Technische Hochschule Ingolstadt

Fakultät Informatik

Bachelorstudiengang Künstliche Intelligenz

Analysis of various Reinforcement Learning algorithms for Unmanned Aerial Vehicle Waypoint Navigation approaches

Bachelor Thesis

Felix Unterleiter

First examiner:	Prof. Dr. Christian Seidel
Second examiner:	Prof. Dr. Sören Gröttrup
Issued on:	February 4, 2025
Submitted on:	February 4, 2025

Speciment Declaration in Accordance with §30 Abs.4 Nr.7 APO THI

I hereby declare that this thesis is my own work, that I have not presented it elsewhere for examination purposes and that I have not used any sources or aids other than those stated. I have marked verbatim and indirect quotations as such.

30.09.2016, Ingolstadt

Ort, Datum

Verfasser der Arbeit

Abstract

Unmanned Aerial Vehicles (UAVs) have achieved extensive integration across multiple sectors, prompting the need for enhancements in autonomous navigation methodologies. Conventional waypoint navigation methods, including A*, D*, and PID controllers, encounter difficulties in dynamic contexts because of their dependence on pre-established maps and manually calibrated parameters. Reinforcement Learning (RL) offers a unique methodology, allowing UAVs to acquire navigation strategies via environmental interaction. This thesis examines the application of actor-critic reinforcement learning algorithms—Deep Deterministic Policy Gradient (DDPG), Soft Actor-Critic (SAC), and Proximal Policy Optimisation (PPO)—for unmanned aerial vehicle waypoint navigation. A tailored simulation environment was developed using PyFlyt and Gymnasium, featuring an observation space augmented with angular positional data and a reward function specifically suited for stable, efficient navigation. Comparative experiments were performed to assess algorithm performance across various control schemes, including characteristics such as sample efficiency, incentive stability, and convergence behaviour.

The results demonstrate that PPO consistently surpasses DDPG and SAC in trustworthy waypoint navigation, exhibiting enhanced stability and sampling efficiency. The research emphasises the significance of observation space design, hyperparameter optimisation, and command architecture in reinforcement learning applications for UAVs. These findings establish a basis for future advancements in multi-agent reinforcement learning (MARL) and practical UAV implementation.

Disclaimer

This thesis was conducted in cooperation with Airbus Defence and Space. The framework and data employed for both training and evaluation were carefully aligned with the application scope defined by Airbus. By ensuring that all methodologies and datasets remained within the specified scope, the integrity and relevance of the study to real-world applications were maintained and verified throughout the project.

Contents

1 Introduction

1.1 Motivation

Unmanned Aerial Vehicles (UAVs) have garnered considerable attention in recent years thanks to their versatility and extensive applications, including in agriculture, urban mapping, and remote sensing [Debnath et al., 2024]. Recent developments in sensor technology, communication systems, and embedded computing capabilities have made possible the autonomous or semi-autonomous operation of UAVs [Wilson et al., 2022]. A primary challenge in autonomous UAV operations is waypoint navigation, which involves a UAV traversing a series of predefined waypoints in order to arrive at its final destination.

Waypoint navigation usually relies on path-planning algorithms, including A*, D*, and Rapidly-exploring Random Trees (RRT*). The A* algorithm has become popular as it ensures optimal solutions in discrete environments and systematically explores search spaces while avoiding obstacles [Mandloi et al., 2021]. A* requires a fully or partially known map and is limited to grid-based representations, which restricts its adaptability to dynamic environments. D*, an extension of A*, improves dynamic replanning capabilities in response to detected obstacles or terrain shifts, thereby making it suitable for UAV navigation in evolving environments [Suanpang and Jamjuntr, 2024]. Sampling-based methods, such as RRT*, demonstrate higher efficiency in high-dimensional and continuous spaces; however, they may suffer from significant computational costs in real-time applications [Chen and Yu, 2021].

Although path-planning algorithms demonstrate performance in structured environments, their practicality decreases in real-world applications where UAVs need to adjust to unexpected disturbances. In scenarios free of either static or dynamic obstacles, a Proportional-Integral-Derivative (PID) controller can be employed to effectively navigate the UAV between waypoints by adapting control inputs in proportion to the difference between desired and actual flight parameters. PID controllers exhibit inherent limitations due to their dependence on fixed parameters, linear assumptions [Oersted and Ma, 2023], and vulnerability to sensor noise and external disturbances, such as variations in airflow [Shi, 2024]. The identified limitations hinder adaptability in dynamic flight conditions.

Reinforcement Learning (RL) is recognised as a significant area of research in traditional control and planning [Dong et al., 2022], owing to its ability to facilitate UAVs in learning optimal navigation policies via trial and error. In contrast to PID controllers that depend on specifically tuned parameters, reinforcement learning (RL) facilitates adaptation to environmental changes and generalisation to novel situations without necessitating an explicit model of the environment [Sutton and Barto, 2018].

1.2 Objective

This thesis investigates reinforcement learning methods for waypoint navigation in unmanned aerial vehicles (UAVs), emphasising the significance of three particular actor-critic algorithms: Deep Deterministic Policy Gradient (DDPG), Soft Actor Critic (SAC), and Proximal Policy Optimisation (PPO) within single-agent contexts. Traditional navigation strategies, such as A*, D*, and PID control, have been extensively analysed; however, these approaches reveal fundamental limitations in dynamic and volatile environments. Reinforcement learning provides a viable alternative by allowing unmanned aerial vehicles to develop navigation policies through interaction with their environment, without reliance on explicit map-based planning or predefined control parameters [Sutton and Barto, 2018].

This thesis is situated within the broader scope of Multi-Agent RL (MARL), where multiple UAVs operate in a common airspace, either cooperatively or adversarially. A proposed framework for staged pretraining is presented, wherein:

1. A single UAV is initially trained to navigate through static waypoints in an isolated environment.
2. The agent is then fine-tuned to handle dynamic waypoints that change in real time.
3. A second UAV is introduced, tasked with learning collision avoidance while efficiently navigating to a designated waypoint. During this stage, the primary agent remains stationary in its learnt policy, ensuring a stable environment for the second UAV to adapt.
4. Finally, both agents are jointly adapted in a multi-agent framework where policies must be learnt under non-stationary conditions [Papoudakis et al., 2019].

This thesis focuses solely on the single-agent setting as an initial step, establishing a stable baseline prior to advancing to more complex MARL architectures. The main research objectives are outlined below:

The primary research objectives are as follows:

- **Exploration of Actor-Critic Algorithms**: Evaluate the suitability of three distinct actor-critic methods for waypoint navigation, assessing their convergence properties, stability, and sample efficiency.
- **Hyperparameter Sensitivity Analysis**: Investigate the impact of hyperparameter tuning (e.g., learning rates, discount factors, entropy regularisation) on the overall navigation performance.
- **State-Space and Reward Function Design**: Identify and compare different state-space representations, including angular positional data, velocity information, and auxiliary sensor readings, to determine their effectiveness in policy learning. Additionally, investigate suitable reward formulations to maximise learning progression and task completion efficiency.
- **Comparison of Action Spaces**: Examine the implications of angular and thrust control on flight behaviour, highlighting their respective advantages and challenges.

Through these objectives, this thesis seeks to establish a clear reinforcement learning framework for UAV waypoint navigation, which can subsequently be adapted for multi-agent scenarios involving the coordination of multiple UAVs in navigation and collision avoidance within shared airspace.

2 Theoretical Foundation

2.1 Reinforcement Learning

Reinforcement Learning (RL) as a subfield of machine learning involves a semi-supervised learning approach where an agent iteratively interacts with an exterior environment, receives information about the updated dynamics of said environment as well as a reward signal, and evaluates the agent's performance according to some predefined evaluation metric. The framework for RL is defined by a Markov Decision Process (MDP), consisting of states (S), actions (A), transition dynamics (P), rewards (R), and a discount factor (γ). The given MDP is formally represented by the tuple (S, A, P, R, γ), where S represents the set that defines all conceivable states that the agent might observe at every given timestep t. A defines the set of actions from which the agent can choose. The Transition Dynamics P formally describes the probability that action a in state s leads to state s', denoted as $P_a(s, s') = \Pr(s_{t+1} = s' \mid s_t = s, a_t = a)$.
R is a function that sends back a numerical reward signal after the agent does something. It rewards the agent for moving to a new state that is desirable and punishes states that are undesirable, which has a big effect on how the agent learns. γ, the discount factor defined between 0 and 1 that determines the present value of future rewards, reflecting the trade-off between immediate and delayed rewards. [Sutton and Barto, 2018]
The agent's objective function is formulated to maximise cumulative rewards through its actions, guided by a policy π, mapping the perceived states of the environment at each timestep t to actions that should be taken in those states. π can be defined as a deterministic mapping, where actions are explicitly defined for each state, which often is unfeasible for significantly large state- and action-spaces or MDPs where the dynamics are too complex for deterministic solutions. Value Function: A prediction of the expected return from each state or state-action pair, guiding the agent's decision-making. The state-value function ($V^\pi(s)$) estimates the expected return, i.e., the discounted sum of future returns from state s under the current policy π, while the action-value function ($Q^\pi(s, a)$) estimates the expected return from taking action a in state s under the current policy π.

The primary goal in RL can be broken down to finding an optimal policy (π^*) that maximises the expected return from all states, formalised as the objective function $J(\theta) = \mathbb{E}\tau \sim \pi\theta\,[R(\tau)]$ where τ represents a trajectory of states and actions, θ represents the parameters of the policy in parameterised policies such as those of a neural network.

2.2 Actor-Critic Algorithms

RL techniques can be divided broadly into value-based and policy-based methods. Value-based approaches focus on estimating the value function and derive the optimal policy by greedily choosing actions that maximise these value estimates. Policy-based approaches parametrise and directly optimise a policy function that maps states to actions without explicitly estimating value functions. [Sutton and Barto, 2018]

Actor-Critic algorithms are a class of methods combining policy- and value-based approaches to optimise decision-making processes. These algorithms consist of two primary components: the actor and the critic. The actor selects actions based on a policy, denoted as $\pi(a|s)$, which defines the probability of taking action a in state s. The critic is tasked with evaluating the actions taken by the actor by estimating a value function, such as the state-value function $V(s)$ or the action-value function $Q(s, a)$, providing feedback on the quality of the actions to guide the learning process [Konda and Tsitsiklis, 1999].

These two components effectively address the limitations inherent in purely policy-based (actor-only) and value-based (critic-only) approaches. Policy-based methods optimise the policy directly but suffer from high variance in gradient estimates [Riedmiller et al., 2007], leading to instability and slow learning. Conversely, value-based techniques, although frequently achieving faster convergence, have their overall convergence guaranteed only in very restricted contexts [Konda and Tsitsiklis, 1999], failing to guarantee policy convergence even in "simple MDPs" [Sutton et al., 1999].

Actor-Critic algorithms mitigate these issues by allowing the actor to utilise the critic's evaluations, reducing variance in gradient estimates, while introducing bias [Nachum et al., 2017]. This trade-off leads to more stable training while maintaining theoretical convergence guarantees when specific conditions are met [Konda and Tsitsiklis, 1999].

In the following section, the Deep Deterministic Policy Gradient (DDPG), Soft Actor-Critic (SAC), and Proximal Policy Optimisation (PPO) algorithms will be introduced, each leveraging this actor-critic framework. Their learning dynamics will be outlined, emphasising their distinctions, individual strengths, and limitations. Additionally, their selection in the context of waypoint navigation will be rationalised.

2.2.1 Deep Deterministic Policy Gradient

Deep Deterministic Policy Gradient (DDPG) is a model-free[1], off-policy [2] reinforcement learning algorithm specifically designed for environments with continuous action spaces [Lillicrap et al., 2019]. The algorithm is based on the actor-critic framework and incorporates key components from Deep Q-Networks (DQNs), including the implementation of an experience replay buffer and target networks [Mnih et al., 2015]. DDPG overcomes the limitations of conventional Q-learning methods, which are generally restricted to discrete action spaces, by developing deterministic policies that directly link states with specific actions.

Replay Buffer

The replay buffer is a vital component of DDPG, functioning as a memory module, represented as $\mathcal{D}$, in which transitions collected throughout interactions with the environment are saved. Transitions are denoted as tuples (s_t, a_t, r_t, s_{t+1}), with s_t representing the state at time t, a_t indicating the executed action, r_t signifying the immediate reward after taking action a_t, and s_{t+1} the subsequent state. [Lillicrap et al., 2019]
Transitions are sequentially collected and appended to the buffer during the agent's interaction with the environment. The buffer functions as a finite-sized cache [Lillicrap et al., 2019], wherein new transitions replace the oldest ones upon reaching its maximum capacity N. This allows the agent to utilise prior experiences for several updates, enhancing sample efficiency as D does not require complete repopulation after an update to the actor and critic is completed.
During training, transitions are sampled uniformly into mini-batches of size B from the replay buffer. The sampled mini-batch is denoted as $\{(s_i, a_i, r_i, s'_i)\}_{i=1}^{B}$. This random sampling disrupts the temporal correlations present in consecutive transitions [Mnih et al., 2015]. The replay buffer reduces the likelihood of biased updates, which may otherwise result in instability during training. Moreover, random sampling guarantees that the data distribution utilised for training is diverse, including a wider range of the

[1] Model-free algorithms learn optimal behaviour without relying on an explicit model of the environment's dynamics. They update value functions or policies directly from observed rewards and transitions. [Sutton and Barto, 2018]

[2] Off-policy algorithms are those who learn an optimal target policy while collecting data using a separate, so-called behaviour policy. [Sutton and Barto, 2018]

state-action space.

Critic Network

In DDPG, the critic network approximates the action-value function $Q(s, a|\theta^Q)$, which is parameterised by the critic-network parameters θ^Q. The action-value function represents the anticipated cumulative reward from state s, upon taking action a, and thereafter adhering to the policy μ. The critic's role is to assist the actor by providing gradients that indicate how the Q-value changes with respect to the action.
The critic is trained by minimising the temporal-difference (TD) error, which is defined as:

$$L(\theta^Q) = \mathbb{E}_{(s_t,a_t,r_t,s_{t+1})\sim\mathcal{D}}\left[\Big(y_t - Q(s_t, a_t|\theta^Q)\Big)^2\right], \tag{1}$$

where the TD-target y_t is given by:

$$y_t = r_t + \gamma Q(s_{t+1}, \mu(s_{t+1}|\theta^\mu)|\theta^Q). \tag{2}$$

In this context, r_t represents the immediate reward, γ denotes the discount factor, and s_{t+1} indicates the subsequent state. The target y_t includes the estimated Q-value of the subsequent state, enabling the critic's ability to accurately predict cumulative rewards.
The gradient of $L(\theta^Q)$ with respect to the critic's parameters θ^Q is computed through the chain rule to minimise the loss function:

$$\nabla_{\theta^Q} L(\theta^Q) = -\mathbb{E}_{(s,a,r,s')\sim\mathcal{D}}\left[2 \times \Big(y_t - Q(s, a|\theta^Q)\Big)\, \nabla_{\theta^Q} Q(s, a|\theta^Q)\right]. \tag{3}$$

This optimisation guarantees that the critic effectively assesses the quality of state-action pairs, offering the actor solid feedback for enhancing the policy. $\nabla_{\theta^Q} L(\theta^Q)$, the factor of 2 is frequently omitted, as it does not fundamentally alter the optimisation landscape and is effectively incorporated into the learning rate α during the parameter update 6.

Actor Network

The actor network in DDPG denotes a deterministic policy $\mu(s|\theta^\mu)$, with θ^μ representing the network parameters. The main objective of the actor is to optimise the policy parameters θ^μ in order to maximise the expected cumulative reward, denoted as the objective function $J(\theta^\mu)$.

The actor network utilises the deterministic policy gradient theorem for optimisation purposes. This theorem offers a method for calculating the gradient of the objective function with respect to the parameters of the actor:

$$\nabla_{\theta^\mu} J(\theta^\mu) = \mathbb{E}_{s_t \sim \mathcal{D}} \left[\nabla_a Q(s, a|\theta^Q) \nabla_{\theta^\mu} \mu(s|\theta^\mu) \right]. \tag{4}$$

where,

1. **Objective Function:** The actor's goal is to maximize the expected return, represented as

$$J(\theta^\mu) = \mathbb{E}_{s_t \sim \mathcal{D}}[Q(s, a|\theta^Q)]. \tag{5}$$

   This objective ensures that the actor selects actions that lead to states with high Q-values, as predicted by the critic network.

2. **Critic Network Guidance:** The critic network $Q(s, a|\theta^Q)$ estimates the expected return for a given state-action pair. It acts as a guide for the actor, providing a scalar value (the Q-value) that quantifies how good a particular action a is in state s.

3. **Chain Rule for Policy Gradient:** To update the actor's parameters θ^μ, the gradient $\nabla_{\theta^\mu} J(\theta^\mu)$ is computed. This involves:

   - $\nabla_a Q(s, a|\theta^Q)$: The derivative of the Q-value with respect to the action a, indicating how the Q-value changes as the action changes. This term provides feedback on the direction in which the action should change to increase the expected return.
   - $\nabla_{\theta^\mu} \mu(s|\theta^\mu)$: The derivative of the actor's output (action) with respect to its parameters, showing how the actor's policy changes with updates to its parameters.

4. **Policy Update:** Combining these derivatives, the actor network updates its parameters θ^μ in the direction that maximizes the Q-value. The gradient ascent step is given by:

$$\theta^\mu \leftarrow \theta^\mu + \alpha \nabla_{\theta^\mu} J(\theta^\mu), \tag{6}$$

where α is the learning rate. This step aligns the actor's policy with the critic's guidance, ensuring that the actor selects actions that maximize the expected return.

DDPG enhances exploration by adding noise to the actions a during training [Lillicrap et al., 2019]. Originally, time-correlated Ornstein-Uhlenbeck (OU) noise was used to promote exploration in physical control problems. However, recent research suggests that uncorrelated Gaussian noise is comparably effective while being simpler to implement. [Hollenstein et al., 2023]

Target Networks

Target networks are utilised in DDPG to stabilise the training process by offering consistent targets for updating the actor and critic networks. The target networks, μ' and Q', serve as gradually updated replicas of the primary actor and critic networks. Their main function is to reduce instability and oscillation in the policy. [Mnih et al., 2015]
The instability in TD learning is partly due to the Q-value update relying on the current parameters of the critic network, which are continuously updated throughout the training process. This phenomenon, known as "concept drift" [Widmer and Kubat, 1996], arises when the target that the Q-function aims to approximate undergoes changes during training, potentially resulting in instability within the learning process [Mnih et al., 2015] [Kim et al.,].
This instability, while not intrinsic to the actor network, propagates from the value network due to the actor's dependence on the critic for parameter updates. The introduction of target networks in DDPG facilitates a gradual and predictable change in the targets utilised in the critic's loss function, thereby smoothing the learning process. However, this approach results in a deceleration of learning due to delayed updates of the value function [Kim et al.,].

For the critic network, the TD target is defined as:

$$y_t = r_t + \gamma Q'(s_{t+1}, \mu'(s_{t+1}|\theta^{\mu'})|\theta^{Q'}), \tag{7}$$

where Q' is the target critic network, μ' is the target actor network.
The parameters of the target networks are then updated using a soft update mechanism, defined as:

$$\begin{aligned} \theta^{Q'} &\leftarrow \tau\theta^{Q} + (1-\tau)\theta^{Q'}, \\ \theta^{\mu'} &\leftarrow \tau\theta^{\mu} + (1-\tau)\theta^{\mu'}, \end{aligned} \tag{8}$$

Here, $\tau \ll 1$ represents a hyperparameter that regulates the rate of updates. This soft update facilitates the gradual evolution of target networks, thereby avoiding abrupt alterations in target values that may disrupt the learning process.

Rationale

DDPG was selected as one of the algorithms for this thesis due to its application in previous studies on UAV waypoint navigation, which illustrates its capability to produce trajectories that effectively reach designated waypoints [Bouhamed et al., 2020] [Li and Wu, 2020].

2.2.2 Soft Actor Critic

Soft Actor-Critic (SAC) is an off-policy, model-free reinforcement learning algorithm. SAC retains core structural similarities with DDPG, such as employing separate networks for policy (actor) and value estimation (critic) and saving past experiences in a replay buffer. However, SAC introduces key concepts addressing limitations inherent in DDPG, making it more robust to hyperparameters and stable in its learning process [Haarnoja et al., 2018]. DDPG is heavily reliant on deterministic policies and value-based learning, which can lead to brittle learning dynamics [Haarnoja et al., 2018]. Deterministic policies tend to overfit to specific trajectories, limiting exploration, leading to convergence to suboptimal behaviour [Plappert et al., 2017]. SAC adopts a stochastic policy representation, wherein the actor network outputs a probability distribution over actions instead of a single deterministic action. Unlike DDPG, which optimises purely for cumulative reward and exploration is only achieved by adding noise to the actions during training, SAC naturally incorporates exploration by leveraging the maximum entropy framework [Ziebart et al., 2008]. This involves adding an entropy term to the objective function, encouraging policies to remain stochastic, consequently avoiding early exploitation of suboptimal policies. Susceptibility to instability, present in DDPG, is partly due to overestimation bias in value function approximations, which is overcome in SAC through the use of twin Q-networks, which minimises overestimation by selecting the minimum value between two independent critics when computing the target Q-value. This approach enhances stability by ensuring more reliable value estimates. [Fujimoto et al., 2018a]

Actor Network

The policy in SAC, parametrized by the actor network, is a stochastic policy that optimizes the maximum entropy objective. Unlike deterministic policies used in algorithms like DDPG, SAC employs a stochastic approach where the actor samples actions from a distribution during training to encourage effective exploration. The policy is updated to minimize the *Kullback-Leibler (KL) divergence* between itself and a Boltzmann distribution over the Q-values, aligning with the maximum entropy framework. The policy is typically modeled as a Gaussian distribution:

$$\pi_\phi(a|s) = \mathcal{N}(\mu_\phi(s), \sigma_\phi(s)), \tag{9}$$

where $\mu_\phi(s)$ and $\sigma_\phi(s)$ are the mean and standard deviation of the action distribution, parameterized by the policy network. The objective function for the policy aims at minimizing the KL divergence between the policy $\pi_\phi(a|s_t)$ and a Boltzmann distribution derived from the Q-function:

$$J_\pi(\phi) = \mathbb{E}_{s_t \sim \mathcal{D}} \left[D_{\mathrm{KL}} \left(\pi_\phi(\cdot|s_t) \middle\| \frac{\exp(Q_\theta(s_t, \cdot))}{Z_\theta(s_t)} \right) \right]. \tag{10}$$

where

- D_{KL} is the KL divergence, which measures the difference between the policy distribution and the target Boltzmann distribution.
- $\exp(Q_\theta(s_t, \cdot))$ represents an unnormalized softmax distribution over the Q-values.
- $Z_\theta(s_t)$ is the partition function:

  $$Z_\theta(s_t) = \int \exp(Q_\theta(s_t, a))\, da,$$

  which normalizes the exponentiated Q-function, allowing it to define a valid probability distribution that integrates to 1. This normalization is essential to ensure the KL divergence is well-defined and mathematically consistent. [Wasserman, 2023]

After simplifying the objective function under the definition of the KL-divergence [Lahire, 2021], a remaining issue persists which is that direct sampling from the Gaussian distribution 9 is non-differentiable with respect to the mean and variance, making

gradient computation problematic [Gundersen, 2018]. The reparameterization trick resolves this by transforming the sampling process into a deterministic function of a fixed noise variable ϵ [Lahire, 2021]:

$$a = f_\phi(\epsilon_t; s_t) = \mu_\phi(s) + \sigma_\phi(s) \cdot \epsilon, \quad \epsilon \sim \mathcal{N}(0, 1). \tag{11}$$

Substituting $\pi_\phi(a|s)$ with the linear transformation $f_\phi(\epsilon_t; s_t)$ the final objective of the policy can be defined as:

$$J_\pi(\phi) = \mathbb{E}_{s_t \sim \mathcal{D}, \epsilon_t \sim \mathcal{N}} \left[\log \pi_\phi(f_\phi(\epsilon_t; s_t)|s_t) - Q_\theta(s_t, f_\phi(\epsilon_t; s_t)) \right], \tag{12}$$

and the subsequent gradient of $J_\pi(\phi)$ with respect to the actors parameters ϕ as:

$$\nabla_\phi J_\pi(\phi) = \nabla_\phi \log \pi_\phi(a_t|s_t) + (\nabla_{a_t} \log \pi_\phi(a_t|s_t) - \nabla_{a_t} Q_\theta(s_t, a_t)) \nabla_\phi f_\phi(\epsilon_t; s_t), \tag{13}$$

Value- and Q-Network

In SAC, both the value function and the Q-function play integral roles in the learning process. While DDPG relies solely on a Q-function to approximate the expected return for a given state-action pair, SAC employs an additional approximate value function $V(s)$. Given that the value $V(s)$ of any arbitrary state s can be derived from the Q-Network $Q(s, a)$ and the policy $\pi(a|s)$ a separate network for approximating $V(s)$ is theoretically not needed. The theoretical redundancy is justified in its practical application, where it aids in the stability of the algorithm while further being "convenient" to optimise alongside the Q-Function as well as the stochastic policy. [Haarnoja et al., 2018]
The Q-function in SAC, denoted as $Q_\phi(s, a)$, represents the expected cumulative reward starting from state s, taking action a, and subsequently following the current policy π. Its objective function $J_Q(\theta)$ minimises the squared error between the estimated and actual Q-value and is therefore similarly defined as for DDPG 1:

$$J_Q(\theta) = \mathbb{E}_{(s_t, a_t) \sim \mathcal{D}} \left[\frac{1}{2} \left(Q_\theta(s_t, a_t) - y_t \right)^2 \right], \tag{14}$$

where the Temporal Difference (TD) target y_t is given by:

$$y_t = r + \gamma \mathbb{E}_{s_{t+1} \sim p} \left[V_\psi(s_{t+1}) \right]. \tag{15}$$

The Q-Function in SAC takes on the identical role as in DDPG, measuring the value of state-action pairs and providing gradients,

$$\hat{\nabla}_\theta J_Q(\theta) = \nabla_\theta Q_\theta(a_t, s_t) \left(Q_\theta(s_t, a_t) - r(s_t, a_t) - \gamma V_\psi(s_{t+1})\right). \tag{16}$$

to aid in policy improvement while also being utilised to refine the approximation of the Value Function 17.

