

# Reinforcement Learning Project

UAV Control via Deep Reinforcement Learning

PyFlyt Simulator

---

<b>Course</b>	INFO8003 – Reinforcement Learning
<b>Groups</b>	2 students
<b>Simulator</b>	PyFlyt v0.29+ [1]

---

*Train, evaluate, and compare deep RL agents for drone control.  
Compete in a dogfight tournament.*

# Contents

<b>1</b>	<b>Environments &amp; MDP Formalization</b>	<b>[3/20]</b>	<b>2</b>
1.1	Flight Modes . . . . .		2
1.2	QuadX-Hover-v4 (Stabilization) . . . . .		2
1.3	QuadX-Waypoints-v4 (Navigation) . . . . .		2
1.4	Dogfight – MAFixedwingDogfightEnvV2 . . . . .		3
<b>2</b>	<b>Algorithms</b>		<b>3</b>
<b>3</b>	<b>Experiments &amp; Results</b>	<b>[4/20]</b>	<b>3</b>
<b>4</b>	<b>Analysis &amp; Discussion</b>	<b>[5/20]</b>	<b>3</b>
<b>5</b>	<b>Implementation</b>	<b>[4/20]</b>	<b>3</b>
<b>6</b>	<b>Dogfight Tournament</b>	<b>[2/20]</b>	<b>4</b>
6.1	Rules . . . . .		4
6.2	Submission Format . . . . .		4
6.3	Evaluation Environment . . . . .		4
6.4	Scoring . . . . .		4
6.5	Leaderboard . . . . .		4
<b>7</b>	<b>Bonus / Innovation</b>	<b>[2/20]</b>	<b>4</b>
<b>8</b>	<b>Deliverables</b>		<b>4</b>
<b>9</b>	<b>Provided Code</b>		<b>5</b>

## Use of LLMs

You may use large language models (ChatGPT, Claude, Copilot, etc.) to help write code. However:

- You **must understand every line of code** you submit.
- **Questions will be asked** at the oral presentation and the exam. You must be able to explain your implementation, your training pipeline, and the algorithms you used.
- If you cannot explain your code, it will be considered as not done.

## 1 Environments & MDP Formalization

[3/20]

All environments use PyFlyt [1], a Gymnasium-compatible UAV simulator built on PyBullet. They follow the standard Gymnasium API:

```
import gymnasium
env = gymnasium.make("PyFlyt/QuadX-Hover-v4", flight_mode=0)
obs, info = env.reset(seed=42)          # obs: np.ndarray
action = env.action_space.sample()      # action: np.ndarray(4,) in [-1,
1]
obs, reward, terminated, truncated, info = env.step(action)
env.close()
```

Any RL library that works with Gymnasium can be used. The 4D continuous action space is shared across all environments; its meaning depends on the `flight_mode`.

In your report, **formalize each environment as an MDP**. Explore the environments in code to understand what the observations and rewards look like.

### 1.1 Flight Modes

Mode	Action semantics	Description	Difficulty
-1	$(m_1, m_2, m_3, m_4)$	Direct motor PWM	Hardest
0	$(\dot{p}, \dot{q}, \dot{r}, T)$	Angular velocity + thrust	Hard
4	$(u, v, \dot{r}, z)$	Body-frame velocity + yaw rate + height	Medium
6	$(\dot{x}, \dot{y}, \dot{r}, \dot{z})$	Ground velocity + yaw rate + vertical vel.	Easy
7	$(x, y, r, z)$	Position + yaw + height (full PID)	Easiest

Table 1: Selected flight modes. Higher modes abstract away low-level control via cascaded PID controllers.

### 1.2 QuadX-Hover-v4 (Stabilization)

Maintain a quadrotor at a fixed target position with stable orientation.

### 1.3 QuadX-Waypoints-v4 (Navigation)

Navigate through randomly placed 3D waypoints. The default PyFlyt parameters are very restrictive. The provided code overrides them in `env_config.py`. Read and understand those overrides.

```
env = gymnasium.make("PyFlyt/QuadX-Waypoints-v4", flight_mode=6,
goal_reach_distance=4.0, flight_dome_size=150.0,
max_duration_seconds=120.0, num_targets=4)
```

This environment returns a `Dict` observation space. Most RL libraries need a flat vector. The provided `FlattenWaypointEnv` wrapper handles this.

## 1.4 Dogfight – `MAFixedwingDogfightEnvV2`

1v1 fixed-wing aerial combat. This is a multi-agent environment (`PettingZoo`). The provided `DogfightSelfPlayEnv` wrapper turns it into a single-agent `Gymnasium` environment for training.

## 2 Algorithms

You may use **any RL algorithm** and **any library**. You must train and compare **at least two algorithms**.

In your report, describe each algorithm: its main idea, key properties (on-policy vs off-policy, exploration mechanism, etc.), and why you chose it. Explain things in your own words.

