# Fledgling: Teaching a Flapping Bird How to Flap Using PPO, SAC and TD3

Mostafa Ibrahim
Northeastern University

## 1 Introduction and Motivation

Biomimicry in robotics has led to various forms of locomotion, with flapping-wing flight being one of the most intriguing. While the hardware of bioinspired flapping robots defines their flapping frequency and gait, these systems often lack the ability to adaptively control these motions without manual input or complex controllers. Traditional control methods rely on motion capture or visual-inertial odometry, making them computationally heavy and environment-dependent.

Inspired by how birds learn to fly, this project explores a reinforcement learning (RL) approach where a simulated flapping bird learns to flap autonomously in an environment mimicking the aerodynamics of the real-world to achieve specific goals. Without predefined controllers, the agent discovers that flapping enables movement and learns optimal gaits to maximize forward distance. This data-driven strategy simplifies control, allowing for adaptive and goal-driven behavior in flapping-wing robots. Given an objective, the robot will learn to optimize the method to achieve it. The methods I utilized to train the robot where Twin Delayed Deep Deterministic Policy Gradient (TD3), Proximal Policy Optimization (PPO), Soft Actor-Critic (SAC). The robot evnetually learned how to flap in the correct flapping rate with differences in each method. Not to mention that hyperparameters significantly influence the learning of the robot.

## 2 Problem Statement

The goal of this project is to learn a control policy that enables a simulated flapping-wing bird, modeled in MuJoCo, to achieve forward flight by learning to generate appropriate wing motions. The bird must learn to sustain altitude and maximize horizontal travel using flapping alone, without relying on any predefined gait or controller.

At each timestep, the agent observes a 6-dimensional continuous state vector:

$$\mathbf{s}_t = \begin{bmatrix} x_t & z_t & \theta_t & \dot{x}_t & \dot{z}_t & \dot{\theta}_t \end{bmatrix}$$

Where:

- $x_t$: Horizontal position (forward progress)
- $z_t$: Vertical position (height)
- $\theta_t$: Pitch angle of the body
- $\dot{x}_t$: Horizontal velocity
- $\dot{z}_t$: Vertical velocity
- $\dot{\theta}_t$: Pitch rate (angular velocity)

Although the MuJoCo simulator provides full access to the system's degrees of freedom (DOFs), including orientation quaternions, lateral motion, and joint-specific information, we restrict the state space to the six features above. These features are chosen to focus the learning problem on essential flight dynamics while reducing dimensionality and computational complexity.

The agent outputs a continuous action vector $\mathbf{a}_t \in \mathbb{R}^3$, representing the control angles applied to the left and right wing flapping joints and the tail. The objective is to learn a stochastic or deterministic policy $\pi(a_t \mid s_t)$ that maximizes cumulative reward over time, where the reward is based on both forward distance and stable flight.

In summary, the problem is to learn a control policy that maps:

$$\pi : \mathbb{R}^6 \to \mathbb{R}^3$$

such that the bird flies as far and as stably as possible using learned flapping behavior, without falling or flipping over.

The optimal flapping gait would follow the following sinusoidal pattern:

$$\sin(2\pi\omega t)$$

Where $\omega$ is the flapping frequency and t is the timestep in the simulation.
**Note:** There are 5 possible actions: wing angles and pitches and tail angle. However, in this project, I taught the robot to only control the wings and the tail angles with no regard to the pitch

# 3 Environment Description

To simulate flapping-wing locomotion in a physically grounded setting, we developed a custom environment using the MuJoCo physics engine [1], integrated into a `gymnasium`-compatible API. The environment models a simplified bird-like agent equipped with two flapping wings and a tail, and is initialized in a free-fall configuration above a flat plane.