The value function $V_\psi(s)$, represents the expected return of a state s under the current policy π, integrating over all possible actions. Its objective is to minimise the residual error between each value estimate and the Q-function value over the entire action distribution:

$$J_V(\psi) = \mathbb{E}_{s_t \sim \mathcal{D}} \left[\frac{1}{2} \left(V_\psi(s_t) - \mathbb{E}_{a_t \sim \pi_\phi} \left[Q_\theta(s_t, a_t) - \log \pi_\phi(a_t|s_t) \right] \right)^2 \right]. \tag{17}$$

To ensure consistency with the maximum entropy objective of SAC, which incorporates the entropy of the action distribution into the policy optimisation, the term $\log \pi_\phi(a_t|s_t)$ is subtracted from the Q-function estimate when computing the value function. This ensures the value function accurately reflects the expected return under the stochastic policy. The gradient of $J_V(\psi)$ can be subsequentially defined as:

$$\hat{\nabla}_\psi J_V(\psi) = \nabla_\psi V_\psi(s_t) \left(V_\psi(s_t) - Q_\theta(s_t, a_t) + \log \pi_\phi(a_t|s_t) \right). \tag{18}$$

Twin Q-Networks

Two separate Q-networks, commonly referred to as *Twin Q-Networks*, are used in SAC to mitigate the overestimation bias commonly observed in single Q-function approximations. This approach originates from *Double Q-Learning*, where it was introduced to address similar issues in value-based reinforcement learning. Overestimation occurs when the Q-function overestimates the value of certain actions due to inherent data noise or model bias [He and Hou, 2020]. This can result in suboptimal policy updates, as the policy becomes biased toward selecting actions that yield lower-than-expected returns. In certain cases, this bias can cause the learning process to diverge or fail altogether, preventing convergence to an optimal policy [Thrun and Schwartz, 1993]. This most commonly occurs in the early stages of training, where there is documented to be a primacy bias, i.e., "a tendency to overfit early experiences" [Nikishin et al., 2022].

By maintaining *two separate Q-functions*, $Q_{\theta_1}(s, a)$ and $Q_{\theta_2}(s, a)$ [Fujimoto et al., 2018b], which are parameterised independently, the minimum of those two Q-values is used to

compute the target for the TD-target y_t in the Q-Network objective function, providing the more conservative and less optimistic estimate:

$$y_t = r + \gamma \min_{i=1,2} Q_{\theta_i}(s_{t+1}, a_{t+1}) - \alpha \log \pi(a_{t+1}|s_{t+1}). \tag{19}$$

Rationale

SAC was selected to address limitations observed in DDPG, namely sensitivity to hyperparameter tuning and the risk of suboptimal convergence. Although, to the best of my knowledge, SAC has not been widely applied in UAV waypoint navigation, it has been shown to be effective in general UAV control [Barros and Colombini, 2020]. Additionally, its improved stability and explicit exploration objective, as explored in this section, make it a compelling algorithm to consider for this task.

2.2.3 Proximal Policy Optimization

Proximal Policy Optimisation (PPO), as another actor-critic algorithm, shares foundational similarities with DDPG and SAC, such as leveraging function approximators for both policy and value estimation. [Schulman et al., 2017b]
PPO introduces a surrogate objective function with a clipping mechanism that constrains policy updates to a small trust region with the intent of improving the stability of training [Schulman et al., 2017b], ensuring policy updates are not too large, while maintaining computational simplicity, which is a major concern in similar trust-region algorithms like Trust Region Policy Optimisation (TRPO) [Schulman et al., 2017a]. Unlike SAC, which explicitly optimises for stochastic policies to balance exploration and exploitation, PPO employs a regularising entropy term in its main objective function to promote exploration in training.
PPO optimizes a composite objective function $L_t^{\text{CLIP+VF+S}}(\theta)$:

$$L_t^{\text{CLIP+VF+S}}(\theta) = \mathbb{E}_t \left[L_t^{\text{CLIP}}(\theta) - c_1 L_t^{\text{VF}}(\theta) + c_2 S[\pi_\theta](s_t) \right], \tag{20}$$

where each term represents a distinct contribution to the learning process, which will be explored on the following pages.

Vanilla Policy Gradient Objective

The foundation of PPO lies in the vanilla policy gradient objective, which aims to optimize the policy by maximizing the expected return. This objective is defined as:

$$L^{\mathrm{PG}}(\theta) = \mathbb{E}_t \left[\log \pi_\theta(a_t|s_t) A_t \right], \tag{21}$$

where:

- $\pi_\theta(a_t|s_t)$ is the policy parameterized by θ,
- A_t is the advantage estimate.

This objective intuitively increases the likelihood of actions that result in higher advantages A_t, reinforcing behaviours that yield better outcomes. However, large updates to the policy based on high advantage values can destabilise training, motivating the introduction of mechanisms like clipping in PPO to ensure stability.

Clipped Surrogate Objective

The clipped surrogate objective $L_t^{\mathrm{CLIP}}(\theta)$ builds on the vanilla policy gradient by introducing a trust region to limit the magnitude of policy updates. It is defined as:

$$L_t^{\mathrm{CLIP}}(\theta) = \mathbb{E}_t \left[\min \left(r_t(\theta) A_t, \mathrm{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon) A_t \right) \right], \tag{22}$$

where:

- $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\mathrm{old}}}(a_t|s_t)}$ is the probability ratio between the new and old policies,
- ϵ is a small hyperparameter defining the clipping range,
- A_t is the advantage estimate.

This clipping mechanism constrains $r_t(\theta)$ to the range $[1-\epsilon, 1+\epsilon]$, effectively preventing excessively large policy updates that could lead to catastrophic forgetting or instability. This modification ensures updates remain within stable ranges while preserving the performance improvements of policy gradient methods.

Gradient: The gradient of $L_t^{\mathrm{CLIP}}(\theta)$ is given by:

$$\nabla_\theta L_t^{\mathrm{CLIP}}(\theta) = \mathbb{E}_t \left[\nabla_\theta \min \left(r_t(\theta) A_t, \mathrm{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon) A_t \right) \right]. \tag{23}$$

Value Function Loss

The value function, parameterized by the critic network, is optimized to minimize the error between the predicted value $V_\theta(s_t)$ and the actual discounted return G_t. The objective is defined as:

$$L_t^{\mathrm{VF}}(\theta) = \left(V_\theta(s_t) - G_t\right)^2, \tag{24}$$

where:

- $V_\theta(s_t)$ is the critic's estimate of the state value,
- G_t is the discounted return, calculated as $G_t = r_t + \gamma G_{t+1}$.

Gradient: The gradient of $L_t^{\mathrm{VF}}(\theta)$ is:

$$\nabla_\theta L_t^{\mathrm{VF}}(\theta) = \mathbb{E}_t\left[2\left(V_\theta(s_t) - G_t\right)\nabla_\theta V_\theta(s_t)\right]. \tag{25}$$

The PPO objective shares similarities with related policy-gradient algorithms such as DDPG and SAC, with the notable distinction that the TD-target (referred to as y_t in 1 and 14) in PPO is directly computed as the discounted sum of rewards, without relying on the Q-function 2 or value function 15 for approximation. This difference reflects PPO's on-policy nature, where the TD-target is recalculated after each policy update, unlike the static targets used in off-policy algorithms like DDPG and SAC.

Entropy Regularization

The entropy term $S[\pi_\theta](s_t)$ encourages exploration by penalizing overly deterministic policies. It is defined as:

$$S[\pi_\theta](s_t) = -\sum_a \pi_\theta(a|s_t)\log\pi_\theta(a|s_t). \tag{26}$$

Higher entropy indicates greater stochasticity in the policy, preventing premature convergence to suboptimal strategies and promoting exploration.

Gradient: The gradient of the entropy term is:

$$\nabla_\theta S[\pi_\theta](s_t) = -\mathbb{E}_t\left[\nabla_\theta \sum_a \pi_\theta(a|s_t)\log\pi_\theta(a|s_t)\right]. \tag{27}$$

This gradient encourages the policy to maintain a diverse set of actions, balancing exploration and exploitation.

Alternative KL Divergence-Based Objective

An alternative formulation uses the KL divergence between the new and old policies to enforce a trust region:

$$L_t^{\mathrm{KL}}(\theta) = D_{\mathrm{KL}}(\pi_\theta || \pi_{\theta_{\mathrm{old}}}), \tag{28}$$

where D_{KL} is the Kullback-Leibler divergence. While this approach provides theoretical guarantees, empirical studies have shown that it performs worse than the clipping-based objective [Schulman et al., 2017b] and to decrease learning efficiency attributable to its asymmetry $D_{\mathrm{KL}}(\pi_\theta \| \pi_{\theta_{\mathrm{old}}}) \neq D_{\mathrm{KL}}(\pi_{\theta_{\mathrm{old}}} \| \pi_\theta)$ [Guo et al., 2024].

Rationale

PPO was included as the final candidate algorithm due to its effective prior utilisation in both dynamic path planning [Qi et al., 2022] and waypoint navigation tasks [Joshi et al., 2023] in UAVs, indicating its balance between simplicity of implementation, stability, and efficiency makes it a viable alternative for waypoint navigation.

3 Prior Work

3.1 Observation Space Design

The observation space is essential for waypoint navigation, as it contains the required relative information for the UAV to travel efficiently towards a waypoint while being sufficiently abstract to generaliseto unfamiliar situations. The observation space design must be tailored to provide an accurate representation of the UAV's orientation, movement, and relationship with the target. In [Li and Wu, 2020], a two-dimensional environment is presented, bound to a 500-meter square plane, with randomly placed stationary obstacles. The UAV is given the task to navigate to an assigned waypoint by utilising relative location information about the target and sensor-based obstacle detection.

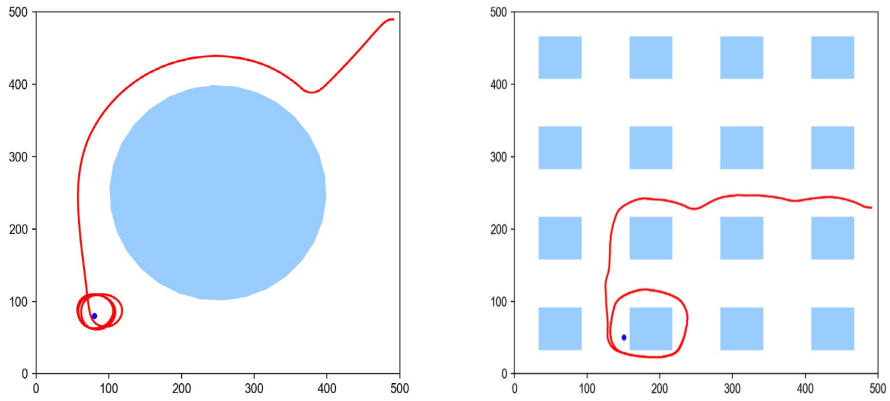


Figure 1: Examplary environments. Obstacles are represented in light blue, the target in dark blue and the trajectory flown by the UAV in red.

The observation space covers the UAV's velocity direction relative to the global x-axis, indicated by the angle ψ, and additionally the relative azimuth angle, measured from the global x-axis, along with its current distance from the target, represented by $[\chi, d_0]$. An illustration is offered in 2.

The state space is designed to be normalized, ensuring that all components lie within predefined bounds. For example, angular components such as ψ and χ are normalized to $[-\pi, \pi]$, while distances such as d_0 and d_i are scaled to $[0, 1]$. This normalization enhances the stability and generalisability of the reinforcement learning algorithm by preventing large-scale variations in the input values. [Shao et al., 2020]

In the state-space design proposed in [Himanshu and Pushpangathan, 2022], the observa-

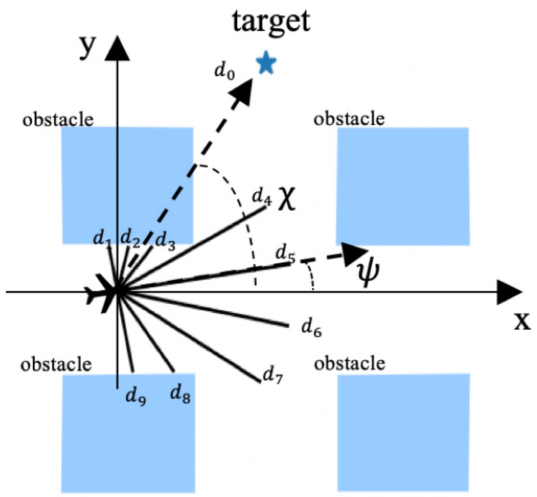


Figure 2: Illustration of what χ and ψ represent inside the state-space

tion space vector **S** provided as input to the actor network is defined as:

$$\mathbf{S} = \langle d_x, d_y, d_z, \omega_x, \omega_y, \omega_z \rangle, \tag{29}$$

where $\langle dx, dy, dz \rangle$ denotes the difference between the quadrotor's present position and the designated waypoint position in the x, y, and z axes (measured in meters). The angular velocities $\langle \omega_x, \omega_y, \omega_z \rangle$ (expressed in radians per second) convey information into the rotational dynamics of the quadrotor about the x, y, and z axes. According to the control algorithm utilised, the paper suggests adding $\langle v_x, v_y, v_z \rangle$ to **S** to account for the UAV's current heading. The incorporation of angular velocities would enhance the algorithm's ability to comprehend the influence of control inputs on rotational dynamics, translational speed, and the overall heading of the UAV.

In [Bouhamed et al., 2020], the observation space consists solely of the UAV's 3D Cartesian coordinates, (x_u, y_u, z_u), and the 3D coordinates of the target destination, (x_d, y_d, z_d), indicating the goal location, without including additional information regarding the UAV's velocity or angular position as suggested in prior studies. This paper does not address any normalisation necessary for a generalisable state-space representation.

3.2 Command Design

The action space, that is, the inputs enabling the agent's control of motion, demonstrates a degree of uniformity across the literature. The action space outlined in [Li and Wu, 2020] emphasises the control of the UAV's heading angle variation ($\Delta\psi$) within a two-dimensional continuous environment. The UAV's three-dimensional motion model is reduced to two dimensions with a fixed velocity v, establishing the yaw rate (ψ) as the primary control variable for navigation.
Actions refer to incremental adjustments to the UAV's yaw angle at each timestep, limited to a practical range. To ensure feasibility in real-world scenarios, the relationship between yaw variations and the physical acceleration constraints of UAVs is defined as follows:

$$a' = \frac{2V \tan\left(\frac{\Delta\psi}{2}\right)}{\Delta t},$$

where a' represents the normal acceleration, V denotes the UAV's velocity, $\Delta\psi$ indicates the yaw adjustment, which corresponds to the current action, and Δt signifies the sampling time interval. This equation guarantees valid yaw adjustments that remain within the physical limits of the UAV, preventing disproportionate accelerations.
In [Himanshu and Pushpangathan, 2022] and [Joshi et al., 2023], waypoint navigation for a quadrotor is achieved through direct control of its linear velocity vector. The state vector input to the high-level control actor-network includes the relative position of the quadrotor to the waypoint in the x, y, and z directions (dx, dy, dz), as well as the angular velocities about the x, y, and z axes $(\omega_x, \omega_y, \omega_z)$. The actor-network yields an action vector comprising the normalised components of linear velocity in the x, y, and z directions (v_x, v_y, v_z), along with the magnitude of the linear velocity (v_{mag}).
The normalised velocity components (v_x, v_y, v_z) and the magnitude (v_{mag}) are merged to form the actual velocity vector.

$$\vec{V} = \frac{\langle v_x, v_y, v_z \rangle}{\| \langle v_x, v_y, v_z \rangle \|} v_{mag}$$

The vector is scaled to a specified speed limit to maintain the quadrotor within its kinetic constraints, precisely to a maximum velocity of 0.25 m/s. Enforcing these limits restrains the action space to a manageable range, establishing more stable and efficient training of the controller. In [Himanshu and Pushpangathan, 2022], the velocity vector is passed on to a low-level controller, which translates the desired velocity into specific

motor commands $\langle m_1, m_2, m_3, m_4 \rangle$ via a PID control mechanism. Although not explicitly stated in [Joshi et al., 2023], it is probable that this translation is also taking place. This layered control design enables the actor-network to concentrate on high-level navigation objectives, whereas the low-level controller manages the quadrotor's physical constraints.

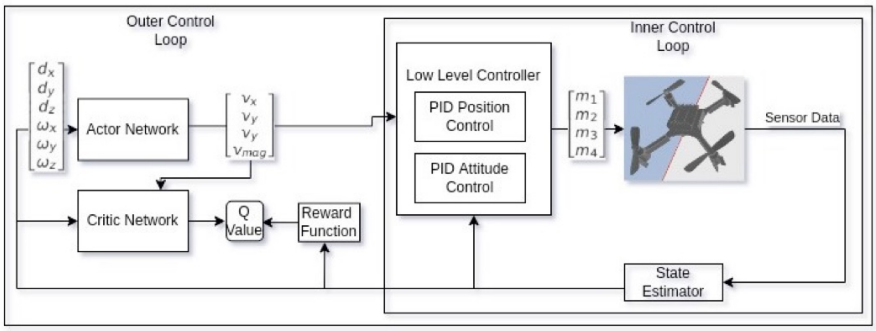


Figure 3: Illustration of the process where high-level commands from the actor network are translated into low-level motor thrusts by the PID controller within the training loop in [Himanshu and Pushpangathan, 2022].

In [Bouhamed et al., 2020], the action space of the UAV is represented through a continuous spherical coordinate system characterised by the parameters ρ (radial distance), ϕ (elevation angle), and ψ (azimuth angle). This facilitates unrestricted motion within a three-dimensional environment. At each time step Δt, the UAV's position (x_u, y_u, z_u) in Cartesian coordinates is updated as per the following equations:

$$x_u = x_u + \rho \sin\phi \cos\psi, \quad y_u = y_u + \rho \sin\phi, \quad z_u = z_u + \rho \cos\phi$$

where:

- ρ is the radial distance traveled in a single step, constrained by $\rho \in [\rho_{\min}, \rho_{\max}]$, with $\rho_{\max}$ determined by the UAV's maximum speed $v_{\max}$.
- $\phi \in [0, \pi]$ is the elevation angle, which determines vertical movement.
- $\psi \in [0, 2\pi]$ is the azimuth angle, controlling horizontal movement.

3.3 Reward Function

Establishing an effective reward function is crucial in reinforcement learning, as it acts as the main mechanism for shaping an agent's behaviour. It formalises incentives and penalties that align the agent's learning objectives with task requirements, thus guiding it towards optimal policies. In UAV waypoint navigation, the reward function aims to balance competing objectives, such as efficient target navigation and stability, while ensuring smooth flight trajectories. An effective reward function must provide the agent with consistent and interpretable feedback on its performance to promote generalisation across various situations. Poorly designed reward signals that misalign with the state space or are excessively influenced by initial conditions may hinder convergence and result in inadequate behaviour. The reward function proposed in [Himanshu and Pushpangathan, 2022] employs separate reward functions for the high-level and low-level controllers to optimize their specific objectives.
The reward function for the high-level controller is defined as:

$$R = r_0 - w_{\text{diff}}\|\langle dx, dy, dz\rangle\| - w_{\text{att}}\|\langle \phi, \theta, \psi\rangle\| - w_{\text{act}}\|A\| - w_{\text{der}}\left\|\frac{dA}{dt}\right\| \tag{30}$$

where:

- r_0: Encourages the agent to remain operational, promoting longer survival.
- $w_{\text{diff}}\|\langle dx, dy, dz\rangle\|$: Penalizes positional error, incentivizing the agent to move closer to the waypoint.
- $w_{\text{att}}\|\langle \phi, \theta, \psi\rangle\|$: Penalizes deviations in roll, pitch, and yaw to maintain attitude stability.
- $w_{\text{act}}\|A\|$: The L_n-norm of the action vector A discourages excessive action magnitudes to ensure efficient use of control inputs.
- $w_{\text{der}}\|\frac{dA}{dt}\|$: The change of the action vector over time penalizes drastic changes in actions to promote smooth transitions without rapid changes.

For the low-level controller, the reward function includes an additional term $w_{\text{vel}}\|\langle v_x, v_y, v_z\rangle\|$ to limit linear velocity. The subsequent reward function remains unchanged apart from w_{vel} being subtracted from R.
The weights w_i used for the components of the reward function for both high-level and low-level controllers are presented in Table 1.

Table 1: Reward Function Parameters

Parameter	**High-Level Controller**	**Low-Level Controller**
r_0	2.0	2.0
w_{diff}	1.0	1.0
w_{vel}	-	0.04
w_{att}	0.02	0.02
w_{act}	0.02	0.01
w_{der}	0.01	0.005

No rationale for the selected weights was provided, although it is probable that a hyperparameter tuning algorithm was utilised. The design of the reward function exhibits potential deficiencies. The incorporation of w_{der}, which penalises rapid changes in actions, fails to have a direct influence since prior actions are clearly excluded from the state space, preventing the agent from explicitly modelling the effects of sudden transitions. Furthermore, w_{diff}, which imposes penalties for positional errors, is significantly impacted by the agent's initial conditions. This results in penalties being assigned not according to the agent's behaviour, but rather based on the initial distance from the target. The aforementioned issues can distort the learning process by generating noisy reward signals that include elements beyond the agent's control or that the agent is unable to perceive. In [Li and Wu, 2020], the reward function is decomposed into several components, each designed to address specific critical aspects of UAV behaviour. The Line-of-sight (LOS) term r_{LOS} encourages the UAV's alignment with the target. The definition is as follows:

$$r_{\text{LOS}} = \lambda \left(\frac{\pi}{2} - |\chi - \psi| \right), \tag{31}$$

where λ is a positive constant and $|\chi - \psi|$ signifies the angle between the target orientation χ and the UAV's velocity vector ψ. The reward reaches its maximum when the UAV's heading coincides with the target, decreasing linearly to zero as the UAV moves perpendicularly away ($\frac{\pi}{2}$, 90 degrees). Subtracting the angle from $\frac{\pi}{2}$ guarantees positive rewards for movement towards the target and negative rewards for angular deviations greater than 90 degrees, which encourages target-directed motion. An action penalty r_{action} is introduced to promote smoother trajectories.

$$r_{\text{action}} = \alpha \|\Delta a\|, \tag{32}$$

where α indicates a negative constant, and Δa represents the change in action between subsequent time steps. This component, akin to w_{der} in [Himanshu and Pushpangathan, 2022], is intended to aid in smooth transitions between successive actions by imposing penalties for significant discrepancies, assigning negative rewards for sudden shifts, and thus discouraging erratic flight manoeuvres. The reward function is thus formulated as the sum of the individual components.

$$R = r_{\text{LOS}} + r_{\text{action}},^{3} \tag{33}$$

Scaling factors α and λ are applied to the individual components r_{LOS} and r_{action}, respectively, allowing a balanced trade-off between these terms that is consistent with the observed flight dynamics during training. Should the agent exhibit challenges in reaching the target, increasing λ could enhance target-directed behaviour. If the agent demonstrates unstable or even erratic movement, increasing α could improve the smoothness and stability of action transitions. While the paper does not explicitly define the computation of Δa, it can be inferred that it represents the difference between consecutive actions, expressed as $\Delta a = a_{t-1} - a_t$, and subsequently involves the calculation of its n-th norm. The paper does not elaborate on the specific choice of norm; however, the application of the L_2 norm ($\|\Delta a\|_2$) is beneficial due to its quadratic nature, which provides leniency for smaller deviations while imposing significant penalties on bigger ones. This characteristic permits minor deviations, ensuring that the penalty for changes in action does not disproportionately dominate or conflict with the r_{LOS} component.
The reward function utilised in [Bouhamed et al., 2020] adopts a distance-only strategy, enabling learning through the penalisation of significant distances between the agent and its target. The reward function is formulated as follows:

$$R = \exp(-5D(u,d)^2),$$

where:

- $D(u,d)$ represents the Euclidean distance between the UAV (u) and its target waypoint (d).

[3]The component r_{obstacle} and r_{terminal} in R has been purposefully omitted given its irrelavance to the topic of waypoint navigation.

