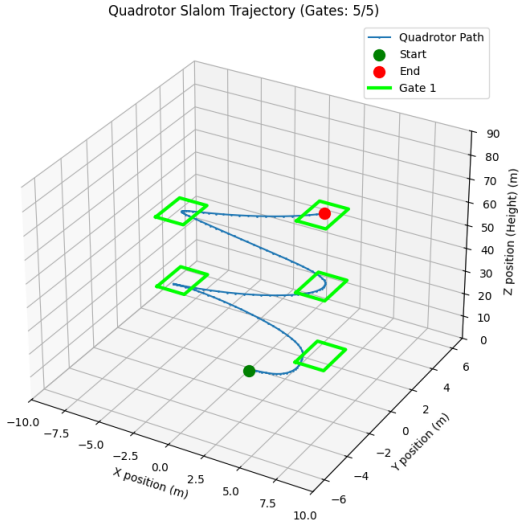

Figure 8: Slalom navigation after training for 5 million timesteps

5 Python Files - Requirements

To run the code Python version 3.8 or higher is required. The libraries used can be installed with pip in terminal.

```
pip install gymnasium numpy matplotlib stable-baselines3[extra]
   pybullet tensorboard torch
```

Listing 1: python libraries

The file **quadrotor.py** containts the class QuadRotor which is inherited by the sub-classed used for training.The files **quadrotor_flip.py, quadrotor_dive.py, quadrotor_slalom.py** are the files used for training. To train the agent from the start set TRAIN_MODEL to True and change the name of the SAVE_MODEL_PATH in each file. The rest of the files are used for visualition. We have provided zip files with trained model's for each task. These could also be loaded and used to continue training without starting again from scratch.

6 Conclusion

This project successfully demonstrates the application of reinforcement learning to train a quadrotor to perform three distinct agile maneuvers: flips, dives, and slalom navigation. Through careful environment design, reward function engineering, and the implementation of realistic quadrotor physics, we achieved successful training of autonomous flight behaviors using the PPO algorithm. The results show that reinforcement learning can effectively learn complex flight maneuvers when provided with appropriate reward structures and sufficient training time. Each task required specific considerations in terms of state representation, reward design, and safety constraints, highlighting the importance of domain knowledge in RL applications for autonomous systems.Future work could explore more complex maneuvers, multi-objective optimization, and transfer learning between different flight tasks to create more versatile autonomous flight controllers.

References

[1] Mizouri Walid, Najar Slaheddine, Aoun Mohamed, Bouabdallah Lamjed (2014) *Modeling and control of a quadrotor UAV*. In *2014 15th International Conference on Sciences and Techniques of Automatic Control and Computer Engineering (STA)*. IEEE

[2] Maziar Mardan, Masoumeh Esfandiari and Nariman Sepehri (2017). *Attitude and position controller design and implementation for a quadrotor*. In *International Journal of Advanced Robotic Systems*. SAGE

[3] Bouabdallah Samir, Murrieri Pierpaolo, Siegwart Roland (2004). *Design and Control of an Indoor Micro Quadrotor*. In *IEEE International Conference on Robotics and Automation (ICRA 2004), New Orleans, LA, USA, 26 April-1 May, 2004*. ETH-Zürich.

[4] Beomyeol Yu, Taeyoung Lee (2023). *Equivariant Reinforcement Learning for Quadrotor UAV*. In *2023 American Control Conference (ACC)*.

[5] Gabriel M. Hoffmann, Haomiao Huang, Steven L. Waslander, Claire J. Tomlin. (2007) *Quadrotor Helicopter Flight Dynamics and Control: Theory and Experiment*. In *AIAA Guidance, Navigation and Control Conference and Exhibit20 - 23 August 2007, Hilton Head, South Carolina*. AIAA.

[6] Jacob Murel, Eda Kavlakoglu, *What is reinforcement learning?* Available: https://www.ibm.com/think/topics/reinforcement-learning

[7] Konstantinos Chatzilygeroudis (2025) *Intelligent Control Lectures*