Your trained model must expose the following interface (used by the evaluation scripts and the tournament):

```
action, info = model.predict(obs, deterministic=True)
# obs:      np.ndarray    -- observation from the environment
# action:   np.ndarray    -- action to take (shape depends on env)
# info:     any           -- optional extra info (can be {})
```

## 3 Experiments & Results

[4/20]

I encourage to log your runs to [Weights & Biases](#): episode reward, evaluation reward (deterministic policy), and policy videos.

Use appropriate statistical tools to compare your algorithms. Do not draw big conclusions on a single run of training. The quality of your evaluation matters.

*Hint:* Agarwal et al. [2] discuss best practices for statistical evaluation in deep RL. Worth reading.

## 4 Analysis & Discussion

[5/20]

This section carries the most weight. We care more about your understanding than about high scores. Your report should cover:

- Why does one algorithm work better than the other on a given environment?
- Effect of flight mode on waypoint performance: why is mode 0 harder than mode 6?
- Failure analysis: what causes crashes, reward plateaus, or policy collapse?
- What would you do differently with more compute?

## 5 Implementation

[4/20]

- Clean, readable code with `requirements.txt` and `README.md`.
- Proper use of `Weights & Biases` for experiment tracking.
- Reproducible results: seeded experiments, documented hyperparameters.
- Any RL library is fine, implementing your own algorithm or exploring an algorithm proposed in papers is encouraged and rewarded.

## 6 Dogfight Tournament

[2/20]

### 6.1 Rules

Each group may submit **as many relevant agents as they want**. Test them before submitting. Motivate how you want to obtain the best agent and what you have done.

The tournament environment is fixed:

```
MAFixedwingDogfightEnvV2(  
    team_size=1, assisted_flight=True,  
    flatten_observation=True, max_duration_seconds=60, agent_hz=30,  
)
```

### 6.2 Submission Format

Submit Python files named `groupXX_name.py` (e.g., `group03_aggressive.py`) with a `load_model()` function. The only requirement is the `predict()` interface. See `scripts/submission_template.py`.

### 6.3 Evaluation Environment

The evaluation server loads your submission inside a pinned Python 3.10 venv whose exact dependency versions are listed in `requirements.txt` at the root of the repository (`numpy`, `torch`, `stable-baselines3`, `gymnasium`, `pyflyt`, ...). Pickled checkpoints embed version-specific module paths and Python code objects; a model saved under a different numpy or Python version may fail to load at grading time.

You do not need to train inside this environment. However, before submitting, **re-export your final model from a venv matching requirements.txt**. Loading and re-saving is sufficient; no retraining required:

```
from stable_baselines3 import PPO # or SAC, etc.  
PPO.load("your_model.zip").save("your_model_pinned.zip")
```

Submissions that fail to load will be flagged and you will be asked to re-export. This is a compatibility check, not a correctness check.

### 6.4 Scoring

Agents play a round-robin tournament, Elo-rated. Ranking bonus: top-3 (+1pt), top-5 (+0.5pt), based on your best agent only.

### 6.5 Leaderboard

A live leaderboard is available at [arthur-louette.com/leaderboard](https://arthur-louette.com/leaderboard). You can submit agents and track your ranking before the final deadline.

## 7 Bonus / Innovation

[2/20]

Extra credit for going beyond the required experiments: hyperparameter ablation, reward shaping, curriculum learning, architecture exploration, additional algorithms, etc.

## 8 Deliverables

You are expected to submit a complete zipped file with the following on the Montefiore submission platform before the **12th May**.

1. **Code repository** (Git): training scripts, wrappers, analysis. Include `requirements.txt` and `README.md`.
2. **Report** (PDF): MDP formalization, algorithm description, experimental setup, results with statistical analysis, flight mode comparison, dogfight strategy, discussion, conclusion.
3. **Wandb project link (optional but recommended)**: public or shared.
4. **Trained models**: best checkpoints per (algorithm, environment) pair.
5. **Tournament submissions**: `groupXX_name.py` files.

## 9 Provided Code

The repository contains environment configuration and evaluation scripts.

Script	Purpose
<code>scripts/env_config.py</code>	Environment parameters (waypoint overrides)
<code>scripts/wrappers.py</code>	<code>FlattenWaypointEnv</code> : flattens dict observations to a fixed-size vector
<code>scripts/dogfight_wrapper.py</code>	<code>DogfightSelfPlayEnv</code> : wraps multi-agent dogfight as single-agent Gymnasium env
<code>scripts/evaluate.py</code>	Evaluate any model ( <code>.py</code> module or <code>.zip</code> checkpoint)
<code>scripts/tournament.py</code>	Dogfight tournament (used for grading)
<code>scripts/submission_template.py</code>	Tournament submission template

`evaluate.py` and `tournament.py` are what we use to grade your work. Make sure your models run correctly with them before submitting.

## References

- [1] Tai, J. J. et al. (2023). *PyFlyt – UAV Simulation Environments for Reinforcement Learning Research*. arXiv:2304.01305.
- [2] Agarwal, R. et al. (2021). *Deep Reinforcement Learning at the Edge of the Statistical Precipice*. NeurIPS 2021.