Reward signals diminish exponentially with increasing distance between the UAV and its destination, thereby incentivising the UAV to approach its target. The proposed reward function has a significant limitation: it demonstrates weak gradients at large distances as a result of the rapid decay of the exponential. The gradient

$$\frac{\partial f_{\mathrm{R}}}{\partial D} = -10D(u,d)\exp(-5D(u,d)^2),$$

approaches zero as $D(u,d)$ increases, yielding minimal feedback for long-range navigation. This may restrict the UAV's capacity to develop effective strategies when initiating operations at a considerable distance from the target.

3.4 Key Takeaways

From the comprehensive literature review on waypoint navigation of UAVs, several conclusions can be drawn.

The design of the observation space must prioritise variables that accurately represent the UAV's rotational dynamics, target-relative information, and positional abstractions to enhance generalisation alongside efficiency. Angular positions represented as $\langle \psi, \chi \rangle$ and angular velocities denoted by $\langle \omega_x, \omega_y, \omega_z \rangle$ are crucial for conveying rotational information [Himanshu and Pushpangathan, 2022].

Furthermore, integrating line-of-sight (LOS) information [Li and Wu, 2020], including the relative azimuth angle χ and the angle between the UAV's current velocity vector and the LOS vector to the target, is critical for achieving target-relative navigation. This approach is beneficial as it maintains a natural value range, thus eliminating the necessity for artificial scaling.

Absolute positional information, disconnected from the target and can limit generalisation, and is therefore intentionally excluded.

The normalised LOS vector $\langle d_x, d_y, d_z \rangle$ is redundant, as its representation is adequately represented within the LOS angle already present in the observation space. Large action transitions between consecutive steps should be penalised to encourage smooth transitions. [Li and Wu, 2020] [Himanshu and Pushpangathan, 2022]

Furthermore, all elements incorporated into the reward function must align with observable variables within the state space, facilitating the agent's ability to learn from and respond to them properly.

The normalisation of components is essential, as it ensures that all terms are scaled to

established ranges, thereby avoiding unintentional overemphasis on specific aspects and underemphasis on others. Distance-based reward signals, such as those relying on Euclidean distance, can present challenges as they are prone to the influence of initial conditions, resulting in distorted learning dynamics. Incorporating line-of-sight (LOS) angle deviations [Li and Wu, 2020] provides a more robust alternative, as it prevents this inherent bias. Sparse terminal rewards [Li and Wu, 2020] should indicate critical events, including successful navigation and collisions, thus providing explicit guidance for the agent. Reward shaping techniques can effectively address the problem of disproportionately small gradients, as highlighted in [Bouhamed et al., 2020].

4 Methodology and Experimental Design

4.1 Simulation Environment

The PyFlyt package is a Python simulation framework designed for the development and assessment of reinforcement learning algorithms, especially within UAV research. Constructed using the open-source Bullet physics engine, which is supported by a substantial community and comprehensive documentation [Coumans and contributors, 2025]. PyFlyt provides a modular architecture that allows the simulation of diverse UAV configurations, that includes quadcopters, fixed-wing aircraft, and hybrid designs.

PyFlyt is particularly suitable from a technical perspective due to its native integration with the Gymnasium API, which promotes compatibility with prevalent reinforcement learning libraries like stable-baselines3. PyFlyt offers predefined environments designed for particular UAV tasks, illustrated by the QuadX-Hover-v2 environment [Tai and contributors, 2025b] for hovering control. These environments are designed to allow intuitive modifications of state and action spaces to meet the needs of particular problem domains. The library's modular design encourages the definition of custom UAV models through components such as motors, sensors, and aerodynamic surfaces. Additionally, PyFlyt's incorporation of wind field modelling enhances simulation realism by introducing environmental disturbances, which are essential for evaluating the robustness of reinforcement learning algorithms in non-ideal conditions and for exploring effective sim-to-real transfer. [Tai and contributors, 2025a]

PyFlyt, despite its features, has certain limitations. This is primarily due to the utilisation of the Bullet Engine, which displays a comparatively low real-time factor (RTF), possibly hindering the scalability of training efforts [Erez et al., 2015]. PyBullet exhibits limitations in simulation accuracy relative to alternative engines such as MuJoCo or Gazebo, which raises important concerns regarding sim-to-real transferability. [Körber et al., 2021].

PyFlyt has been selected as a practical testbed for this thesis due to its ease of implementation, existing environments, and seamless integration with stable-baselines.3. PyFlyt is considered a suitable compromise between accessibility and physical accuracy, providing it a viable resource for this academic purpose. It is important to acknowledge the limitations in translating simulation results to real-world scenarios.

4.2 Environment

The environment was built utilising Gymnasium, a framework initially created by OpenAI and currently managed by the Farama Foundation, chosen for its industry-standard design and compatibility with pre-existing RL libraries such as stable-baselines3. The use of Gymnasium enables researchers to employ standardised interfaces for environment interaction, thereby significantly minimising the overhead linked to the development of custom simulations. This thesis provides an implementation that extends the "QuadXWaypointsEnv" [Tai and contributors, 2024] environment provided by PyFlyt, specifically designed for waypoint navigation. Modifications involve custom non-sparse reward functions and an enhanced observation space, eliminating Gymnasium's "Sequence" objects to maintain compatibility with stable-baselines.3. The established environment comprises essential components as assessed in section 3. Each episode models the flight of a quadrotor UAV assigned to navigate towards a randomly generated target within a confined three-dimensional environment. The UAV operates within a spherical "flight dome" of defined dimensions, maintaining its position within the established boundaries during the entire episode. At the beginning of each episode, the UAV is initialised in the same position, at the centre of the flight dome, ascending until it reaches its starting height of $0.1 \times$ flight dome size. The UAV's orientation is adjusted to face the target, which is randomly positioned within the flight dome. The conditions for the target's position are that $\mathbf{p}_{\text{target}}\|$ must be less than or equal to the flight dome size, and z_{target} must be greater than or equal to 0.6 times the flight dome size. At each time step, the UAV is iteratively controlled by receiving an action from the agent, which subsequently alters the UAV's dynamics, influencing its position, orientation, and velocity. The physics engine updates the UAV's state, simulating realistic physical behaviour. The episode concludes, and the environment resets when the agent reaches the target within a specified margin or another terminating state is achieved.

State Space

The state of the UAV is represented by a vector $\mathbf{s}$ that encapsulates both angular and positional information:

$$\boldsymbol{S} = \langle \theta_{\text{az}}, \theta_{\text{el}}, p, q, r, \dot{p}, \dot{q}, \dot{r}, z \rangle$$

- **Angular Representation:** Since the notion of including this deviation from the LOS was introduced in [Li and Wu, 2020] for a 2-dimensional scenario it has beed consequently adapted to 3 dimensions by projecting both the velocity vector and the LOS vector to the global xy- and xz-plane. The azimuth (θ_{az}) and elevation (θ_{el}) angles respectively capture the horizontal and vertical alignment of the UAV relative to the LOS to the target.

  Let $\mathbf{v}_{\text{LOS}} = [v_{\text{LOS},x}, v_{\text{LOS},y}, v_{\text{LOS},z}]$ represent the line-of-sight vector, and $\mathbf{v}_{\text{vel}} = [v_{\text{vel},x}, v_{\text{vel},y}, v_{\text{vel},z}]$ represent the velocity vector. The projections of these vectors onto the xy- and xz-planes are defined as:

$$v_{\text{LOS},i}^{(xy)} = \begin{cases} v_{\text{LOS},x}, & \text{if } i = x, \\ v_{\text{LOS},y}, & \text{if } i = y, \end{cases} \quad v_{\text{LOS},i}^{(xz)} = \begin{cases} v_{\text{LOS},x}, & \text{if } i = x, \\ v_{\text{LOS},z}, & \text{if } i = z. \end{cases}$$

$$v_{\text{vel},i}^{(xy)} = \begin{cases} v_{\text{vel},x}, & \text{if } i = x, \\ v_{\text{vel},y}, & \text{if } i = y, \end{cases} \quad v_{\text{vel},i}^{(xz)} = \begin{cases} v_{\text{vel},x}, & \text{if } i = x, \\ v_{\text{vel},z}, & \text{if } i = z. \end{cases}$$

  The azimuth and elevation angles are then defined as:

$$\theta_{\text{az}} = \arctan 2(v_{\text{LOS},y}^{(xy)}, v_{\text{LOS},x}^{(xy)}) - \arctan 2(v_{\text{vel},y}^{(xy)}, v_{\text{vel},x}^{(xy)}), \quad \theta_{\text{az}} \in [-\pi, \pi],$$

$$\theta_{\text{el}} = \arctan 2(v_{\text{LOS},z}^{(xz)}, v_{\text{LOS},x}^{(xz)}) - \arctan 2(v_{\text{vel},z}^{(xz)}, v_{\text{vel},x}^{(xz)}), \quad \theta_{\text{el}} \in [-\pi, \pi].$$

- **Rotational Dynamics:** The angular positions $\text{Œ} = [p, q, r]$ and angular velocities $\dot{\text{Œ}} = [\dot{p}, \dot{q}, \dot{r}]$ both $\in [-\pi, \pi]$ provide information about the UAV's orientation as well as rotational rates.

- **Altitude:** The altitude $z \in [0, 1]$. describes the UAV's height relative to the ground. Min-max normalization was utilized to ensure bounded values.

The state vector is normalised to ensure that all components are confined within the range $[-1, 1]$, encouraging efficient training.

Command Space

PyFlyt offers nine unique control schemes for quadcopters, catering to diverse research and application contexts. [Tai, 2025] This thesis examines two distinct approaches that

correspond with modern methodologies: (1) direct control of motor thrusts, providing low-level actuation for precise dynamic adjustments, and (2) direct control of angular positions, allowing high-level orientation adjustment to accomplish stable and goal-directed flight. The action space $\mathbf{a}$ determines the control of the UAV and is dependent on the selected flight mode. There are two primary configurations:

- **Angular Control:** The UAV is controlled by directly setting its angular position $[v_p, v_q, v_r]$ and linear velocity along the z-axis v_z. These actions are bound as following:
$$\mathbf{a} = [v_p, v_q, v_r, v_Z], \quad \mathbf{a} \in [-\pi, \pi] \times [-1, 1],$$
- Thrust Control: The agent sets thrust levels for each of the four rotors. The rotor speeds are limited to a maximum value of 0.8, though the justification for this particular limit is not provided in the documentation. The PyBullet physics engine abstracts the physical rotor speeds, managing the underlying dynamics of the UAV.
$$\mathbf{a} = [m_1, m_2, m_3, m_4], \quad \mathbf{a} \in [0, 0.8],$$

Reward Function

The reward function R is formulated to encourage alignment with the target while to ensure a smooth and stable flight. It is defined as:

$$R = w \cdot R_{\mathbf{LOS}} + (1 - w) \cdot R_{\mathbf{SMOOTH}},$$

where:

- $R_{\mathbf{LOS}}$ rewards the UAV for minimizing the angular misalignment between its velocity- and the LOS vector.
- $R_{\mathbf{SMOOTH}}$ penalizes abrupt changes in control inputs by applying the L_2 norm to the difference between control inputs and the UAV's angular position, promoting stable flight.
- w balancing the influence of both components on R, enabling efficient learning dynamics.

Termination Conditions

The episode terminates under the following terminating states, and the environment is being reset:

- **Collision:** If the UAV comes into contact with the ground the agent is penalized with a highly negative reward $R_{\text{collision}} = -100$.
- **Flight Boundary Exceeded:** When the position of the UAV surpasses the flight dome radius D. Due to insufficient information about the agent's global position within the flight dome, no penalising reward is produced
- **Unstable Orientation:** To ensure stable orientation of the UAV a threshold of the agents angular position is set to 0.6π radians. A negative reward of $R_{\text{Unstable}} = -100$ is produced.
- **Target Completion:** Upon reaching a waypoint such that:

  $$d = \|\mathbf{p}_{\text{target}} - \mathbf{p}_{\text{UAV}}\|, \quad d \leq \delta_r,$$

  where δ_r is the distance threshold for waypoint success. A highly positive reward $R_{\text{complete}} = 100$ is yielded.

5 Results and Discussion

This section outlines the experimental results assessing the performance of PPO, SAC, and DDPG in waypoint navigation tasks across multiple setups. This section begins with a summary of the hyperparameter tuning strategy utilised, subsequently presenting a quantitative analysis of the results with regard to essential metrics such as success rates, episode durations, and average reward signals, directly comparing default hyperparameter settings with tuned configurations and investigating the effects of angular versus thrust control command design. Furthermore, the impact of learning rate annealing on the overall performance of the three algorithms is analysed prior to conducting a qualitative assessment of flight trajectories. The section concludes with an in-depth evaluation study that examines algorithm-specific shortcomings, structural challenges, and limitations of the experimental setup. This systematic method aims to provide a comprehensive analysis of the strengths and weaknesses of each algorithm in relation to the waypoint navigation task.

5.1 Hyperparametertuning

The initial hyperparameter tuning strategy involved specifying a range for each hyperparameter and generating a list of potential values evenly distributed within these ranges. This exhaustive grid search approach, which evaluates all possible combinations over a limited number of training steps, proved impractical due to limited computational resources. The combinatorial explosion of permutations resulted in time requirements that were impractical for this work. To overcome these limitations, a sequential incremental tuning method was adopted, in which each hyperparameter was adjusted individually and in sequence, without a predefined order. A range of values was established for each hyperparameter to cover a diverse array of potential configurations. The chosen values originated from pre-tuned hyperparameters that have been shown to work for a wide array of standard continuous control environments. [Raffin and contributers, 2020].
Each chosen value was tested for 300,000 training steps for PPO and 400,000 training steps for SAC and DDPG. The picked duration is informed by preliminary experiments using default hyperparameters, which demonstrated that substantial increases in average reward generally occurred within this range of steps. Upon reviewing all values for a specific hyperparameter, the value that produced the greatest fraction of successfully terminating evaluation episodes was selected, and the procedure advanced to the following hyperparameter. This method does not ensure a globally optimal set of hyperparameters; however, it does however provide a practical balance between optimisation and computational feasibility given limited resources. The actor and critic networks were initialised with two hidden layers, each containing 64 neurones. This architecture aligns with the baseline established in previous studies and adheres to the default architecture of stable-baselines 3.
The reward function generates a signal within the range $[-1, 0]$ at each timestep, reflecting the agent's alignment with the LOS vector. To assess the effect of shifting the reward signal, a constant *reward shift* was introduced, modifying the range to $[-1 + \mathit{reward\_shift}, 0 + \mathit{reward\_shift}]$. This adjustment was designed to evaluate whether incorporating slightly positive rewards could enhance learning performance. Experiments were initially conducted Across tested shifts $[0, 0.25, 0.5, 0.75, 1.0]$, a reward shift of 0.75 using PPO with default hyperparameters yielded the best performance was subsequently applied to the other algorithms. Initial testing runs under PPO weighted $R_{\mathbf{LOS}}$ with 1.0 and $R_{\mathbf{SMOOTH}}$ with 0.0 and showed promising results as no instability nor erratic ma-

neuvers were observed, the weight for $R_{\mathbf{SMOOTH}}$ was consequently set to 0.0 to avoid unnecessary complexity and confusion for the agent. Ideally, each algorithm and control design would require careful tuning to balance both rewards sufficiently, but due to time constraints, a pragmatic approach was taken, prioritizing core task performance over stability concerns, for now.

The finetuned hyperparamters are summarized in Table 5). Those not mentioned in the tables are adopted by the default hyperparameters for the respective algorithm [Raffin and contributers, 2025].

Table 2: PPO

Hyperparameter	**Value**
Learning Rate (`learning_rate`)	0.0003
Batch Size (`batch_size`)	256
Discount Factor (`gamma`)	0.975
GAE Lambda (`gae_lambda`)	0.94
Entropy Coefficient (`ent_coef`)	0.009

Table 3: SAC

Hyperparameter	**Value**
Learning Rate (`learning_rate`)	0.0003
Buffer Size (`buffer_size`)	1,000,000
Batch Size (`batch_size`)	256
Action Noise (`action_noise`)	None
Entropy Coefficient (`ent_coef`)	auto

Table 4: DDPG

Hyperparameter	**Value**
Learning Rate (`learning_rate`)	0.001
Batch Size (`batch_size`)	256
Tau (`tau`)	0.005
Discount Factor (`gamma`)	0.99
Action Noise (`action_noise`)	None

Table 5: Hyperparameter configurations for PPO, SAC, and DDPG

5.2 Quantative Results

This section provides the quantitative results for the PPO, SAC, and DDPG algorithms utilising angular and thrust control strategies. The analysis examines essential performance metrics that accurately indicate the algorithms' success in waypoint navigation, such as success rates, average episode durations, and normalised rewards per timestep. A summary of these metrics is presented in 6. The selected metrics aim to provide insight into the effectiveness and efficiency of each algorithm. The results are analysed across default and optimised hyperparameter configurations as well as focussing on the impact of angular and thrust control on performance.

Metric	**Description**
$\bar{p}_{\text{best}}$	The highest fraction of successful navigation episodes across evaluation trials.
$\bar{S}_{\text{best}}$	The average step count (in thousands) at which the highest success rate was achieved.
$\bar{L}_{F_{\text{best}}}$	The average episode duration in seconds, including both successful and failed episodes.
$\bar{R}_{\text{norm}}$	The average normalized reward, calculated over successful and failed episodes, including terminal rewards.

Table 6: Summary of performance metrics used to evaluate PPO, SAC, and DDPG.

It is important to interpret the following results with caution, as the performance comparison 4 and annealing 7 are derived from the tensorboard log files populated during training. This evaluation was restricted to 100 episodes due to time constraints, which could not accurately represent the algorithms' performance over a greater number of trials. Furthermore, $\bar{L}_{F_{\text{best}}}$ reflects the mean episode length across all validation episodes, that includes both unsuccessful and successful outcomes. Likewise, $\bar{R}_{\text{norm}}$ computes the average reward across both successful and unsuccessful episodes, including terminal rewards, which may distort the interpretability of these metrics. To address these shortcomings, an additional evaluation is considered in 5.6.

It is crucial to note that the reported means and standard deviations are derived from just a handful of independent runs: three for PPO and DDPG, and six for SAC. The sample

size is not adequate for a comprehensive stochastic analysis and mainly functions as a concise representation of multiple runs, rather than providing a robust estimate.

Under angular control, both PPO and SAC show notable improvement in success rates when trained with optimised hyperparameters (h_t), resulting in an increase in $\bar{p}_{\text{best}}$ of 0.1 and 0.16, respectively. This indicates a greater adaptability to parameter tuning, which directly enhances their consistency in reaching designated targets. PPO and SAC achieve similar success rates of 0.48 and 0.44, respectively, after tuning 4a.

SAC demonstrates a notable enhancement in $\bar{p}_{\text{best}}$ with optimised hyperparameters; however, the average episode length ($\bar{L}_{F_{\text{best}}}$) nearly doubles from 18.07s to 35.48s. PPO, by contrast, exhibits a virtually constant episode duration, with a marginal increase of less than one second. This implies that SAC demonstrates greater consistency in achieving task success, albeit at the expense of time efficiency. DDPG performs significantly worse than both PPO and SAC, demonstrating a lower average $\bar{p}_{\text{best}}$ overall. The hyperparameter tuning process for DDPG presented major obstacles, primarily due to its suboptimal baseline performance with default hyperparameters. Identifying superior configurations proved challenging, as modifications to default hyperparameters resulted in minimal to no consistent improvement, which accounts for DDPG's slightly superior performance under the default hyperparameter configuration 4a. The algorithm's significant sensitivity to hyperparameter variations [Haarnoja et al., 2018] contributes to this problem, necessitating an extensive search of appropriate configurations.

For thrust control, all three algorithms were trained with the tuned hyperparameters for angular control. The effectiveness of waypoint navigation utilising thrust commands is considerably lower than that of angular control strategies, with target waypoints achieved in merely 10 and 9 percent of assessed episodes for PPO and SAC, respectively—substantially below the performance recorded with angular control. DDPG is characterised by consistently low success rates across both command designs 4b. The observed performance gap is justifiable to a certain degree, as the direct control of motor thrusts presents a considerably more complex control objective for the policy. The Angular control system simplifies the process by automatically translating angular position commands into corresponding thrust values, thereby reducing complexity.

In summary, both PPO and SAC exhibited significant enhancements with optimised hyperparameters, with mean success rates rising by 0.1 and 0.16, respectively, underscoring their capacity to effectively utilise hyperparameter optimisation for performance improve-

ment. In contrast, DDPG's unresponsiveness to tuning efforts highlights its fundamental limitations in this context, as its performance remained largely inadequate. Under thrust control, performance typically drops for PPO and SAC, while DDPG maintains consistent performance. A performance comparison of tuned versus default hyperparameters in the context of angular thrust control is presented in 4.

Performance Metrics					
Algorithms	**Hyperparams**	$\bar{\boldsymbol{p}}_{\text{best}}$	$\bar{\boldsymbol{S}}_{\text{best}}$	$\bar{\boldsymbol{L}}_{\boldsymbol{F}_{\text{best}}}$	$\bar{\boldsymbol{R}}_{\text{norm}}$
PPO	h_d	0.38 ± 0.11	273 ± 148	15.79 ± 0.54	0.55 ± 0.05
	h_t	0.48 ± 0.06	486 ± 81	16.26 ± 1.66	0.61 ± 0.07
SAC	h_d	0.28 ± 0.22	540 ± 61	18.07 ± 2.11	0.43 ± 0.09
	h_t	0.44 ± 0.26	752 ± 191	35.48 ± 20.95	0.44 ± 0.13
DDPG	h_d	0.04 ± 0.03	447 ± 306	24.0 ± 13.57	0.16 ± 0.15
	h_t	0.01 ± 0.01	280 ± 368	4.97 ± 5.32	-1.69 ± 1.4

(a) Angular Control

Performance Metrics					
Algorithms	**Hyperparams**	$\bar{\boldsymbol{p}}_{\text{best}}$	$\bar{\boldsymbol{S}}_{\text{best}}$	$\bar{\boldsymbol{L}}_{\boldsymbol{F}_{\text{best}}}$	$\bar{\boldsymbol{R}}_{\text{norm}}$
PPO	h_t	0.1 ± 0.064	447 ± 25	3.21 ± 0.05	0.49 ± 0.07
SAC	h_t	0.09 ± 0.05	620 ± 324	5.62 ± 1.47	0.34 ± 0.13
DDPG	h_t	0.03 ± 0.02	447 ± 208	5.26 ± 2.41	0.18 ± 0.07

(b) Thrust Control

Figure 4: Performance comparison for PPO, SAC, and DDPG algorithms under different control strategies: (a) angular control and (b) thrust control. All metrics are detailed in 6. Rows h_d and h_t correspond to default and tuned hyperparameters, respectively. All results were obtained using a constant learning rate α_{constant}

5.3 Annealing of Learning Rate

A study was conducted to investigate the effects of various learning rate annealing strategies on the convergence and overall learning dynamics of all three algorithms, aiming

to enhance their performance. This study was motivated by the results presented in [Bordelon et al., 2023], which highlight the effectiveness of annealing schedules in addressing plateaus encountered during value function approximation in TD-Learning algorithms. This method is especially applicable for SAC and DDPG, given that both utilise TD-Learning to revise their value estimates for states or state-action pairs **??** 2.
Although lacking theoretical support for PPO, experiments were performed to assess possible empirical benefits.
Three distinct annealing strategies were employed: linear, exponential, and cosine. The linear schedule systematically reduces the learning rate from its initial value to zero, ensuring a clear and gradual decline. The exponential schedule, utilising a decay factor of $\gamma = 0.99$, increases the reduction of the learning rate as training advances, facilitating stronger adjustments in the later phases. The cosine schedule introduces a smooth oscillatory pattern that modulates the learning rate based on a cosine curve, resulting in a gradual and periodic decline. The selected strategies were designed to analyse various reduction behaviours and their impact on overall performance. These annealing strategies were applied solely during training that employed the angular command design. Refer to A1 for the utilised schedules.
The implementation of annealing strategies in PPO was observed to negatively impact the success of target navigation, resulting in a performance decrease of approximately 0.08 for exponential annealing and 0.11 for cosine annealing, relative to the tuned hyperparameters with a constant learning rate. Despite declines in success rates, episode lengths remain stable.
In the case of SAC, linear annealing leads to a notable decrease in performance, significantly lowering success rates and normed rewards. In contrast, exponential and cosine annealing marginally decrease success rates while effectively reducing average episode lengths from the significant 35 seconds observed with constant learning rates to durations comparable to those seen with PPO. The decrease in episode length indicates that although SAC's performance during annealing does not reach its maximum compared to constant learning rates, the strategies raise the efficiency of the navigation policy.

In the context of DDPG, both linear and exponential annealing do not produce successful navigation outcomes in the evaluated episodes. Nonetheless, a minor enhancement is evident with cosine annealing, which, despite being modest, represents the sole

condition under which DDPG achieves any degree of navigational success. This outcome suggests that the smoother transition in learning rates offered by cosine annealing is somewhat more compatible with DDPG.