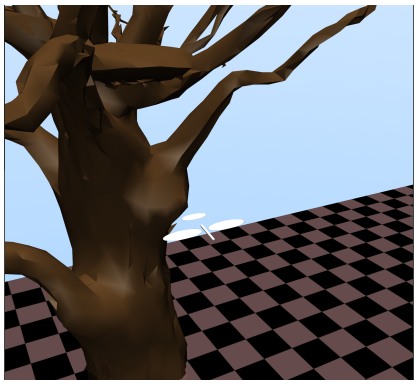

Figure 1: Flapping Bird Environment

## 3.1 Simulation Model

The underlying model is defined in an XML file of MuJoCo Model. The agent consists of a central body with two articulated wings, each controlled via a pair of actuated joints: one for flapping (vertical motion) and one for wing pitch. A tail is attached via a single joint that allows pitch adjustment. The environment includes aerodynamic forces defined via the `fluidshape` and `fluidcoef` parameters on the wing and tail geometries, enabling realistic lift and drag effects during motion [2].

The MuJoCo model uses:

- **Gravity:** $[0, 0, -9.81]$ m/s$^2$
- **Integrator:** Implicit Euler
- **Actuators:** Position and velocity actuators for wings and tail
- **Sensors:** Linear/angular velocity and orientation from a central site on the body

## 3.2 Action Space

The action space is a 2-dimensional continuous vector:

$$\mathbf{a}_t = \begin{bmatrix} a_{\text{left}} & a_{\text{right}} \end{bmatrix} \in \mathbb{R}^2, \quad a_i \in [-1.5, 1.5]$$

Each component represents the torque applied to the left or right wing flapping joint. For simplicity, pitch control and tail actuation were disabled after trying training with it for multiple times, although the MuJoCo model supports them.

## 3.3 Observation Space

The observation space is a 6-dimensional continuous vector:

$$\mathbf{s}_t = \begin{bmatrix} x & z & \theta & \dot{x} & \dot{z} & \dot{\theta} \end{bmatrix} \in \mathbb{R}^6$$

Where:

- $x$: Horizontal position
- $z$: Vertical height
- $\theta$: Pitch angle (from quaternion)
- $\dot{x}$: Forward velocity
- $\dot{z}$: Vertical velocity
- $\dot{\theta}$: Angular velocity

This representation is derived from a subset of MuJoCo's full state vector, which includes all DOFs and joint positions. I deliberately omit lateral motion, joint angles, and full orientation (quaternions) to focus the learning on global flight dynamics, rather than low-level body articulation.

## 3.4 Reward Function

To come up with the best reward function, I went through a range of complexities in reward functions. From very simple to very complex reward functions, each has performed differently and surprisingly the simpler reward function yielded the best results in all models.

### 3.4.1 Simple Reward Function: Simple Forward Distance

A simpler variant of the reward function was also tested. This function rewards the agent solely based on how far it has moved along the x-axis from its starting position:

$$\text{reward} = \exp(0.1 \cdot (x - x_{\text{start}})) + 0.1$$

Here, $x_{\text{start}} = -8.0$, and the reward increases exponentially with distance traveled forward. This design encourages fast movement but does not penalize unstable orientation or low altitude, which may hypothetically lead to unrealistic behaviors such as flipping or crashing. Nevertheless, the robot learned to flap stably and for a long distance.

The reward function encourages forward motion and stable flight. At each timestep, the agent receives a reward based on its progress along the $x$-axis

This design incentivizes the bird to travel farther from its initial position while staying airborne. The episode terminates early if the bird's height drops below a minimum threshold ($z < 2.0$).

### 3.4.2 Complex Reward Function

The reward function $r$ used to train the bird agent encourages controlled flapping and stable forward flight. It consists of multiple components that shape the behavior through physical and aerodynamic heuristics:

$$r = r_{\text{vel}} + r_{\text{height}} + r_{\text{orient}} + r_{\text{progress}} + r_{\text{dist}} - p_{\text{low}} + r_{\text{bonus}}$$

The velocity term $r_{\textbf{vel}} = \max(0, v_x) \cdot r_{\textbf{orient\_factor}}$ promotes forward motion only when the bird maintains a stable orientation. Orientation stability is captured by $r_{\textbf{orient\_factor}} = \exp(-2(|\phi| + |\theta|))$, where $\phi$ and $\theta$ are the roll and pitch angles respectively. This factor multiplies several other rewards to ensure that progress is only rewarded when the bird remains reasonably level. Additionally, a direct orientation penalty is included: $r_{\textbf{orient}} = -(|\phi| + |\theta|)$, discouraging aggressive tilting.