In summary, while learning rate annealing has demonstrated potential in prior research for improving learning dynamics and preventing learning plateaus, its advantages in this study have differed among algorithms. The implementation of annealing strategies in PPO primarily led to decreased success rates, aligning with expectations based on prior research that predominantly concentrated on TD-Learning algorithms. In the case of SAC, annealing caused a slight decrease in success rates, but it substantially decreased episode lengths. Cosine annealing for DDPG achieved the highest fraction of successful episodes among all evaluated checkpoints, which suggests a modest but significant improvement in performance. The outcomes indicate that although annealing may be beneficial its effectiveness and influence are significantly reliant upon the particular algorithm employed. The rate of decay of the learning rate influences the balance between stability and convergence speed, as noted in [Bordelon et al., 2023]. If decay occurs too rapidly, learning may stagnate prior to achieving optimal performance; conversely, if it occurs too slowly, the value function risks becoming caught in a plateau. Therefore, identifying the suitable decay exponent is essential for maintaining consistent advancement and avoiding premature convergence.
A comparison of the tuned hyperparameters and the three learning rate schedules is presented in Table 7.

5.4 Qualitative Results

This section will present a visual overview of the three algorithms, specifically illustrating their navigation to the desired waypoint. Visualising these trajectories allows for the investigation of failure patterns and structural weaknesses in waypoint generation that could give rise to navigation challenges. A comparative analysis of PPO and SAC is carried out to assess their respective capabilities in achieving navigation goals under the same conditions. This study aims to identify both the limitations of the waypoint generation process and the comparative strengths and weaknesses of the chosen algorithms.

The flight trajectories produced by PPO 5 under angular control exhibit consistent stability, characterised by smooth paths and minimal sharp turns, alongside stable velocity,

Algorithms	**Hyperparams**	$\bar{\boldsymbol{p}}_{\text{best}}$	$\bar{\boldsymbol{S}}_{\text{best}}$	$\bar{\boldsymbol{L}}_{\boldsymbol{F}_{\text{best}}}$	$\bar{\boldsymbol{R}}_{\text{norm}}$
PPO	$\boldsymbol{h_t + \alpha_{\text{constant}}}$	$\mathbf{0.48 \pm 0.06}$	$\mathbf{486 \pm 81}$	$\mathbf{16.26 \pm 1.66}$	$\mathbf{0.61 \pm 0.07}$
	$h_t + \alpha_{\text{linear}}$	0.39 ± 0.08	287 ± 74	15.86 ± 1.44	0.58 ± 0.03
	$h_t + \alpha_{\text{exp}}$	0.40 ± 0.03	380 ± 142	14.74 ± 0.38	0.58 ± 0.03
	$h_t + \alpha_{\text{cos}}$	0.37 ± 0.15	186 ± 96	16.44 ± 1.51	0.55 ± 0.11
SAC	$\boldsymbol{h_t + \alpha_{\text{constant}}}$	$\mathbf{0.44 \pm 0.26}$	$\mathbf{752 \pm 191}$	$\mathbf{35.48 \pm 20.95}$	$\mathbf{0.44 \pm 0.13}$
	$h_t + \alpha_{\text{linear}}$	0.14 ± 0.08	664 ± 349	19.61 ± 11.82	0.04 ± 0.66
	$h_t + \alpha_{\text{exp}}$	0.40 ± 0.16	904 ± 427	19.86 ± 2.72	0.46 ± 0.04
	$h_t + \alpha_{\text{cos}}$	0.38 ± 0.19	664 ± 286	19.18 ± 5.16	0.43 ± 0.09
DDPG	$h_t + \alpha_{\text{constant}}$	0.01 ± 0.01	280 ± 368	4.97 ± 5.32	-1.69 ± 1.4
	$h_t + \alpha_{\text{linear}}$	0.0 ± 0.0	-	-	-
	$h_t + \alpha_{\text{exp}}$	0.0 ± 0.0	-	-	-
	$\boldsymbol{h_t + \alpha_{\text{cos}}}$	$\mathbf{0.04 \pm 0.06}$	$\mathbf{247 \pm 321}$	$\mathbf{6.75 \pm 8.11}$	$\mathbf{-2.04 \pm 1.74}$

Table 7: Performance metrics for the PPO, SAC, and DDPG algorithms with tuned hyperparameters and various learning rate annealing strategies. All metrics are detailed in 6. The rows α_{linear}, α_{exp}, and α_{cos} indicate linear, exponential, and cosine learning rate annealing strategies, respectively, while h_t represents the performance comparision of tuned hyperparameters.

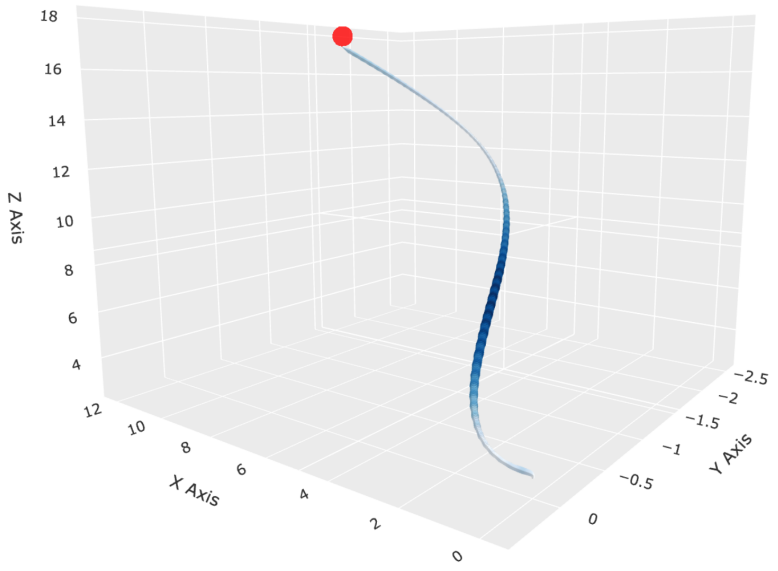
which indicates a well-regulated policy. The agent often conforms to a curved trajectory when approaching the target, rather than an optimal straight-line path 5a. A recurrent failure observed in all three algorithms, especially noticeable in the PPO policy, is that the agent approaches the target's approximate position. However, upon reaching an appropriate altitude, it abruptly changes direction and departs from the target 5b. This behaviour is speculated to result from the waypoint generation process of the environment, wherein all waypoints are located above the agent's initial altitude. As a result, the elevation angle in these scenarios predominantly positive. As the agent nears the target altitude and the elevation angle approaches zero or turns negative, the policy seems to encounter difficulties, probably due to insufficient exposure to comparable states in the training phase. A balanced waypoint generation approach would enable waypoints to be positioned below the agent's initial altitude, which would improve the robustness of the policy by expanding

the range of encountered states. Furthermore, some trajectories display the agent initially moving in the incorrect direction, failing to adjust its flight path, which frequently results in out-of-bounds failures. This issue may stem from inadequate policy convergence and could potentially be alleviated through extended training.
SAC 6 exhibits less efficient flight trajectories than PPO, often characterised by abrupt turns, sudden accelerations, and subsequent phases of relatively slow movement 6a. In certain instances, the agent may circle before realigning its trajectory towards the target. SAC exhibits comparable stability to PPO when navigating to distant waypoints, ensuring a smooth trajectory in such circumstances. However, its performance significantly drops when the target is close in the x-y plane but at a higher altitude. In such situations, the agent frequently exhibits erratic flight paths or exits the flight dome altogether 6b.
This behaviour is hypothesised to arise from the sudden transitions in the elevation angle, which is the state variable denoting the orientation of the UAV's velocity vector in relation to the line of sight vector projected onto the xz-plane. The sudden changes in state variables, especially when the agent operates below the target, seem to pose a challenge to SAC's policy. SAC exhibits challenges with stabilising its trajectory in response to rapid changes, in contrast to PPO, which adapts more effectively. This may be attributed to SAC's limited capacity to generalise across dynamic states.

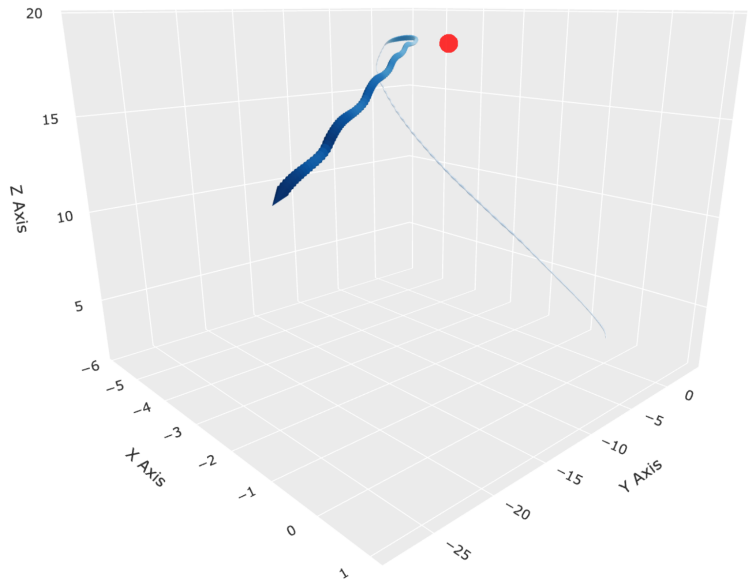
Consistent with the quantitative results presented earlier, DDPG 7 shows a lack of capability in producing efficient trajectories for navigation tasks. The flight paths generated by DDPG's policy, while occasionally achieving the target 7a, are considerably longer than those produced by PPO and SAC. This inefficiency is marked by erratic flight behaviours, such as sharp and abrupt turns, inconsistent acceleration and deceleration, and frequent deviations from optimal straight-line trajectories towards the target 7b. Like SAC, the agent exhibits repetitive circular movements, highlighting its instability and lack of control. While SAC demonstrates erratic behaviours as well, the instability of DDPG is more prominent leading to the lowest overall performance among the assessed algorithms. The observed behaviours indicate that DDPG has difficulty in learning policies that reduce unnecessary manoeuvres or enhance path efficiency.

Under thrust control, PPO 8 reveals flight envelopes akin to those observed under angular control, marked by smooth, slightly curved trajectories with minimal erratic manoeuvres and overall stable speeds 8a. Nonetheless, the policy remains ineffective in

scenarios where waypoints are located below the current altitude 8b.SAC 9 exhibits a similar pattern; however, comparable to angular control, it is marked by erratic manoeuvres, unpredictable accelerations and decelerations, and suboptimal pathfinding 9a. In contrast, DDPG 10 exhibits a notable behavioural shift under thrust control, generating highly stable and near-optimal trajectories solely for waypoints situated close to the starting position 10a. In contrast to its performance under angular control, DDPG's trajectories in this command setting closely align with the direct line-of-sight path.

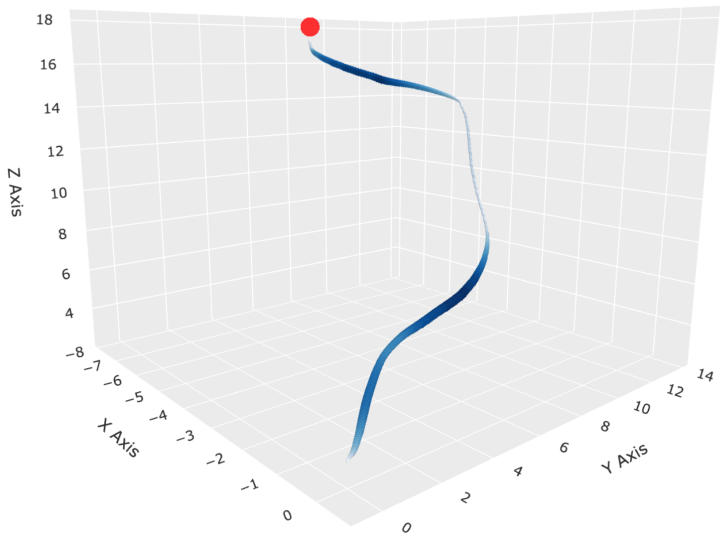


(a) A successful navigation episode under PPO.

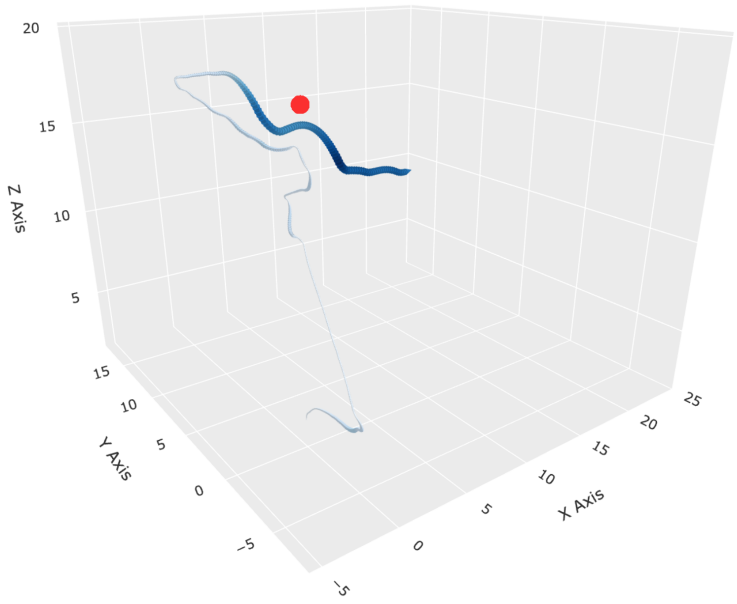


(b) A failed navigation episode under PPO.

Figure 5: Analysis of **UAV flight envelope** using a policy obtained by **PPO** and under **angular control**: (a) shows a successful navigation trajectory, and (b) illustrates a failed navigation trajectory. The red dot represents the target waypoint. Higher speeds are indicated by darker blue tones and bigger cones.

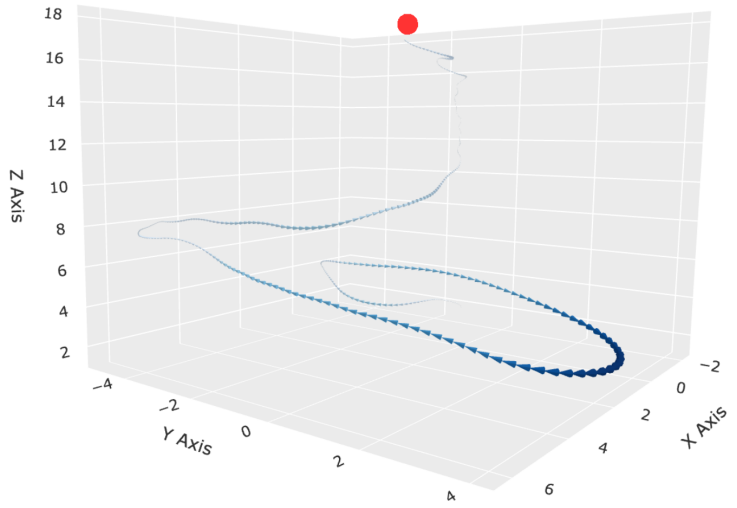


(a) A successful navigation episode under SAC.

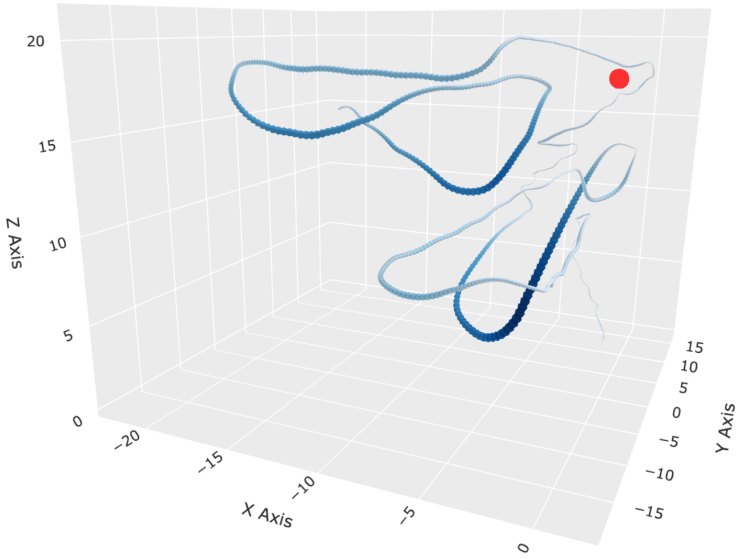


(b) A failed navigation episode under SAC.

Figure 6: Analysis of **UAV flight envelope** using a policy obtained by **SAC** and under **angular control**: (a) shows a successful navigation trajectory, and (b) illustrates a failed navigation trajectory. The red dot represents the target waypoint. Higher speeds are indicated by darker blue tones and bigger cones.

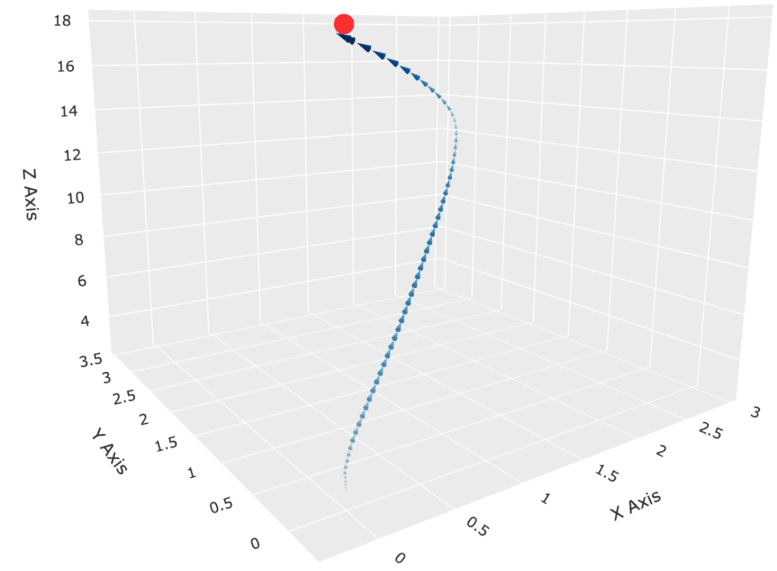


(a) A successful navigation episode under DDPG.

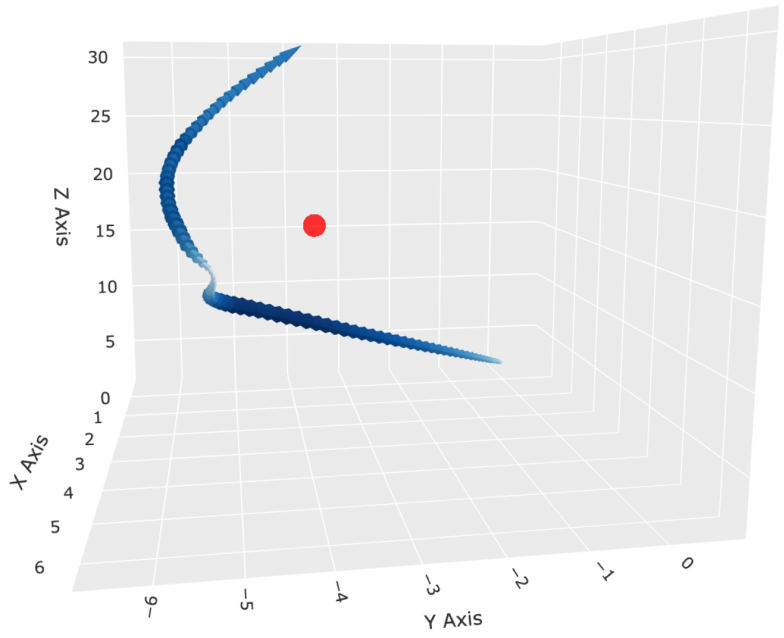


(b) A failed navigation episode under DDPG.

Figure 7: Analysis of **UAV flight envelope** using a policy obtained by **DDPG** and under **angular control**: (a) shows a successful navigation trajectory, and (b) illustrates a failed navigation trajectory. The red dot represents the target waypoint. Higher speeds are indicated by darker blue tones and bigger cones.

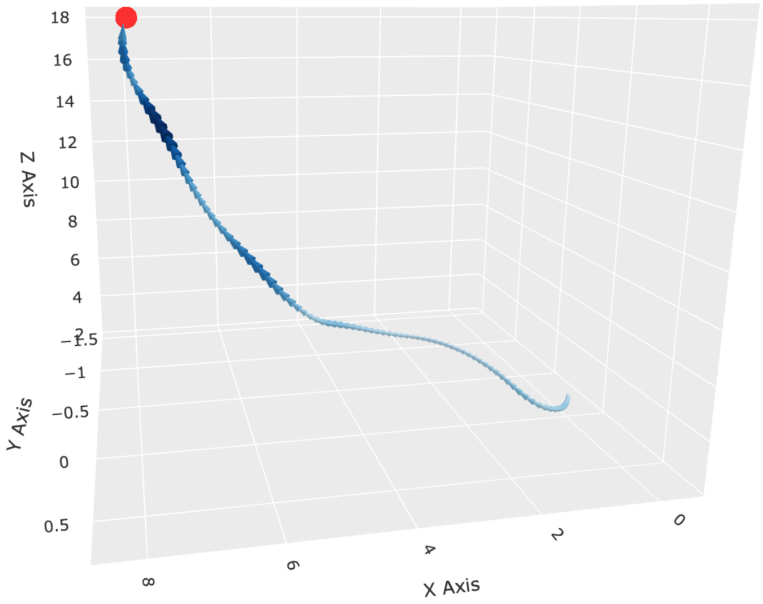


(a) A successful navigation episode under PPO.

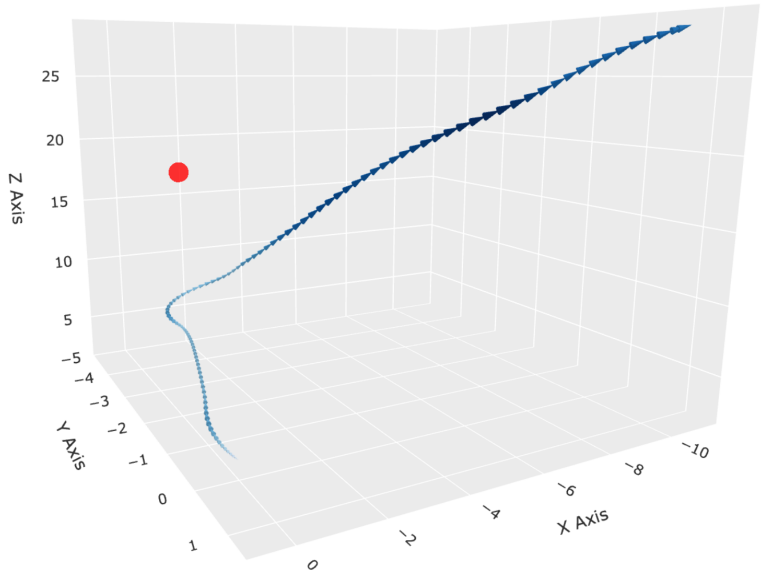


(b) A failed navigation episode under PPO.

Figure 8: Analysis of **UAV flight envelope** using a policy obtained by **PPO** and under **thrust control**: (a) shows a successful navigation trajectory, and (b) illustrates a failed navigation trajectory. The red dot represents the target waypoint. Higher speeds are indicated by darker blue tones and bigger cones.

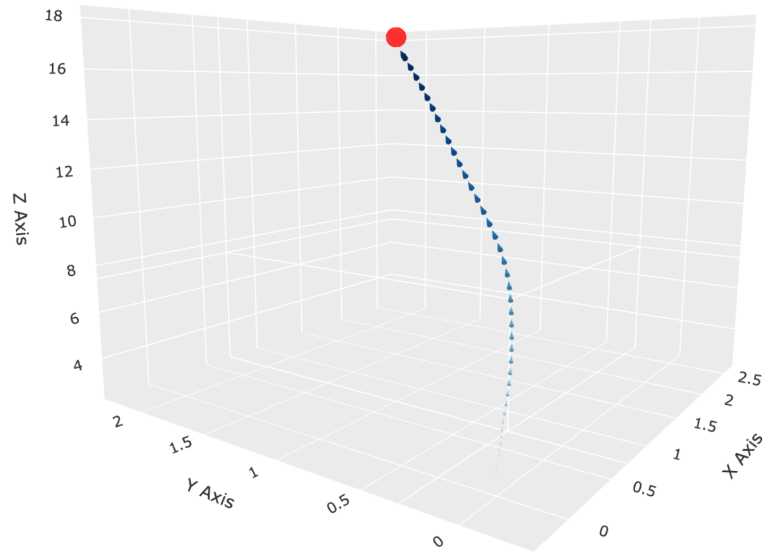


(a) A successful navigation episode under SAC.