To maintain altitude, we include a height reward $r_{\textbf{height}} = \exp(-0.5 \cdot |z - 7.0|) \cdot r_{\textbf{orient\_factor}}$, which rewards the agent for flying close to a desired vertical position. Forward progress is encouraged using a normalized $x$-position metric, $r_{\textbf{progress}} = \frac{x+8.0}{16.0}$, which grows linearly as the bird moves along the horizontal axis. In addition to local progress, a global goal-based shaping reward is included: $r_{\textbf{dist}} = \exp(-0.1 \cdot \|\mathbf{x}_{\textbf{current}} - \mathbf{x}_{\textbf{target}}\|)$, where the target position is set at $(8.0, 0.0, 7.0)$.

To penalize dropping too low, we add a height penalty $p_{\mathbf{low}} = 3.0 - z$ if the vertical position $z$ falls below 3.0. Finally, a small living bonus $r_{\mathbf{bonus}} = 0.1$ is added at each step to encourage prolonged flight and avoid premature termination of episodes.

Although this reward function attempts to shape desirable flight patterns, it can overconstrain the learning process. In practice, we observed that the bird often fails to discover effective flapping strategies, instead learning behaviors that simply avoid penalties without achieving efficient or sustained flight. This indicates that some reward terms—particularly orientation constraints—may dominate the gradient signal and hinder effective exploration during training.

For a similar flapping robot model that has been implemented by a team from University of California Berkeley [2], the reward function they used was as follows:

$$r = 0.5r_{\text{pos}} + 0.1r_{\Omega} + 0.2r_{\phi,\,\theta} + 0.05r_{\text{energy}}$$

The reward function balances trajectory tracking, stability, and efficiency: $r_{\text{pos}}$ minimizes positional error, $r_{\Omega}$ penalizes high angular velocities to promote smooth motion, and $r_{\phi,\theta}$ encourages level orientation for consistent horizontal thrust. An energy term $r_{\text{energy}}$ promotes gliding and discourages excessive flapping, aligning the learned behavior with that of real birds.

This reward worked well for a similar flapping bird model that is more optimized and with higher fidelity for trajectory tracking.

# 4 Methods and Algorithms [3]

## 4.1 Proximal Policy Optimization (PPO)

PPO is the only on-policy algorithm I tried known for its simplicity and sample efficiency [4]. It uses a clipped objective to avoid large policy updates, which improves training stability. PPO was found to **learn much faster** in early training stages compared to SAC and TD3, I was able to run 3000 Episodes in few minutes.

**Pros:**

- Learns very quickly.
- Easy to implement and tune.
- Stable learning dynamics due to the clipped surrogate loss.

**Cons:**

- As an on-policy method, it is less sample-efficient overall.
- It tends to converge to suboptimal flapping patterns.
- Learned behaviors are often unstable over long trajectories, with poor control over orientation and altitude.

## 4.2 Soft Actor-Critic (SAC)

SAC is an off-policy actor-critic algorithm that optimizes a stochastic policy using entropy regularization [5]. The inclusion of entropy maximization encourages exploration and results in more robust policies. In the flapping bird environment, SAC learned smoother and more consistent control strategies than PPO, albeit at the cost of slower convergence.

**Pros:**

- Learns robust and smooth flapping behaviors.
- Handles high-dimensional continuous action spaces effectively.
- Exploration is naturally encouraged through entropy regularization.

**Cons:**

- Slower to converge than PPO.
- More sensitive to hyperparameter tuning and reward scaling.

### 4.3 Twin Delayed Deep Deterministic Policy Gradient (TD3)

TD3 is another off-policy algorithm that improves upon DDPG by reducing overestimation bias in Q-values through the use of twin critics [6]. It uses deterministic policies and delayed policy updates for stability. In the flapping task, TD3 discovered **high-quality solutions** that maintained forward flight while stabilizing roll and pitch.

**Pros:**

- Produces highly stable and efficient flapping trajectories.
- Deterministic policy helps in fine-grained control of wing actuation.
- Lower variance and improved stability compared to DDPG.