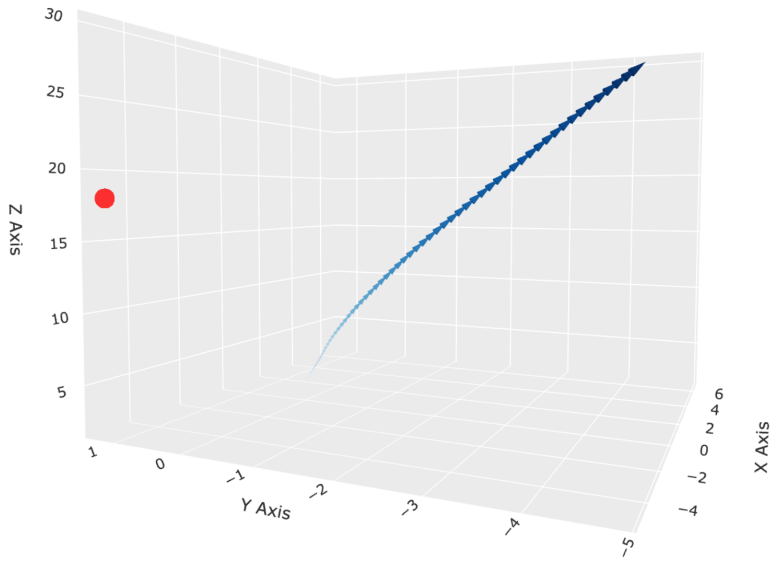


(b) A failed navigation episode under SAC.

Figure 9: Analysis of **UAV flight envelope** using a policy obtained by **SAC** and under **thrust control**: (a) shows a successful navigation trajectory, and (b) illustrates a failed navigation trajectory. The red dot represents the target waypoint. Higher speeds are indicated by darker blue tones and bigger cones.



(a) A successful navigation episode under DDPG.



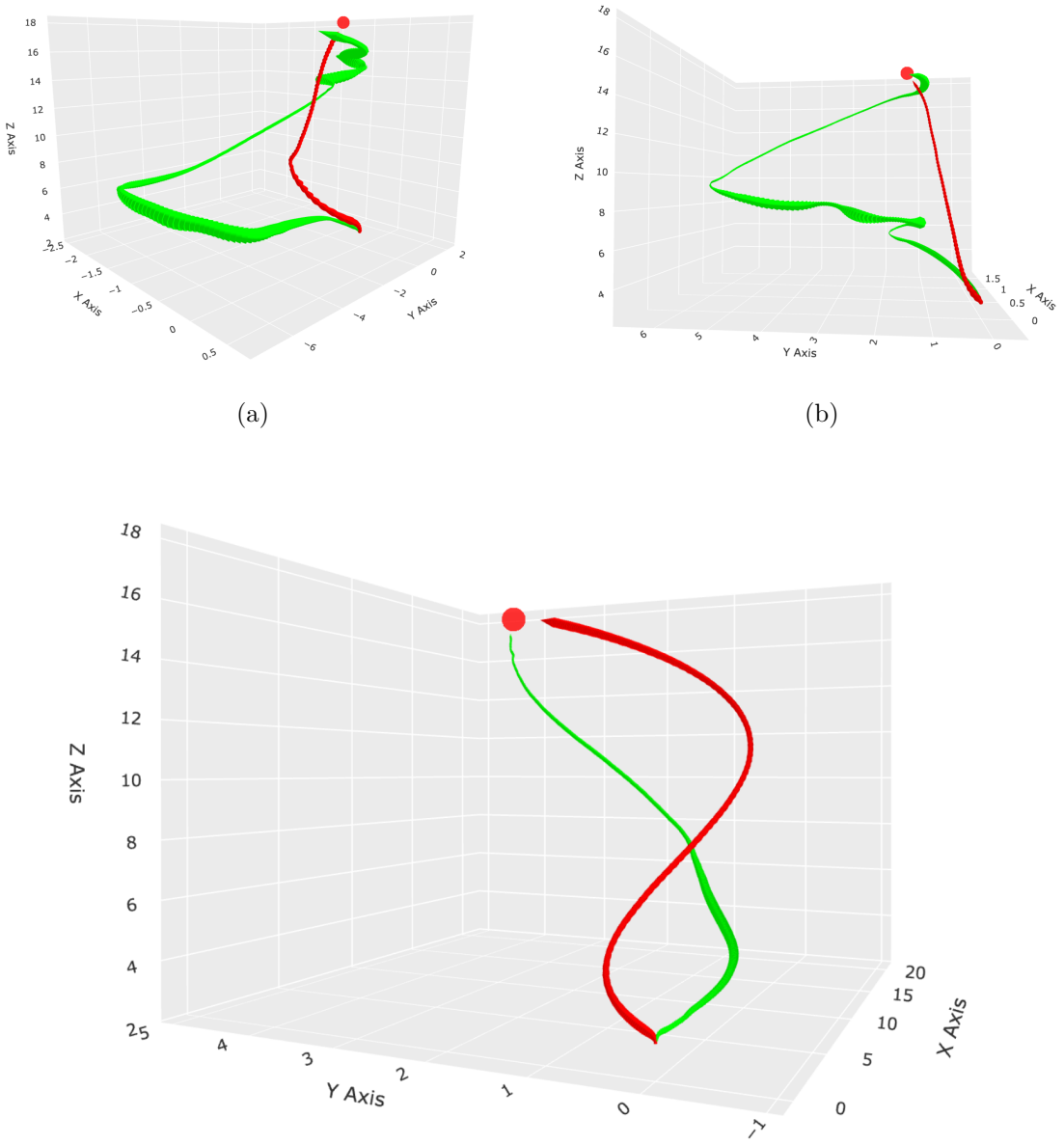
(b) A failed navigation episode under DDPG.

Figure 10: Analysis of **UAV flight envelope** using a policy obtained by **DDPG** and under **thrust control**: (a) shows a successful navigation trajectory, and (b) illustrates a failed navigation trajectory. The red dot represents the target waypoint. Higher speeds are indicated by darker blue tones and bigger cones.

5.5 Comparative Analysis

The direct comparison between PPO and SAC, which generates trajectories for identical waypoints, additionally shows that both algorithms are capable of producing smooth and consistent paths towards waypoints at greater distances, navigating with steady velocities and gentle curves 11c. When navigating to waypoints that are closely situated in the xy-plane but at raised altitudes, PPO consistently provides a more stable and direct trajectory 11a 11b. PPO's trajectory, although not as smooth as in the more distant waypoint scenario, is considerably less erratic than that of SAC, which demonstrates an inclination for sharp turns and circular paths, failing to maintain a direct trajectory towards the waypoint. In the comparison of flight envelopes for PPO and SAC under thrust control, a similar pattern emerges where PPO consistently demonstrates smoother and more direct trajectories to the waypoint than SAC 12.

DDPG was omitted from this comparison because of its comparatively low success rate in both angular and thrust control. During a 24-hour period of generating random waypoints for each control strategy, no individual waypoint was identified where all three algorithms effectively performed the navigation task. This suggests that the areas where DDPG successfully navigates do not adequately coincide with those where SAC and PPO perform effectively. This limitation on the completeness of the comparison has the benefit of reducing clutter in the figures.

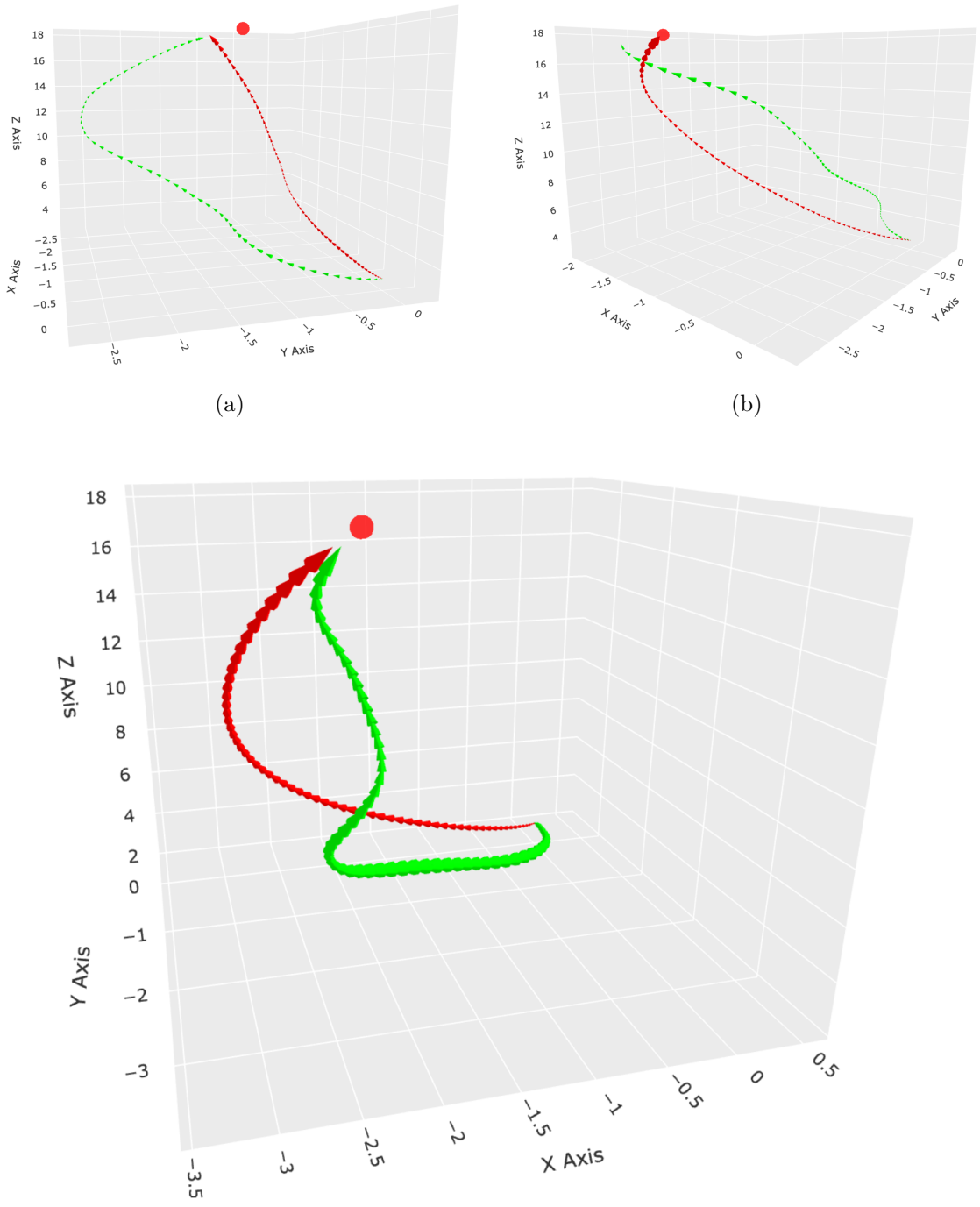


(a)

(b)

(c)

Figure 11: Direct comparison of trajectories produced from the PPO Policy (red) versus the SAC Policy (green) navigating towards the waypoint (red) under Angular-Control



(a)

(b)

(c)

Figure 12: Direct comparison of trajectories produced from the PPO Policy (red) versus the SAC Policy (green) navigating towards the waypoint (red) under Thrust-Control

5.6 Extended Evaluation and Error Analysis

To address the concerns outlined in 5.2 regarding the limited expressiveness of the utilised metrics, an additional evaluation was performed. This involved a series of navigational challenges in which each algorithm was assessed by generating $2,000$ random waypoints and measuring the agents' success in navigating to these targets. The model checkpoint that corresponds to the highest fraction of successfully reached waypoints during the training evaluation period was chosen for each algorithm. The metrics employed in this study, although akin to those in prior evaluations, have been modified to concentrate specifically on successful navigation episodes. SAC demonstrates the highest success rate among the algorithms, achieving a 0.65 fraction of successful navigation episodes, followed by PPO at 0.48 and DDPG at 0.1. SAC and PPO demonstrate comparable episode durations of $16.22s$ and $15.3s$, respectively, while DDPG's extended duration of $21.0s$ indicates a lower level of navigation efficiency. The normalised reward per step, $\bar{R}_{\text{norm}}$, was calculated excluding the terminal reward to mitigate bias, reinforces these findings. SAC and PPO exhibit values of 0.58 and 0.66, respectively, indicating effective reward strategies in successful episodes. In contrast, DDPG's score of 0.42 underscores its difficulties in attaining similar performance levels. Previous evaluations have indicated the inferior performance of thrust control relative to angular control. The results of this extended evaluation further reinforce these findings, as the success rates for thrust control are substantially lower than those for angular control. The normalised reward per step, R_{norm}, is comparable between the two control strategies, especially for SAC and PPO. Nonetheless, this apparent similarity is misleading, as the average episode length, L_{best}, for thrust control is significantly shorter. This indicates that effective navigation for thrust control is primarily restricted to waypoints in close proximity to the starting position, resulting in brief episode durations and consequently elevated per-step rewards. The findings are presented in table 8.

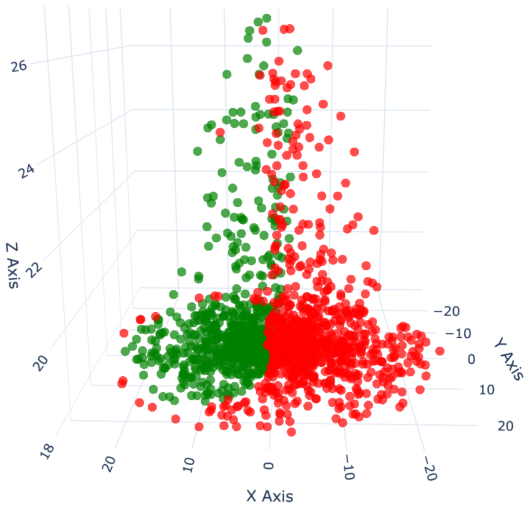
Algorithm	Angular Control (a)			Thrust Control (b)		
	$\bar{\boldsymbol{p}}_{\text{best}}$	$\boldsymbol{L}_{\text{best}}$	$\boldsymbol{R}_{\text{norm}}$	$\bar{\boldsymbol{p}}_{\text{best}}$	$\boldsymbol{L}_{\text{best}}$	$\boldsymbol{R}_{\text{norm}}$
PPO	0.5	15.4	0.66	0.07	2.2	0.6
SAC	**0.67**	**16.1**	**0.58**	0.12	3.1	0.58
DDPG	0.08	20.9	0.43	0.01	1.1	0.57

Table 8: Comparison of performance for PPO, SAC, and DDPG under angular control (a) and thrust control (b). $\bar{p}_{\text{best}}$ is the fraction of successful navigation episodes, L_{best} represents the average episode length for successful episodes, and R_{norm} corresponds to the average reward per step for successful episodes.

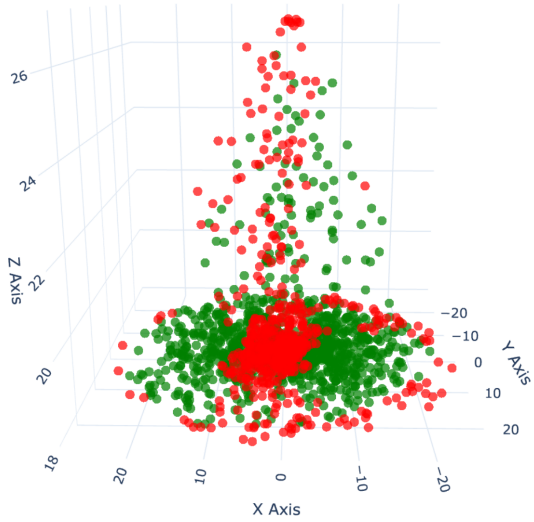
An analysis of termination flags F22 for both Thrust- and Angular-control across the $2,000$ waypoints indicates that the designed control schemes effectively prevent instability and ground collisions, thus offering substantial evidence for stable flight envelopes and safe altitude regulation. Furthermore, out-of-bounds terminations emerge as the primary failure mode, especially for DDPG across the two control schemes. The findings indicate the stability of both control designs and suggest that Angular Control is more effective in achieving task success across all three algorithms.

All $2,000$ waypoints were color-coded based on whether they resulted in a successful or failed termination and plotted to identify trends or biases in the algorithms' behaviour. This visualisation offers critical insights into the algorithms' challenges, emphasising particular shortcomings in their capacity to generalise across various waypoints. A notable observation was the clustering of waypoints at a specific altitude 13. The observed behaviour is a direct result of the waypoint generation logic utilised in PyFlyt. PyFlyt allows the establishment of minimum and maximum heights for waypoints. The generation process entails random sampling within the range of 0 to the maximum height, followed by the clipping of values that fall below the established minimum height in order to satisfy the specified minimum the limit. In the current configuration, with the minimum height set up at 60 percent of the flight dome size, around 60 percent of the waypoints are generated at the same altitude. The imbalance notably constrains the diversity of waypoints available during training, thus hindering the agents' capacity to generalise across different altitudes and resulting in lower-quality learning outcomes. This shortcoming was not

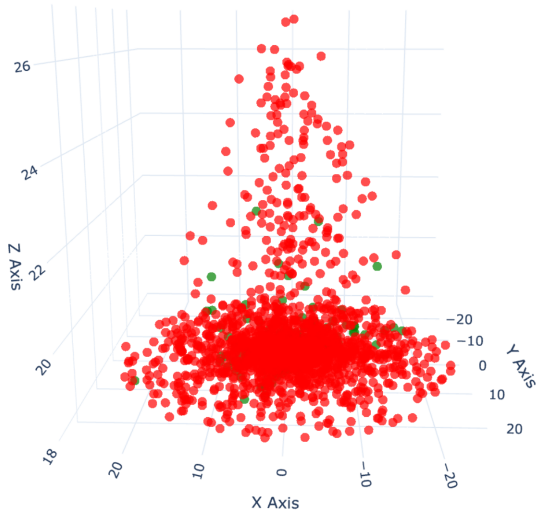
discovered until the evaluation. Shortcomings specific to the algorithm were apparent in the plotted data. PPO exhibits a distinct division at $x = 0$, with waypoints having positive x-coordinates being largely navigated successfully, whereas those with negative x-coordinates commonly experience failures 13a. The observed division indicates a structural shortcoming in the PPO's learnt policy, comparing with the SAC policy, which effectively navigated waypoints on both sides without such asymmetry. SAC, however, demonstrated a distinct pattern of errors as well. A significant percentage of failures were noted along or near the line defined by $x = 0$ 13b. Both algorithmic challenges require additional investigation to identify the root causes.



(a) Navigation Results for PPO



(b) Navigation Results for SAC



(c) Navigation Results for DDPG

Figure 13: Visualization of waypoints where the agent fails (red) and successfully navigates (green) towards the target for PPO, SAC, and DDPG under angular control.

6 Conclusion and Outlook

6.1 Summary of Findings

This thesis presents a comprehensive assessment of the performance of three distinct actor-critic algorithms: PPO, SAC, and DDPG, in UAV waypoint navigation tasks.
PPO has reliable performance, marked by stable and smooth trajectories, especially evident for waypoints situated at greater distances from the UAV. It demonstrated superior flight stability and efficient navigation under many situations. However, significant issues emerged in situations where the agent operated above the desired altitude, resulting in the failure of the policy and sudden trajectory changes. Although PPO demonstrated improved performance upon hyperparameter adjustment, the performance improvements resulting from learning rate annealing were negligible, suggesting limited advantages of this method.
SAC attained the greatest success rates across all algorithms, demonstrating significant application for waypoint navigation problems. Its trajectories were, however, somewhat less efficient than PPO, frequently characterised by more chaotic flight patterns, including abrupt turns, looping, and variable accelerations. The inefficiencies were especially apparent when navigating to proximate destinations with considerable altitude variations, where SAC encountered difficulties with swift state transitions, including sudden alterations in elevation angles. Although learning rate annealing procedures, including exponential and cosine annealing, enhanced the efficiency of SAC by considerably decreasing episode lengths, they slightly diminished its overall success rate. DDPG exhibited considerable difficulties, characterised by the lowest success rates and notably variable flight trajectories. The algorithm's policies were accompanied, akin to SAC, by significantly unpredictable behaviours. The implementation of learning rate annealing yielded marginal enhancements. The design of the control command was crucial to the algorithms' performance. Angular control frequently surpassed thrust control. This method streamlined the navigation process, yielding more fluid trajectories and enhanced overall performance. In contrast, thrust control presented a more complex learning target, resulting in substantially lower success rates. The waypoint generating procedure demonstrated a significant bias, with around 60 percent of waypoints accumulating at a particular altitude due to limitations in the sampling logic. This limitation considerably diminishes the diversity of training scenarios, resulting in inferior policies for waypoints over this elevation.

PPO encountered difficulties with waypoints on one side of the flight dome, indicating a pronounced asymmetry in its learnt strategy, whilst SAC displayed erratic behaviours for targets near $x = 0$. The fundamental rationale for these divergent failure patterns in both algorithms remains unresolved. Nonetheless, inaccuracies in state computations are improbable given each method demonstrated distinct and consistent failure patterns.

6.2 Future Work

This study builds a robust basis for UAV waypoint navigation, with a clearly defined baseline environment that exhibits adequate performance for the task. Nonetheless, numerous areas for enhancement remain, presenting potential for future research to augment the robustness, efficiency, and adaptability of the trained policies.

One approach would involve additional experimentation with the magnitude of terminal rewards, which could encourage more direct flight paths and enhance overall learning dynamics. Likewise, optimising the observation space by adding or omitting observations may enhance the agent's ability to ascertain its location in three-dimensional space. Ablation research examining how modifications to the current state-space influence performance may clarify which feature combinations yield the greatest informational value. Furthermore, incorporating noise into observations can improve resilience and increase sim-to-real transferability, an essential step in implementing learnt policies in real-world settings.

The implementation of imitation learning may offer an alternative method for achieving ideal trajectories, potentially utilising the flight mode offered by PyFlyt, wherein motor thrusts are directly controlled according to xyz-coordinates through a PID controller. This methodology may establish a foundation for agents to acquire stable control policies with greater efficiency. Additionally, investigating the restructuring of the reward function and surroundings to explicitly promote direct flight paths is a significant area for optimisation.

Extensive hyperparameter customisation of DDPG could substantially increase its performance, considering its established sensitivity to hyperparameters. Refining these settings may allow DDPG to produce competitive policies with flight envelopes akin to those of PPO and SAC. Given the computational limitations of UAV onboard CPUs, a critical inquiry is the extent to which actor-critic networks can be pruned before performance deteriorates. Methods like Neural Architecture Search (NAS) may be

employed to determine appropriate network topologies that enhance the balance between performance and computational efficiency for implementation on resource-constrained UAV hardware. Integrating a power consumption model into the reward function may be investigated to penalise excessive thrust commands, hence encouraging more energy-efficient control strategies [Chikhaoui et al., 2022]. This would offer an alternative to the R_{smooth} component by directly incentivising smooth and efficient manoeuvring, guaranteeing that the UAV optimises both trajectory adherence and energy consumption. Thrust-based approaches are noted to pose a more difficult learning target because of their reduced degree of abstraction. This heightened complexity may lead to its relatively diminished performance in this study. A thorough examination may uncover possible enhancements, such as prolonging training periods or augmenting the complexity of the actor-critic networks by increasing the number of layers and neurones.
Enhancing the waypoint-generating logic is essential for waypoint generation and initial conditions. This entails guaranteeing a more equitable distribution of waypoints throughout the whole flight dome, encompassing areas beneath the UAV's initial altitude. Integrating this with a curriculum learning strategy that identifies failure-prone areas during iterative assessments and emphasises them in subsequent training iterations may enhance policy robustness by specifically addressing regions where the policy underperforms.
Initial conditions should be diversified by randomising the UAV's direction, orientation, and angular velocity, rather than consistently commencing aligned with the target or from the ground. This diversity would expand the variety of encountered states, allowing the agent to acquire more generalised navigation techniques and preventing policy failure in unobserved situations. These enhancements collectively seek to increase the adaptability, robustness, and effectiveness of reinforcement learning-based UAV navigation systems, thereby facilitating their practical implementation in real-world contexts.

These proposed enhancements have the potential to significantly improve the current performance, addressing key limitations in stability, efficiency, and generalisation. Upon attaining robust and reliable outcomes in single-agent navigation, the subsequent phase will involve the expansion of this framework to a multi-agent reinforcement learning context, involving the coordination and interaction among multiple UAVs in more complex and non-stationary environments.

References

[Barros and Colombini, 2020] Barros, G. M. and Colombini, E. L. (2020). Using soft actor-critic for low-level uav control.

[Bordelon et al., 2023] Bordelon, B., Masset, P., Kuo, H., and Pehlevan, C. (2023). Loss dynamics of temporal difference reinforcement learning.

[Bouhamed et al., 2020] Bouhamed, O., Ghazzai, H., Besbes, H., and Massoud, Y. (2020). Autonomous uav navigation: A ddpg-based deep reinforcement learning approach.

[Chen and Yu, 2021] Chen, J. and Yu, J. (2021). An improved path planning algorithm for uav based on rrt. In *2021 4th International Conference on Advanced Electronic Materials, Computers and Software Engineering (AEMCSE)*, pages 895–898.

[Chikhaoui et al., 2022] Chikhaoui, K., Ghazzai, H., and Massoud, Y. (2022). Ppo-based reinforcement learning for uav navigation in urban environments. In *2022 IEEE 65th International Midwest Symposium on Circuits and Systems (MWSCAS)*, pages 1–4.

[Coumans and contributors, 2025] Coumans, E. and contributors (2025). Bullet physics sdk: Documentation. `https://github.com/bulletphysics/bullet3/tree/master/docs`. Accessed: 2025-01-31.

[Debnath et al., 2024] Debnath, D., Vanegas, F., Sandino, J., Hawary, A. F., and Gonzalez, F. (2024). A review of UAV path-planning algorithms and obstacle avoidance methods for remote sensing applications. *Remote Sensing*, 16(21):4019. Submission received: 19 September 2024 / Revised: 25 October 2024 / Accepted: 28 October 2024 / Published: 29 October 2024.

[Dong et al., 2022] Dong, L., He, Z., Song, C., and Sun, C. (2022). A review of mobile robot motion planning methods: from classical motion planning workflows to reinforcement learning-based architectures.

[Erez et al., 2015] Erez, T., Tassa, Y., and Todorov, E. (2015). Simulation tools for model-based robotics: Comparison of bullet, havok, mujoco, ode and physx. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4397–4404.

[Fujimoto et al., 2018a] Fujimoto, S., van Hoof, H., and Meger, D. (2018a). Addressing function approximation error in actor-critic methods.

[Fujimoto et al., 2018b] Fujimoto, S., van Hoof, H., and Meger, D. (2018b). Addressing function approximation error in actor-critic methods. *CoRR*, abs/1802.09477.

[Gundersen, 2018] Gundersen, G. (2018). The reparameterization trick. Accessed: 2025-01-02.

[Guo et al., 2024] Guo, Y., Long, H., Duan, X., Feng, K., Li, M., and Ma, X. (2024). Cim-ppo:proximal policy optimization with liu-correntropy induced metric.

[Haarnoja et al., 2018] Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. (2018). Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor.

[He and Hou, 2020] He, Q. and Hou, X. (2020). Wd3: Taming the estimation bias in deep reinforcement learning. In *2020 IEEE 32nd International Conference on Tools with Artificial Intelligence (ICTAI)*, page 391–398. IEEE.

[Himanshu and Pushpangathan, 2022] Himanshu, K. H. and Pushpangathan, J. V. (2022). Waypoint navigation of quadrotor using deep reinforcement learning. *IFAC PapersOnLine*, 55(22):281–286.

[Hollenstein et al., 2023] Hollenstein, J., Auddy, S., Saveriano, M., Renaudo, E., and Piater, J. (2023). Action noise in off-policy deep reinforcement learning: Impact on exploration and performance.

[Joshi et al., 2023] Joshi, B., Kapur, D., and Kandath, H. (2023). Sim-to-real deep reinforcement learning based obstacle avoidance for uavs under measurement uncertainty.

[Kim et al.,] Kim, S., Asadi, K., Littman, M., and Konidaris, G. Deepmellow: Removing the need for a target network in deep q-learning. *Proceedings of the Twenty Eighth International Joint Conference on Artificial Intelligence*.

[Konda and Tsitsiklis, 1999] Konda, V. and Tsitsiklis, J. (1999). Actor-critic algorithms. In Solla, S., Leen, T., and Müller, K., editors, *Advances in Neural Information Processing Systems*, volume 12. MIT Press.

[Körber et al., 2021] Körber, M., Lange, J., Rediske, S., Steinmann, S., and Glück, R. (2021). Comparing popular simulation environments in the scope of robotics and reinforcement learning.

[Lahire, 2021] Lahire, T. (2021). Actor loss of soft actor critic explained.

[Li and Wu, 2020] Li, B. and Wu, Y. (2020). Path planning for uav ground target tracking via deep reinforcement learning. *IEEE Access*, 8:29064–29074.

[Lillicrap et al., 2019] Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. (2019). Continuous control with deep reinforcement learning.

[Mandloi et al., 2021] Mandloi, D., Arya, R., and Verma, A. K. (2021). Unmanned aerial vehicle path planning based on a* algorithm and its variants in 3d environment. *International Journal of System Assurance Engineering and Management*, 12:990–1000.

[Mnih et al., 2015] Mnih, V., Kavukcuoglu, K., Silver, D., et al. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533.

[Nachum et al., 2017] Nachum, O., Norouzi, M., Xu, K., and Schuurmans, D. (2017). Bridging the gap between value and policy based reinforcement learning.

[Nikishin et al., 2022] Nikishin, E., Schwarzer, M., D'Oro, P., Bacon, P.-L., and Courville, A. (2022). The primacy bias in deep reinforcement learning. In Chaudhuri, K., Jegelka, S., Song, L., Szepesvari, C., Niu, G., and Sabato, S., editors, *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 16828–16847. PMLR.

[Oersted and Ma, 2023] Oersted, H. and Ma, Y. (2023). Review of pid controller applications for uavs.

[Papoudakis et al., 2019] Papoudakis, G., Christianos, F., Rahman, A., and Albrecht, S. V. (2019). Dealing with non-stationarity in multi-agent deep reinforcement learning.

[Plappert et al., 2017] Plappert, M., Houthooft, R., Dhariwal, P., Sidor, S., Chen, R. Y., Chen, X., Asfour, T., Abbeel, P., and Andrychowicz, M. (2017). Parameter space noise for exploration. *CoRR*, abs/1706.01905.

[Qi et al., 2022] Qi, C., Wu, C., Lei, L., Li, X., and Cong, P. (2022). Uav path planning based on the improved ppo algorithm. In *2022 Asia Conference on Advanced Robotics, Automation, and Control Engineering (ARACE)*, pages 193–199.

[Raffin and contributers, 2020] Raffin, A. and contributers (2020). Rl baselines zoo: Hyperparameters. `https://github.com/araffin/rl-baselines-zoo/tree/master/hyperparams`. Accessed: 2025-01-20.

[Raffin and contributers, 2025] Raffin, A. and contributers (2025). Stable-baselines3: Reliable reinforcement learning implementations. `https://github.com/DLR-RM/stable-baselines3/tree/master/stable_baselines3`. Accessed: 2025-01-26.

[Riedmiller et al., 2007] Riedmiller, M., Peters, J., and Schaal, S. (2007). Evaluation of policy gradient methods and variants on the cart-pole benchmark. In *2007 IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning*, pages 254–261.

[Schulman et al., 2017a] Schulman, J., Levine, S., Moritz, P., Jordan, M. I., and Abbeel, P. (2017a). Trust region policy optimization.

[Schulman et al., 2017b] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017b). Proximal policy optimization algorithms.

[Shao et al., 2020] Shao, J., Hu, K., Wang, C., Xue, X., and Raj, B. (2020). Is normalization indispensable for training deep neural networks? In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M. F., and Lin, H., editors, *Advances in Neural Information Processing Systems*, volume 33, pages 21624–21634.

[Shi, 2024] Shi, M. (2024). Application of PID control technology in unmanned aerial vehicles. *Applied and Computational Engineering*, 96:24–30.

[Suanpang and Jamjuntr, 2024] Suanpang, P. and Jamjuntr, P. (2024). Optimizing autonomous uav navigation with d* algorithm for sustainable development. *Sustainability*, 16(7867).

[Sutton and Barto, 2018] Sutton, R. S. and Barto, A. G. (2018). *Reinforcement Learning: An Introduction*. The MIT Press, second edition.

[Sutton et al., 1999] Sutton, R. S., McAllester, D. A., Singh, S. P., and Mansour, Y. (1999). Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems*, pages 1057–1063.

[Tai, 2025] Tai, J. J. (2025). Pyflyt: Quadx drone documentation. https://taijunjet.com/PyFlyt/documentation/core/drones/quadx.html. Accessed: 2025-01-31.

[Tai and contributors, 2024] Tai, J. J. and contributors (2024). Pyflyt: Quadx waypoints environment. https://github.com/jjshoots/PyFlyt/blob/master/PyFlyt/gym_envs/quadx_envs/quadx_waypoints_env.py. Accessed: 2025-01-31.

[Tai and contributors, 2025a] Tai, J. J. and contributors (2025a). Pyflyt: Core abstractions module. https://github.com/jjshoots/PyFlyt/tree/master/PyFlyt/core/abstractions. Accessed: 2025-01-31.

[Tai and contributors, 2025b] Tai, J. J. and contributors (2025b). Pyflyt: Quadx hover environment. https://github.com/jjshoots/PyFlyt/blob/master/PyFlyt/gym_envs/quadx_envs/quadx_hover_env.py. Accessed: 2025-01-31.

[Thrun and Schwartz, 1993] Thrun, S. and Schwartz, A. (1993). Issues in using function approximation for reinforcement learning. In *Proceedings of the Fourth Connectionist Models Summer School*, Hillsdale, NJ. Lawrence Erlbaum Publisher.

[Wasserman, 2023] Wasserman, L. (2023). Lecture 12a: Exponential families and generalized linear models. https://stat.cmu.edu/~larry/=stat705/Lecture12a.pdf. Accessed: 2024-12-24.

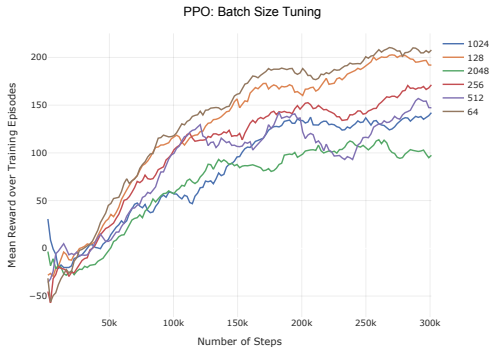
[Widmer and Kubat, 1996] Widmer, G. and Kubat, M. (1996). Learning in the presence of concept drift and hidden contexts. *Machine Learning*, 23(1):69–101.

[Wilson et al., 2022] Wilson, A. N., Kumar, A., Jha, A., and Cenkeramaddi, L. R. (2022). Embedded sensors, communication technologies, computing platforms and machine learning for uavs: A review. *IEEE Sensors Journal*, 22(3):1807–1826.

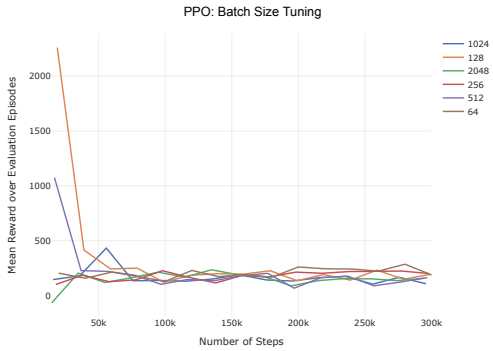
[Ziebart et al., 2008] Ziebart, B. D., Maas, A., Bagnell, J. A., and Dey, A. K. (2008). Maximum entropy inverse reinforcement learning. In *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence*, pages 1433–1438. AAAI.

7 Appendix

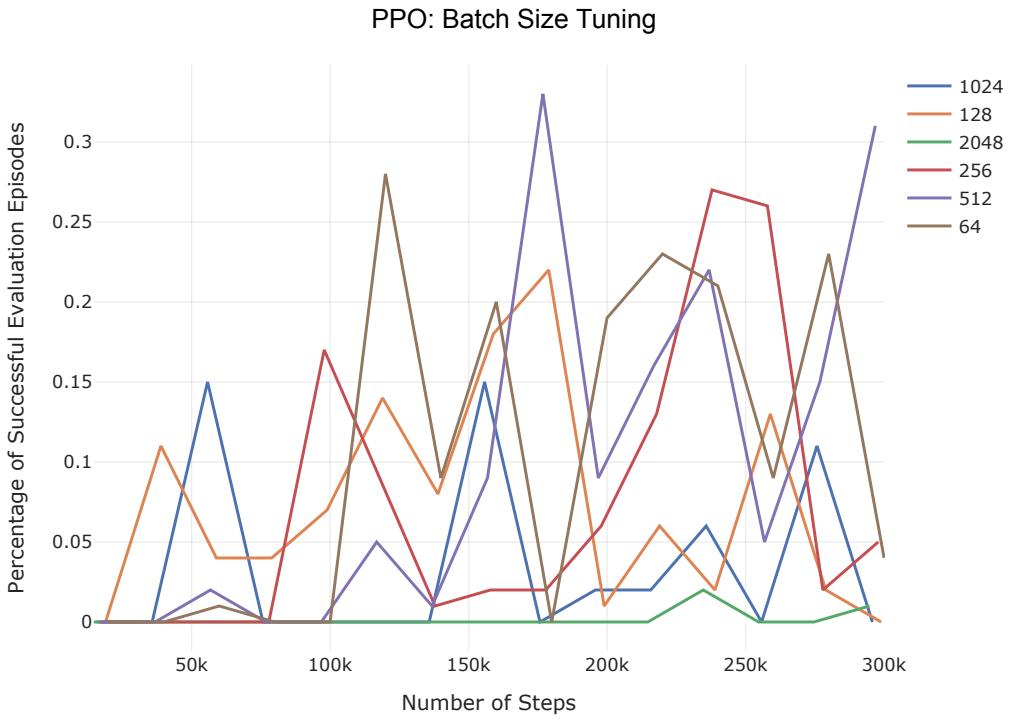
Hyperparameter Tuning: PPO



(a) Mean episodic reward during training rollouts.

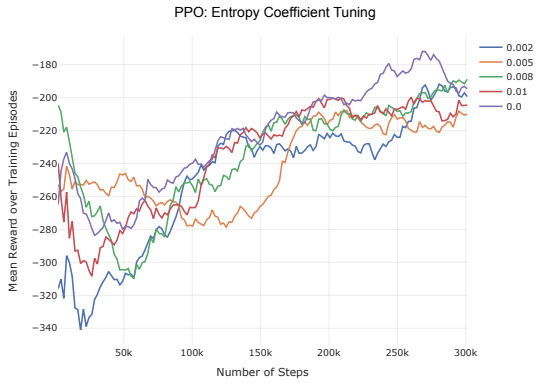


(b) Mean reward over 100 evaluation episodes.

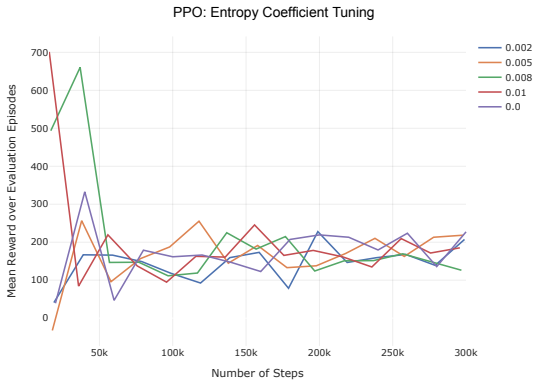


(c) Percentage of successful evaluation episodes over 100 evaluation episodes.

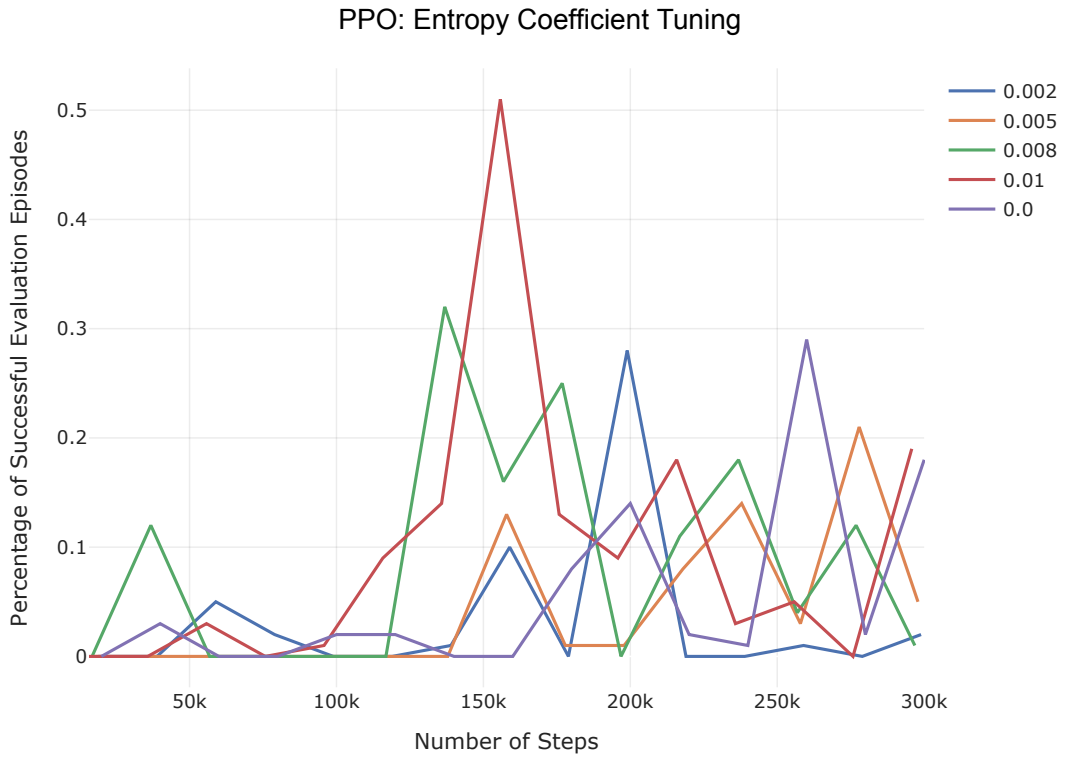
Figure F1: Batch size tuning for PPO algorithm. (a) Mean reward gathered during rollout as training progresses. (b) Mean reward gathered over 100 evaluation episodes, recorded every 20k training steps. (c) Percentage of evaluation episodes that resulted in successful navigation, recorded every 20k training steps.".



(a) Mean episodic reward during training rollouts.

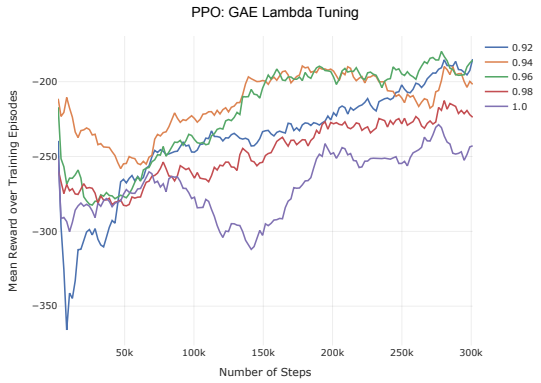


(b) Mean reward over 100 evaluation episodes.

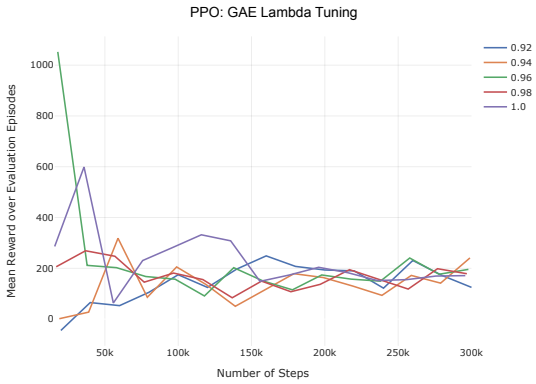


(c) Percentage of successful evaluation episodes over 100 evaluation episodes.

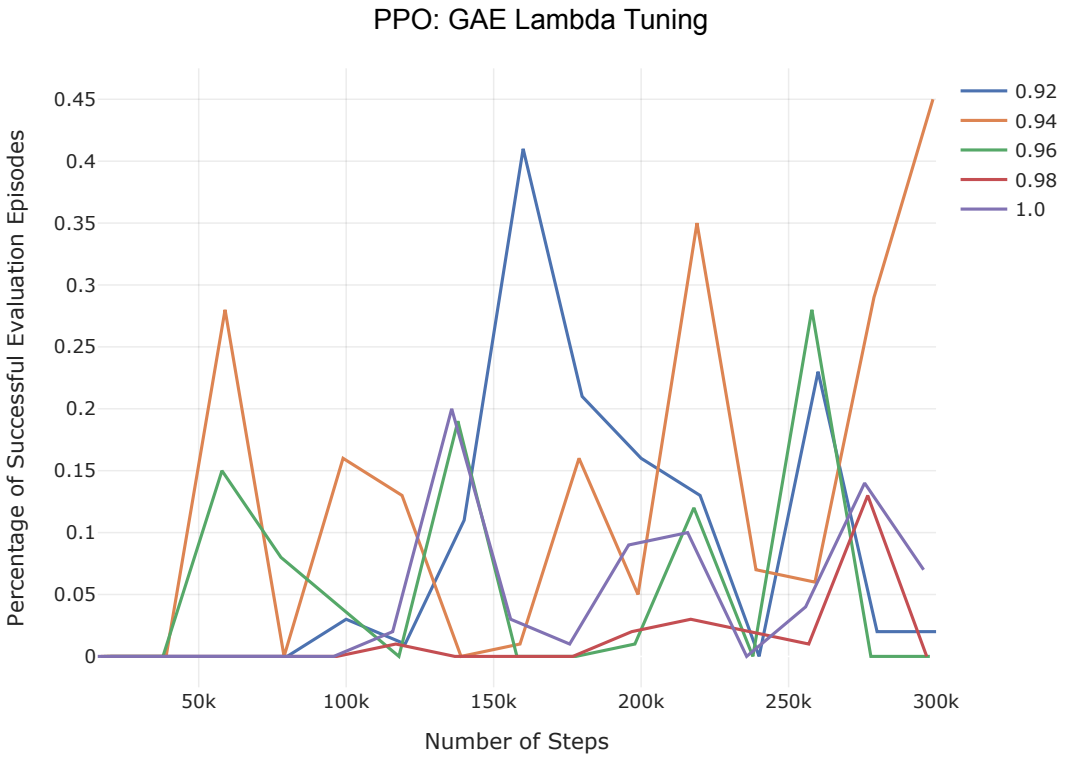
Figure F2: Entropy coefficient tuning for PPO algorithm. (a) Mean reward gathered during rollout as training progresses. (b) Mean reward gathered over 100 evaluation episodes, recorded every 20k training steps. (c) Percentage of evaluation episodes that resulted in successful navigation, recorded every 20k training steps.".



(a) Mean episodic reward during training rollouts.

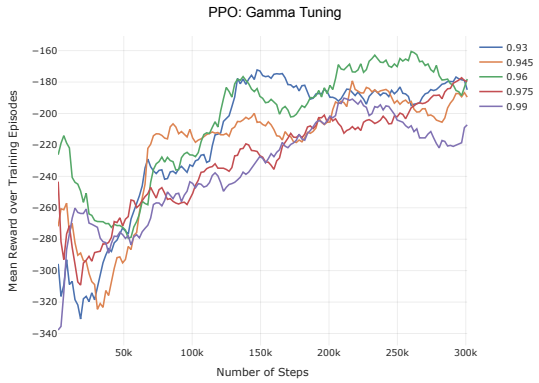


(b) Mean reward over 100 evaluation episodes.

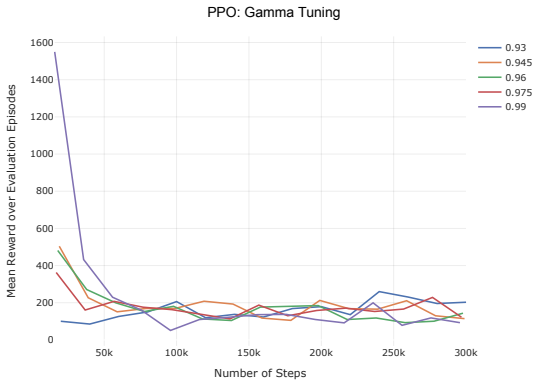


(c) Percentage of successful evaluation episodes over 100 evaluation episodes.

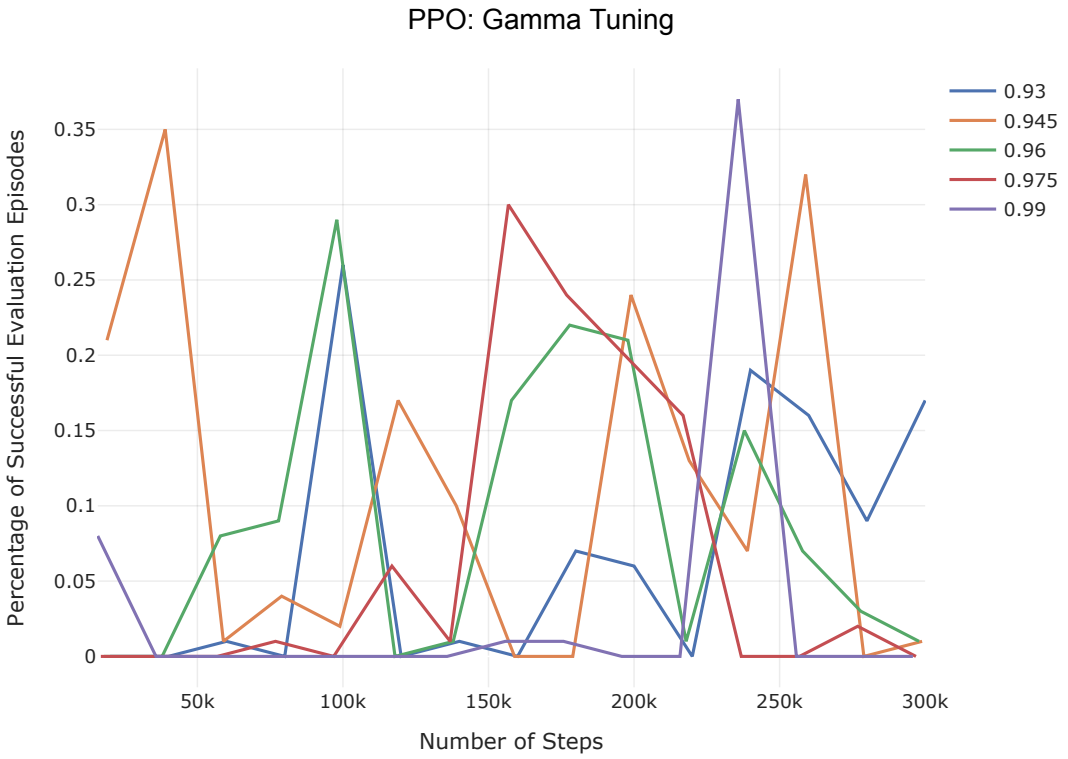
Figure F3: GAE coefficient tuning for PPO algorithm. (a) Mean reward gathered during rollout as training progresses. (b) Mean reward gathered over 100 evaluation episodes, recorded every 20k training steps. (c) Percentage of evaluation episodes that resulted in successful navigation, recorded every 20k training steps.".