**Cons:**

- Slower initial learning curve than PPO.
- Requires careful tuning of target noise, update frequencies and hyperparameters.

### 4.4 Comparison

- PPO learns faster but converges to unstable or less optimal behaviors.
- SAC and TD3 find better overall solutions, especially with respect to orientation, long-term stability and hovering.
- In practice, TD3 and SAC outperform PPO when learning complex motor patterns like coordinated flapping.

## 5 Experimental Setup

Since training requires significant amount of time and computational resources, hyperparameter tuning was constrained. However, I experimented using the same hyperparameters on two different reward functions. The hyperparameters shown in table 1 are based on educated estimation of good hyperparameters given my scenario.

Table 1: Hyperparameters used in SAC and TD3 agents

| **Hyperparameter** | **SAC** | **TD3** | **PPO** |
|---|---|---|---|
| Learning Rate | 0.0003 | 0.0003 | 0.0005 |
| Discount Factor ($\gamma$) | 0.99 | 0.99 | 0.995 |
| Target Update Rate ($\tau$) | 0.005 | 0.005 | – |
| Batch Size | 512 | 512 | Full episode |
| Policy Hidden Layers | [128, 128] | [128, 128] | [128, 128] |
| Q-Network Hidden Layers | [128, 128] | [128, 128] | – |
| Action Noise Std. | – | 0.1 | Implicit |
| Policy Delay | – | 2 | – |
| Target Policy Noise | – | 0.2 | – |
| Replay Buffer Size | 1,000,000 | 1,000,000 | – |
| Action Range | [-1.5, 1.5] | [-1.5, 1.5] | [-1.5, 1.5] |
| Entropy Temperature ($\alpha$) | 0.2 | – | – |
| KL Coefficient | – | – | 0.2 |
| Value Function Coefficient | – | – | 0.5 |
| Optimizer | Adam | Adam | AdamW |

I trained the models on CPU using PyTorch. For each model, the number of training episodes varied and was optimized with early stopping based on learning. During the training, I save the model checkpoints and evaluate them to choose the model with the best learning parameters.

# 6 Results [1]

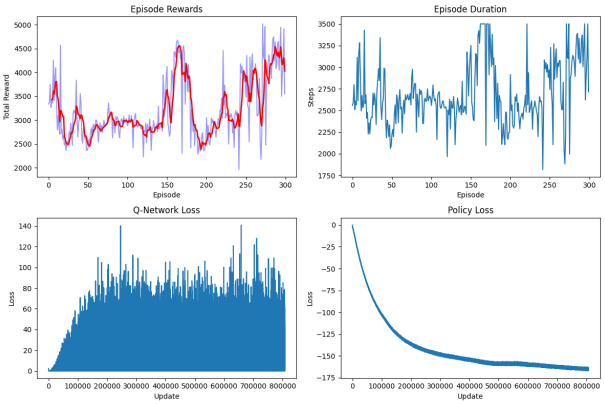


Figure 2: SAC with Simple Reward Function

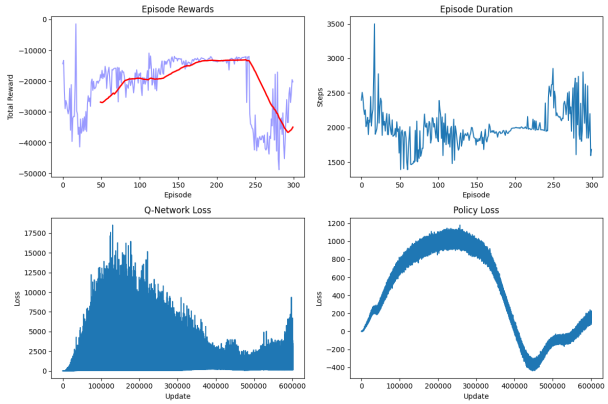


Figure 3: SAC with Complex Reward Function

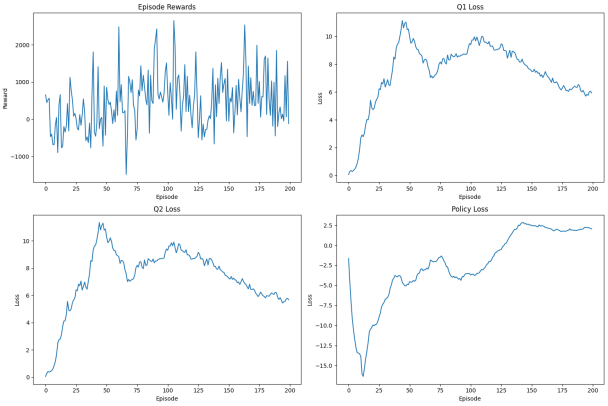


Figure 4: TD3 with Simple Reward Function

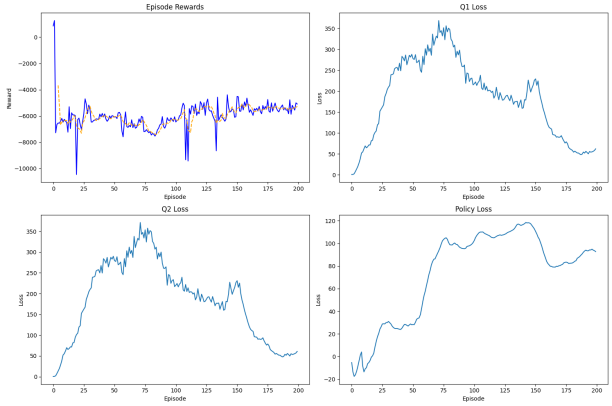


Figure 5: TD3 with Complex Reward Function

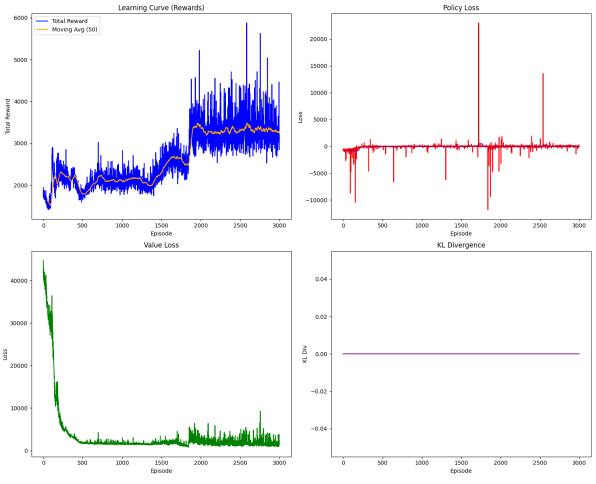


Figure 6: PPO with Simple Reward Function

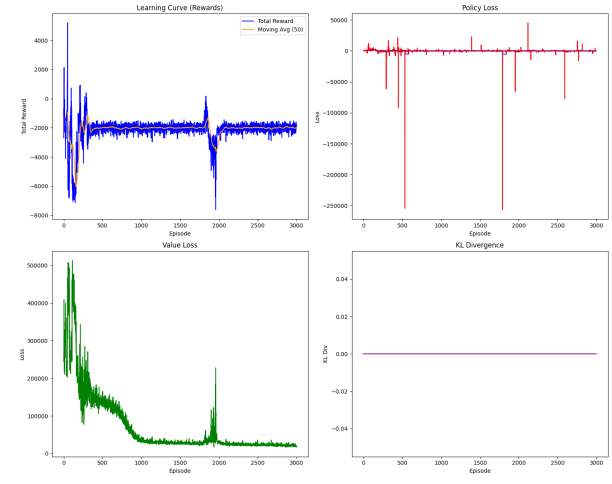


Figure 7: PPO with Complex Reward Function

[1] Source code and agent visualizations found at: https://github.com/Mostafa12d/RLFlapping/tree/main