(a) Mean episodic reward during training rollouts.



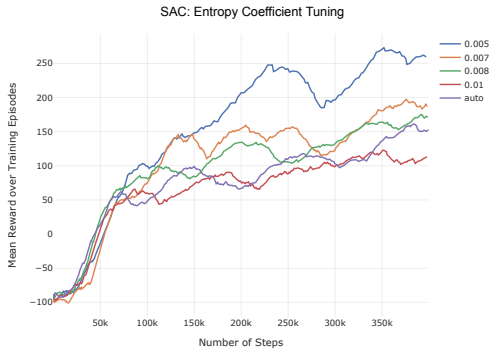
(b) Mean reward over 100 evaluation episodes.



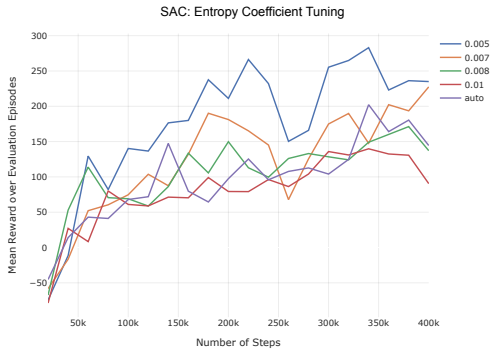
(c) Percentage of successful evaluation episodes over 100 evaluation episodes.

Figure F4: Gamma tuning for PPO algorithm. (a) Mean reward gathered during rollout as training progresses. (b) Mean reward gathered over 100 evaluation episodes, recorded every 20k training steps. (c) Percentage of evaluation episodes that resulted in successful navigation, recorded every 20k training steps.".

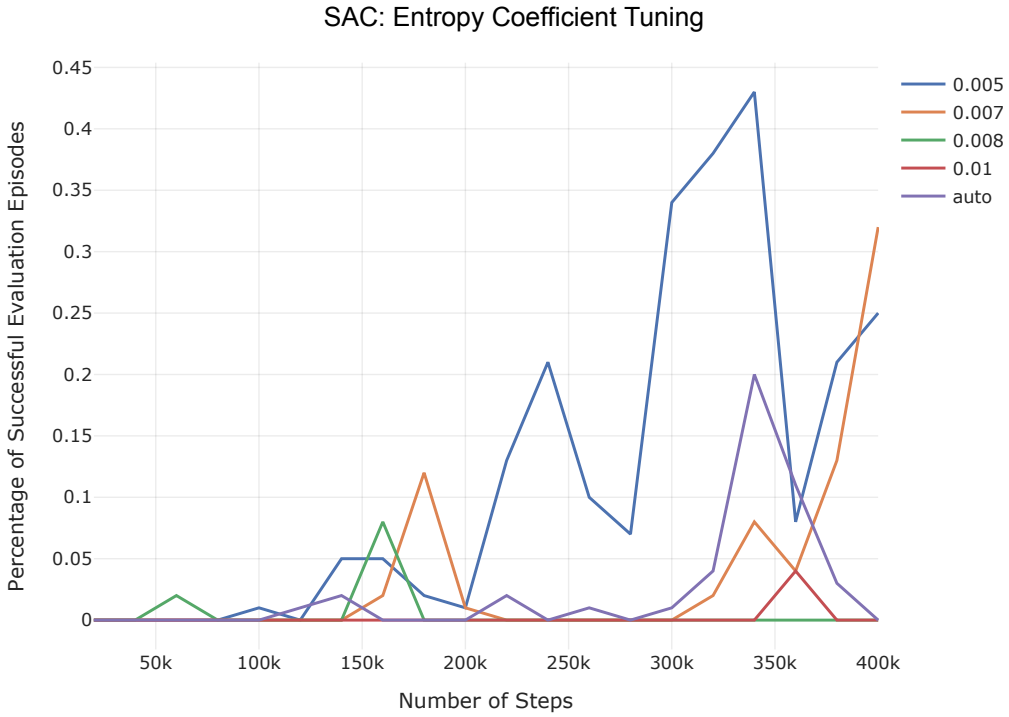
Hyperparameter Tuning: SAC



(a) Mean episodic reward during training rollouts.

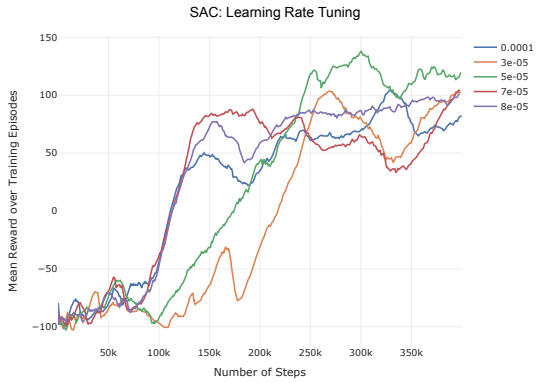


(b) Mean reward over 100 evaluation episodes.

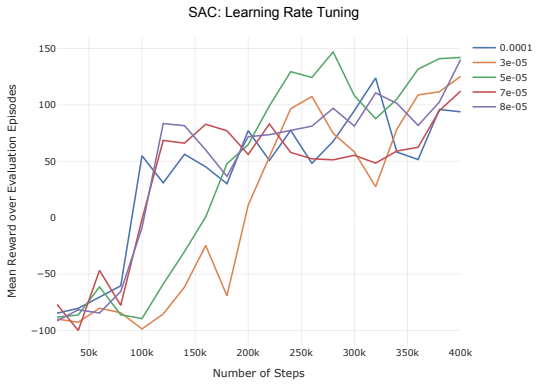


(c) Percentage of successful evaluation episodes over 100 evaluation episodes.

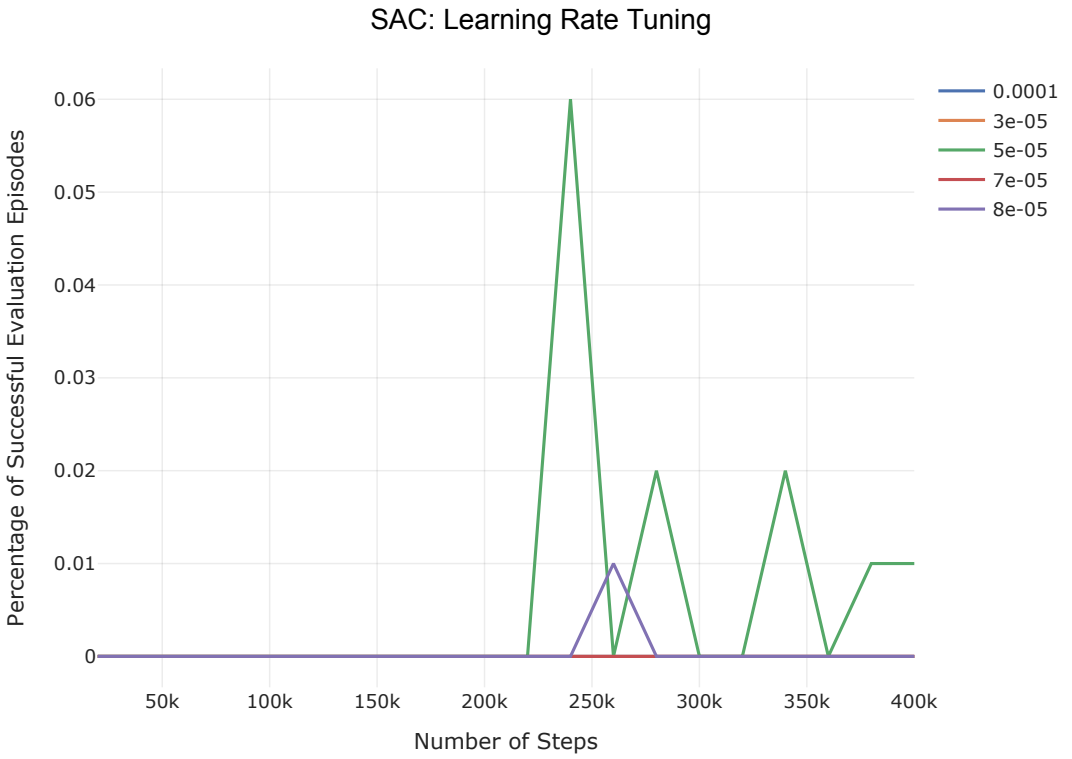
Figure F5: Entropy coefficient tuning for SAC algorithm. (a) Mean reward gathered during rollout as training progresses. (b) Mean reward gathered over 100 evaluation episodes, recorded every 20k training steps. (c) Percentage of evaluation episodes that resulted in successful navigation, recorded every 20k training steps.".



(a) Mean episodic reward during training roll-outs.

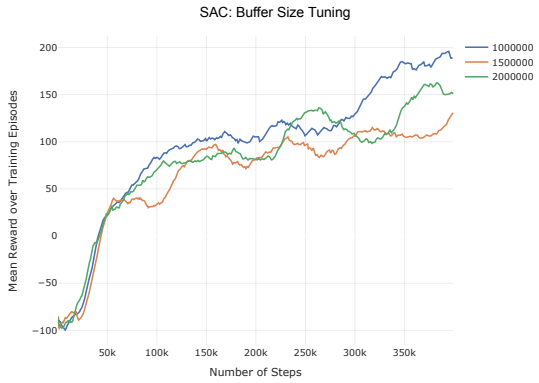


(b) Mean reward over 100 evaluation episodes.

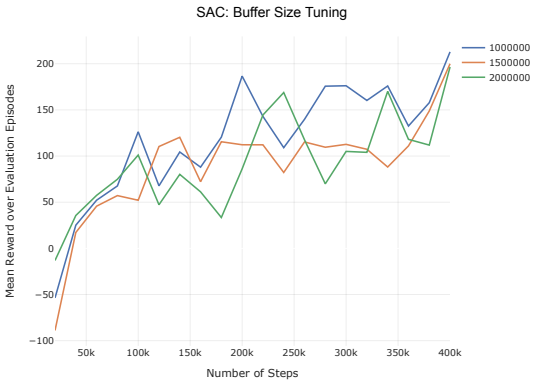


(c) Percentage of successful evaluation episodes over 100 evaluation episodes.

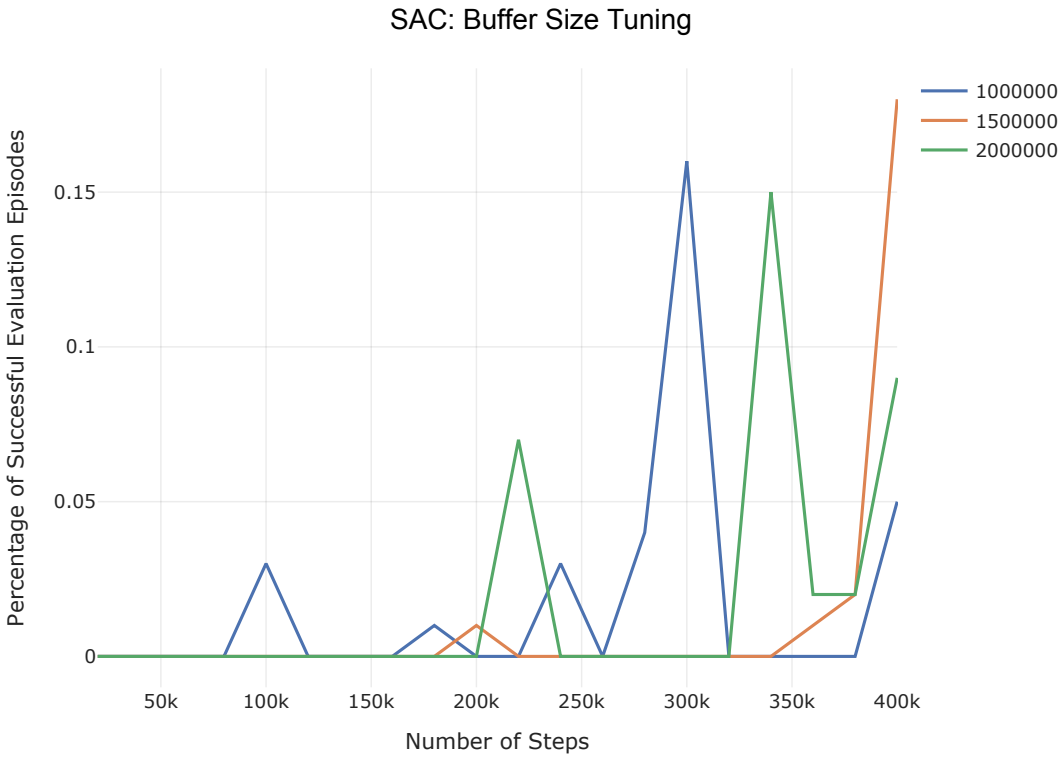
Figure F6: Learning rate tuning for SAC algorithm. (a) Mean reward gathered during rollout as training progresses. (b) Mean reward gathered over 100 evaluation episodes, recorded every 20k training steps. (c) Percentage of evaluation episodes that resulted in successful navigation, recorded every 20k training steps.".



(a) Mean episodic reward during training roll-outs.

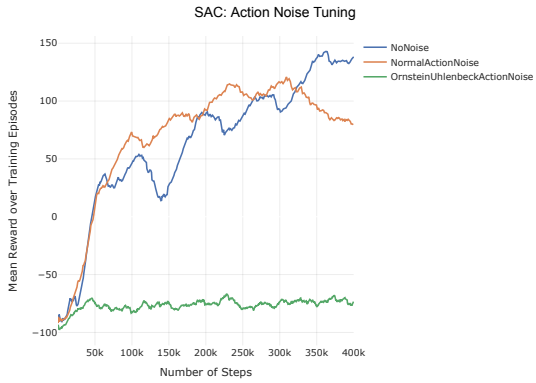


(b) Mean reward over 100 evaluation episodes.

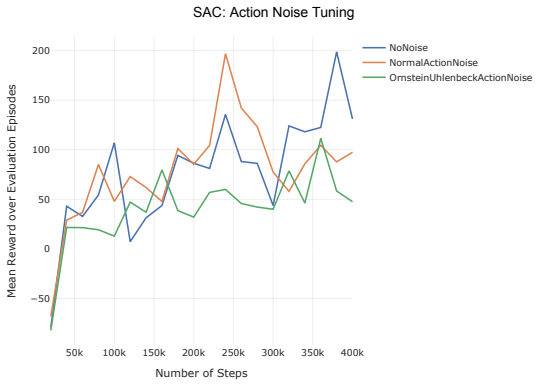


(c) Percentage of successful evaluation episodes over 100 evaluation episodes.

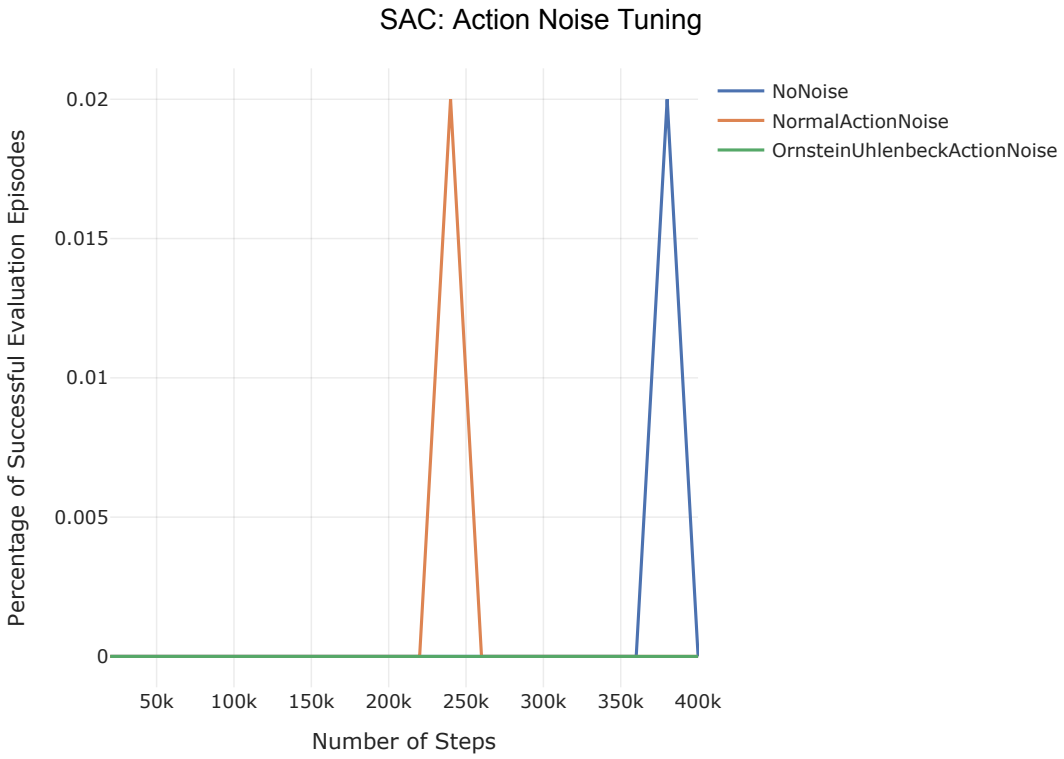
Figure F7: Buffer size tuning for SAC algorithm. (a) Mean reward gathered during rollout as training progresses. (b) Mean reward gathered over 100 evaluation episodes, recorded every 20k training steps. (c) Percentage of evaluation episodes that resulted in successful navigation, recorded every 20k training steps.".



(a) Mean episodic reward during training rollouts.

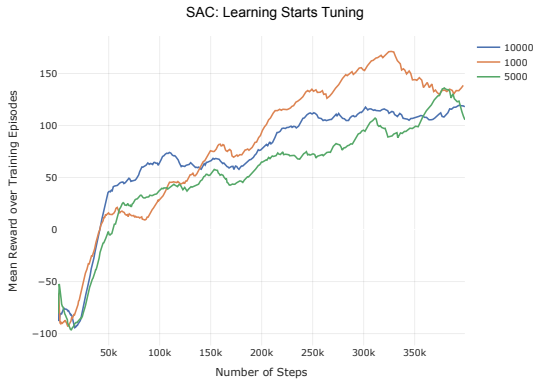


(b) Mean reward over 100 evaluation episodes.

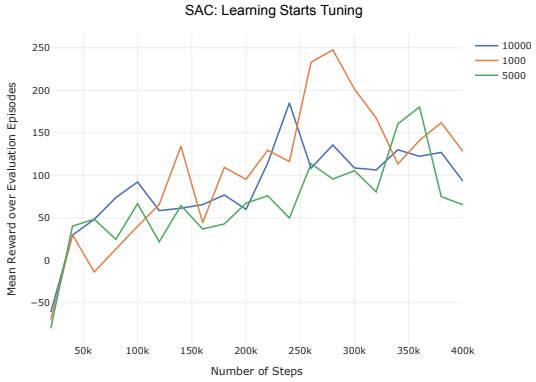


(c) Percentage of successful evaluation episodes over 100 evaluation episodes.

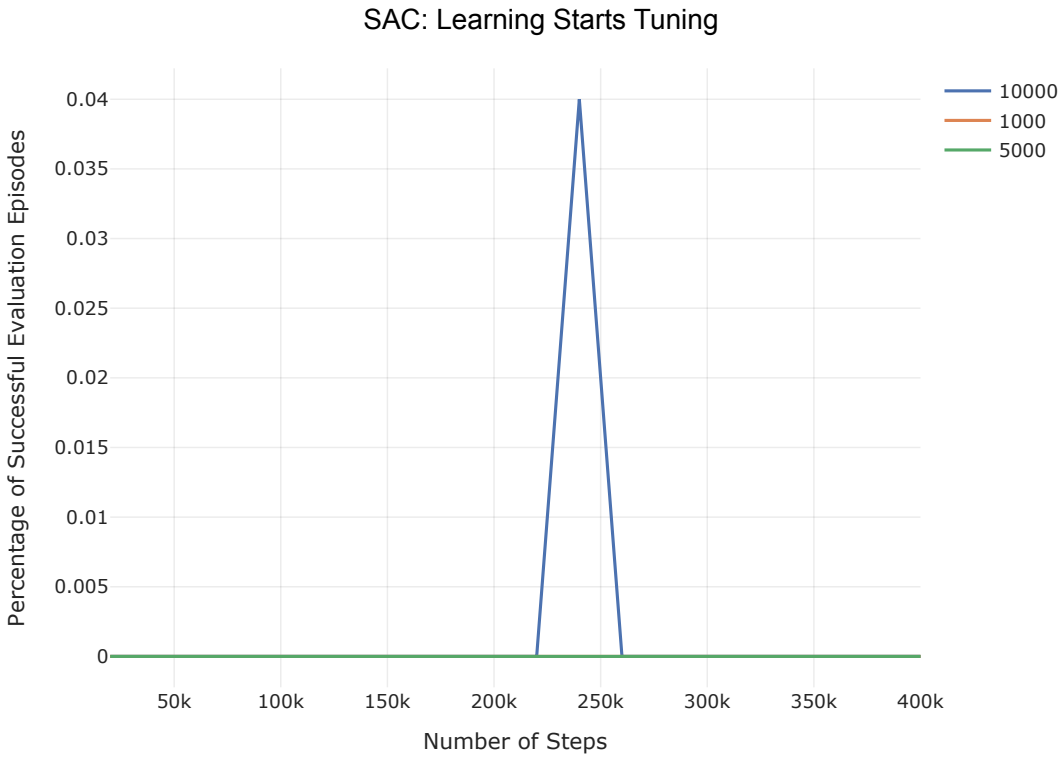
Figure F8: Action noise tuning for SAC algorithm. (a) Mean reward gathered during rollout as training progresses. (b) Mean reward gathered over 100 evaluation episodes, recorded every 20k training steps. (c) Percentage of evaluation episodes that resulted in successful navigation, recorded every 20k training steps.".



(a) Mean episodic reward during training roll-outs.



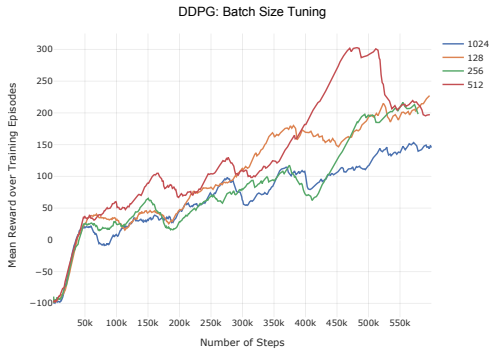
(b) Mean reward over 100 evaluation episodes.



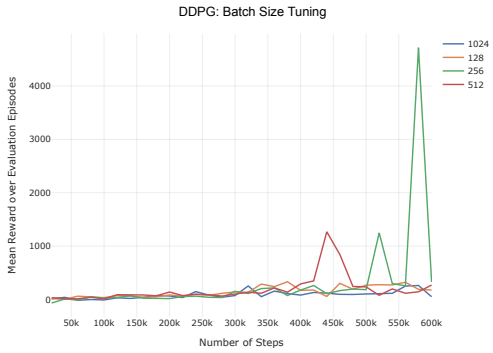
(c) Percentage of successful evaluation episodes over 100 evaluation episodes.

Figure F9: Learning starts tuning for SAC algorithm. (a) Mean reward gathered during rollout as training progresses. (b) Mean reward gathered over 100 evaluation episodes, recorded every 20k training steps. (c) Percentage of evaluation episodes that resulted in successful navigation, recorded every 20k training steps.".