# 7 Analysis

- **Comparison in flight between SAC, TD3, and PPO:** SAC consistently showed smoother learning and adaptation across both simple and complex reward functions. The flapping behavior was more stable, with gradual improvement in reward and policy loss over time. TD3 had noisier learning curves and was more prone to instability, especially under the complex reward function. PPO initially showed high variance but improved stability in value loss and rewards, particularly with the simple reward function.
- **Effects of reward function:** All agents learned better under the simple reward function, which provided clearer, denser feedback for forward motion with PPO agent displaying an interesting performance where the bird flipped upside down and glided horizontally to maximize rewards. The complex reward function, caused TD3, SAC and PPO to learn unstable flying behavior and failed to use the wings and collapsed to suboptimal policies. This could be because the agent is confined by multiple penalties and reward and is not able to find the optimum policy that satisfies all constraints. Hence, it hacks the reward function. **Note**: In the simple reward, the increment of the reward each step was small that's why the learning curves are not resembling the traditional curve.
- **Number of episodes required to learn how to flap properly:** SAC agents began to show coordinated flapping behavior within the first 100–150 episodes in the simple reward case, evident from increasing reward and reduced Q-loss. TD3 took longer ( 200+ episodes) and occasionally regressed, however, it also learned optimal flight patterns. PPO exhibited faster initial reward gains with the simple function but plateaued early in the complex case, likely due to limited exploration or vanishing gradients and couldn't learn how to flap.
- **Computation cost and learning time:** SAC was more computationally expensive due to the twin Q-networks and entropy regularization but yielded more reliable convergence. TD3's use of target policy smoothing helped reduce overestimation but required more time to stabilize and train. PPO was faster per iteration due to its policy gradient nature but required more careful tuning of clipping and KL divergence settings.
- **Effects of modifying state space and action space:** Reducing the state/action space simplified the problem and led to faster learning in SAC, TD3 and PPO. Expanding the action space (e.g., including orientation control or variable wing pitch) introduced more instability, learning time and insufficient learning. The state space could be increased to include all environment states, but that will also increase the complexity considereably.

# 8 Discussion and Future Work

Training reinforcement learning agents for flapping flight presents unique challenges due to the underactuated dynamics and delayed aerodynamic responses. The results show that:

- **Reward shaping is crucial:** A well-designed reward function can dramatically accelerate learning. Simple reward functions that promote forward velocity and stable altitude were effective across algorithms. Complex rewards need careful balancing to avoid sparse gradients or misleading signals.
- **SAC had the best flapping behavior:** The entropy-based exploration in SAC enabled it to discover periodic flapping behaviors, although it required more computation and longer training. This makes SAC a strong candidate for environments with sparse rewards or deceptive local optima.
- **TD3 needs fine-tuning for stability:** While theoretically robust to overestimation bias, TD3 showed high variance in Q-loss and struggled to generalize with the complex reward. It may benefit from more conservative update steps or better action noise strategies. However, TD3 perforfmed really well and learned the flapping behavior. Although less efficient than SAC, it is considered a success.
- **PPO is sample-efficient but brittle:** PPO learned quickly under simpler conditions but was sensitive to changes in the state and reward structure. Future work could explore adaptive clipping strategies or trust-region hybridization to improve robustness. Longer training periods and higher exploration could allow PPO to improve its performance in learning the flapping behavior.
- **Future Work**
  - Add tail and wing pitch control to improve maneuverability and recover from drift.
  - Use curriculum learning: start with tasks like hovering or sinusoidal flapping, then progress to long-range flight, then finally trajectory tracking.

- Extend to flight in simulated wind conditions or constrained tunnels and unstructured environments.
- Integrate temporal reward shaping (e.g., penalize jerky wing motions) and environmental disturbances for generalization. Carefully shape the reward function to be in between complex and simple to be able to conquer harder tasks.

# References


[1] DeepMind. *MuJoCo Documentation*, 2023. `https://mujoco.readthedocs.io/`, Accessed: 2025-04-12.

[2] Jiaze Cai, Vishnu Sangli, Mintae Kim, and Koushil Sreenath. Learning-based trajectory tracking for bird-inspired flapping-wing robots, 2024.

[3] SAC TD3 and PPO Implementations Ex7. CS 5180: Reinforcement Learning and Sequential Decision Making.

[4] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. In *arXiv preprint arXiv:1707.06347*, 2017.

[5] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv preprint arXiv:1801.01290*, 2018.

[6] Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *Proceedings of the 35th International Conference on Machine Learning (ICML)*, pages 1587–1596. PMLR, 2018.