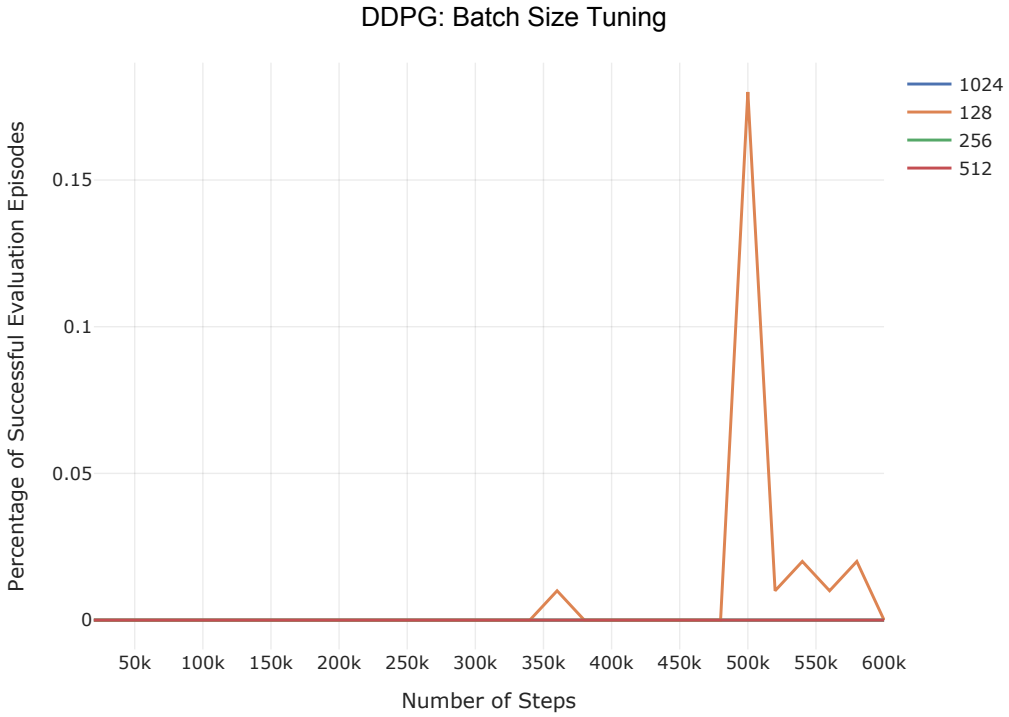
Hyperparameter Tuning: DDPG



(a) Mean episodic reward during training rollouts.

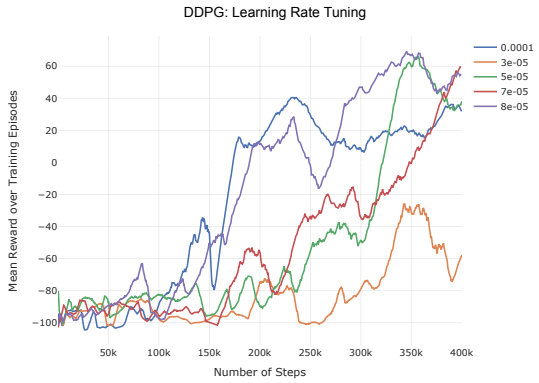


(b) Mean reward over 100 evaluation episodes.

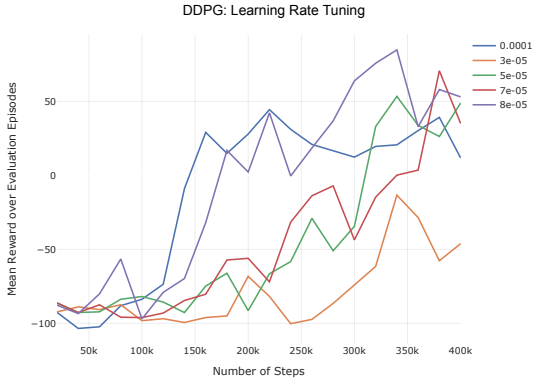


(c) Percentage of successful evaluation episodes over 100 evaluation episodes.

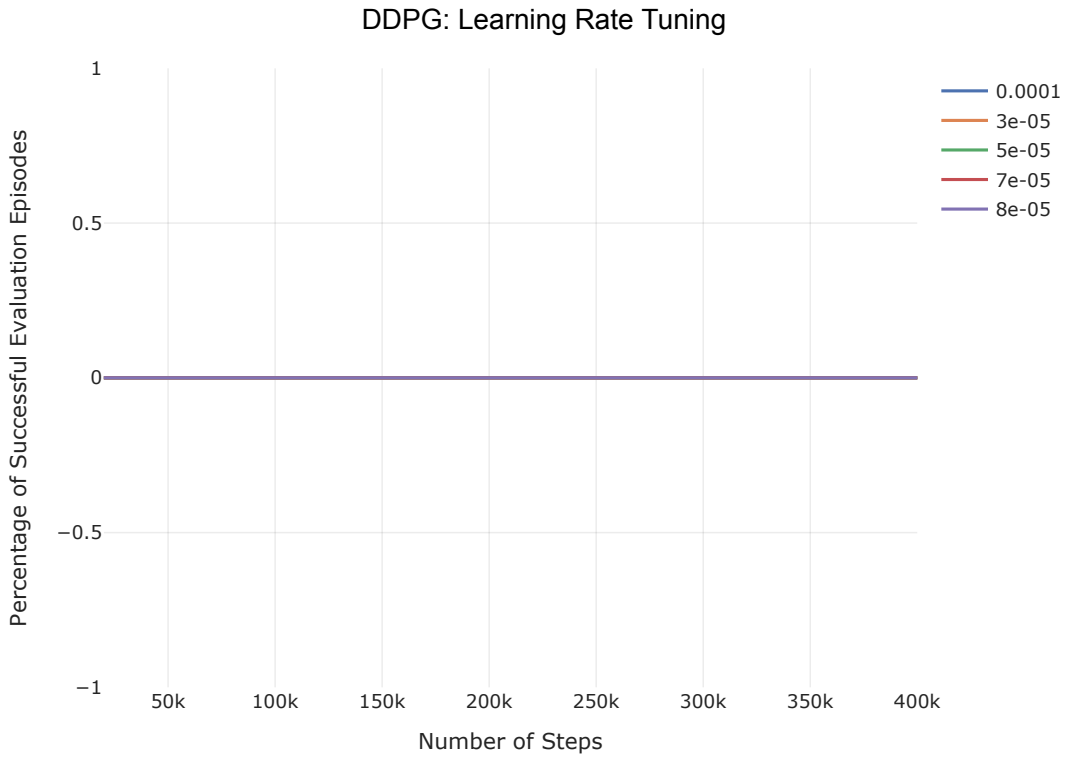
Figure F10: Batch size tuning for DDPG algorithm. (a) Mean reward gathered during rollout as training progresses. (b) Mean reward gathered over 100 evaluation episodes, recorded every 20k training steps. (c) Percentage of evaluation episodes that resulted in successful navigation, recorded every 20k training steps.".



(a) Mean episodic reward during training rollouts.

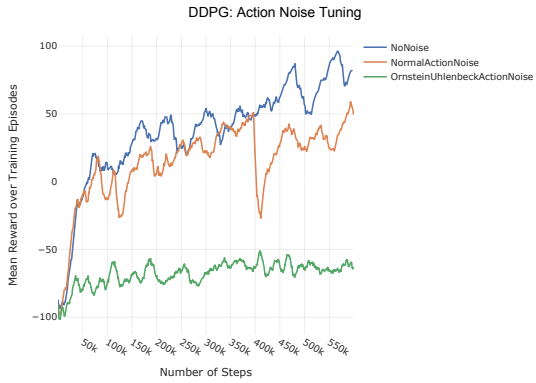


(b) Mean reward over 100 evaluation episodes.

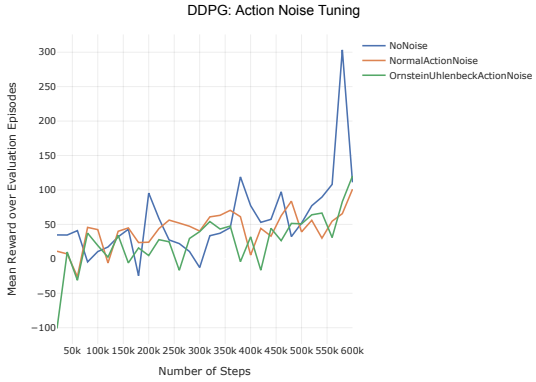


(c) Percentage of successful evaluation episodes over 100 evaluation episodes.

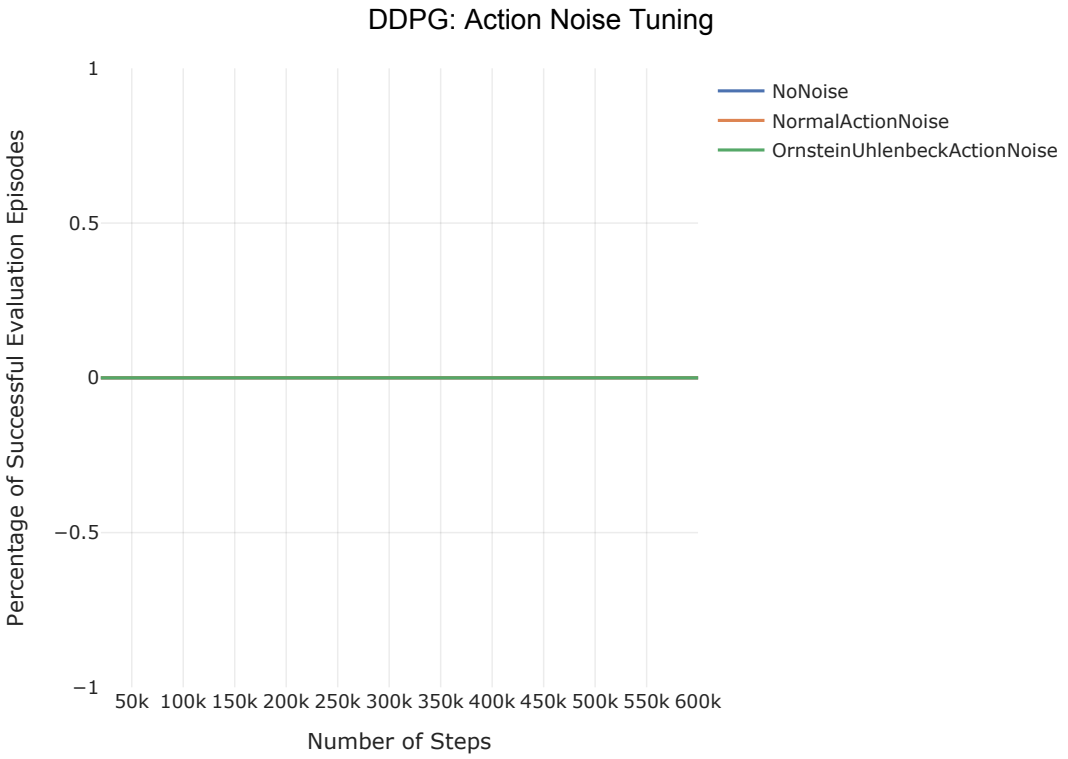
Figure F11: Learning rate tuning for DDPG algorithm. (a) Mean reward gathered during rollout as training progresses. (b) Mean reward gathered over 100 evaluation episodes, recorded every 20k training steps. (c) Percentage of evaluation episodes that resulted in successful navigation, recorded every 20k training steps.".



(a) Mean episodic reward during training rollouts.



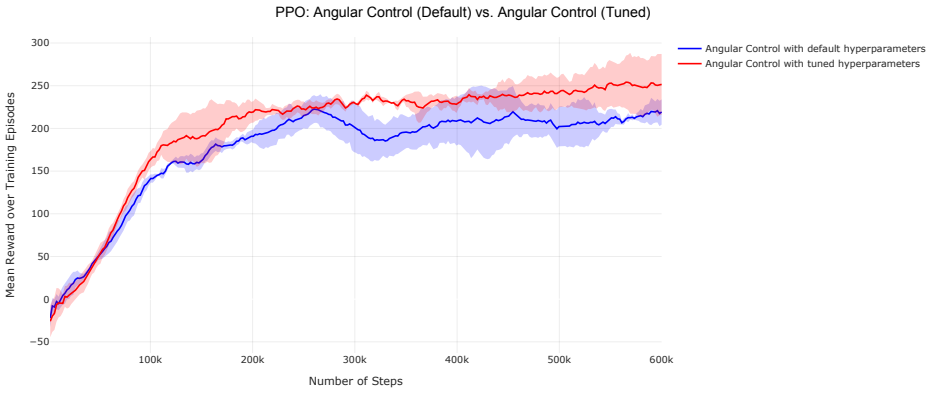
(b) Mean reward over 100 evaluation episodes.



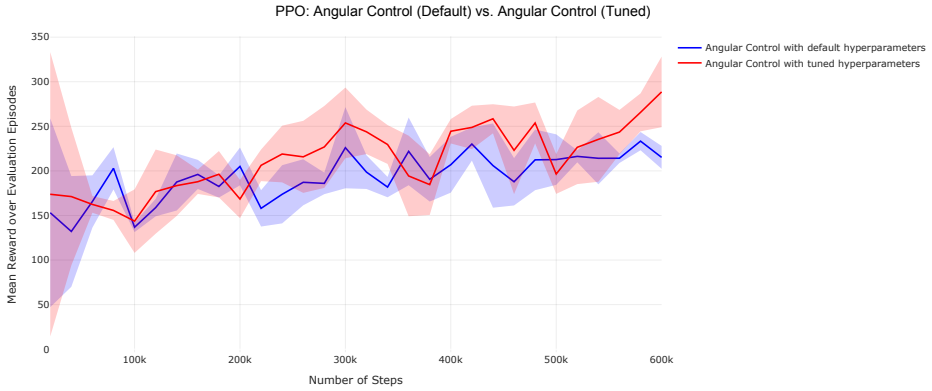
(c) Percentage of successful evaluation episodes over 100 evaluation episodes.

Figure F12: Action noise tuning for SAC algorithm. (a) Mean reward gathered during rollout as training progresses. (b) Mean reward gathered over 100 evaluation episodes, recorded every 20k training steps. (c) Percentage of evaluation episodes that resulted in successful navigation, recorded every 20k training steps.".

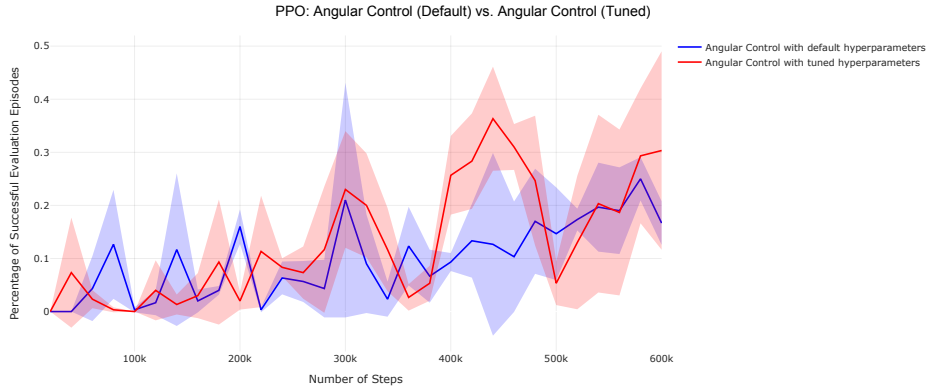
Final Runs Performance: PPO



(a) Mean episodic reward during training rollouts.

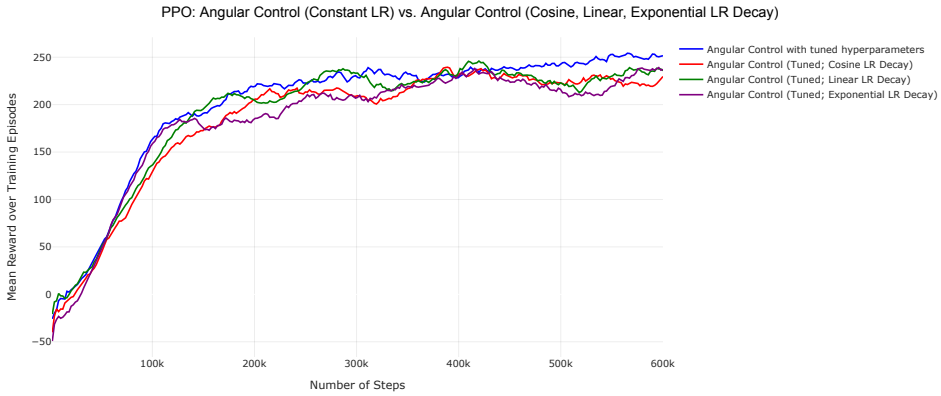


(b) Mean reward over 100 evaluation episodes.

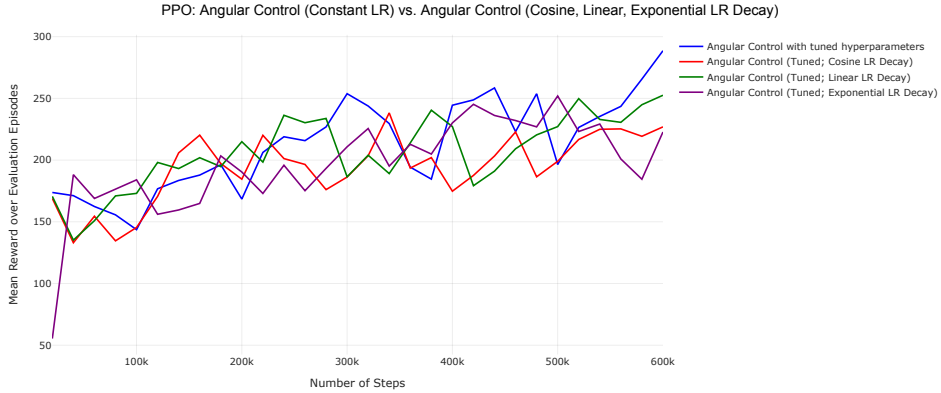


(c) Percentage of successful evaluation episodes over 100 evaluation episodes.

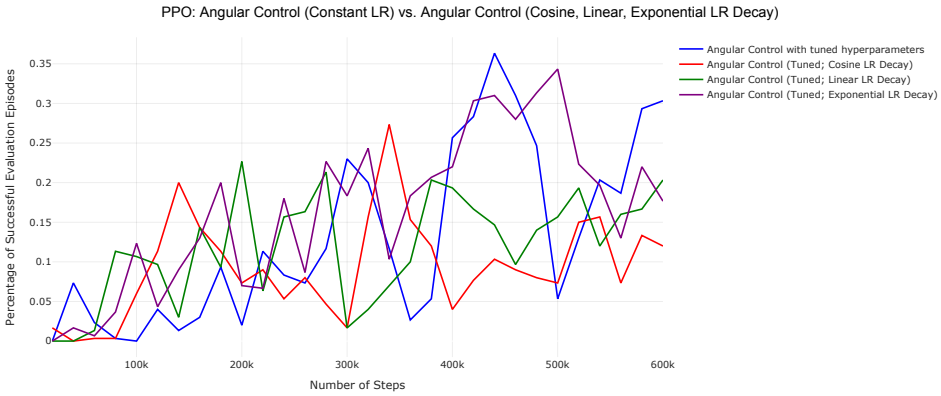
Figure F13: Performance comparison of Angular control under default (blue) and tuned hyperparameters (red) with the PPO algorithm. Shading corresponds to the standard deviation gathered across 3 independant training runs.



(a) Mean episodic reward during training rollouts.

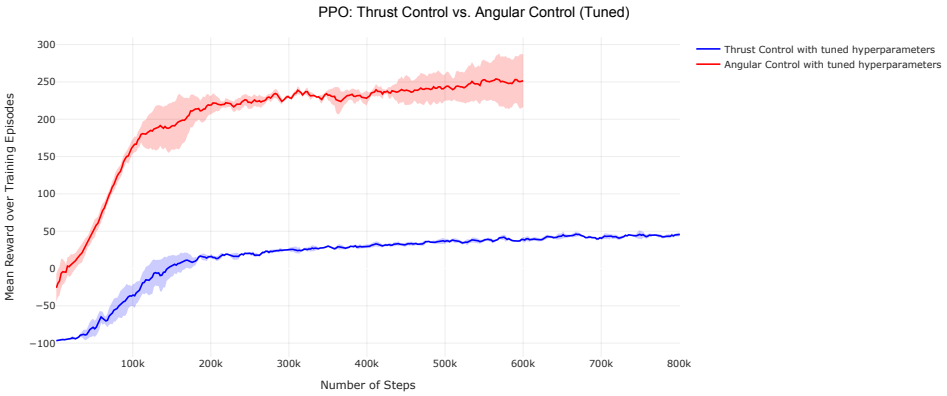


(b) Mean reward over 100 evaluation episodes.

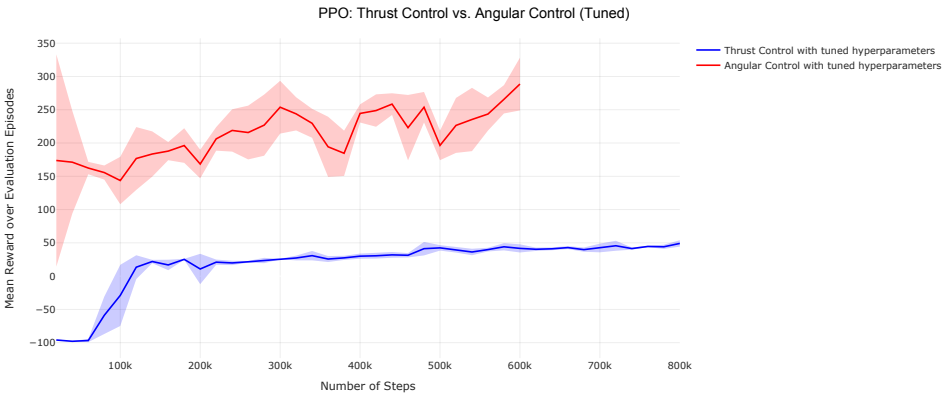


(c) Percentage of successful evaluation episodes over 100 evaluation episodes.

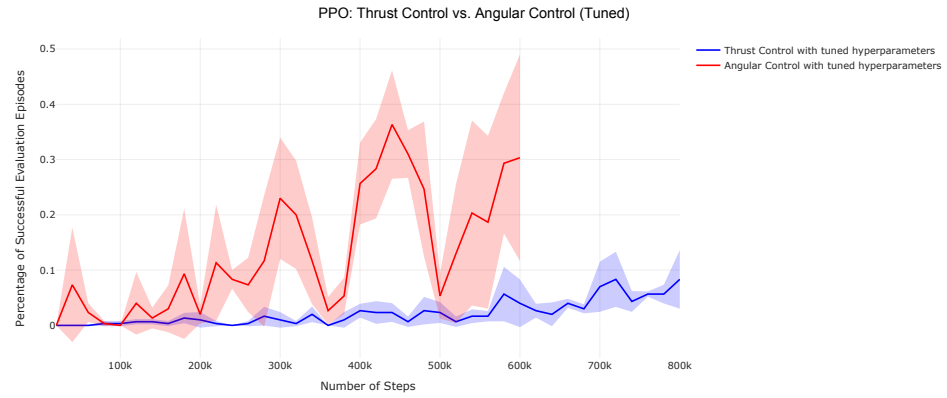
Figure F14: Performance comparison of constant [blue] vs. decay of the learning rate (linear [green], cosine [red], and exponential [purple]) under the PPO algorithm and angular control. All configurations use tuned hyperparameters. No standard deviation is shown to maintain clarity.



(a) Mean episodic reward during training rollouts.



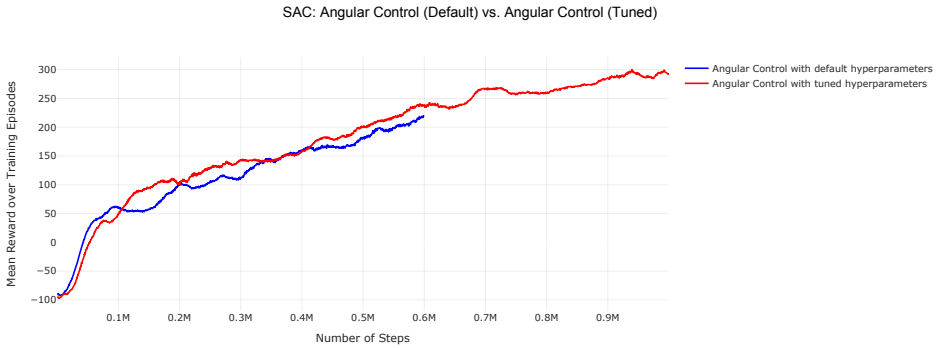
(b) Mean reward over 100 evaluation episodes.



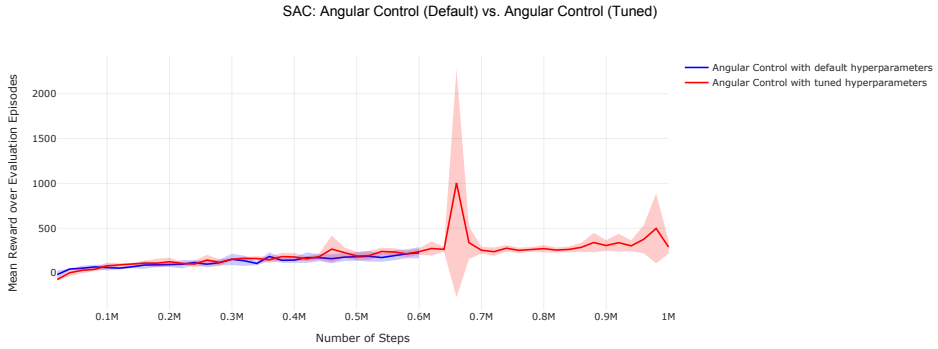
(c) Percentage of successful evaluation episodes over 100 evaluation episodes.

Figure F15: Performance comparison of both command designs: Angular Control (red) and Motor Thrust Control (blue) under the PPO algorithm. Shading corresponds to the standard deviation gathered across 3 independant training runs.

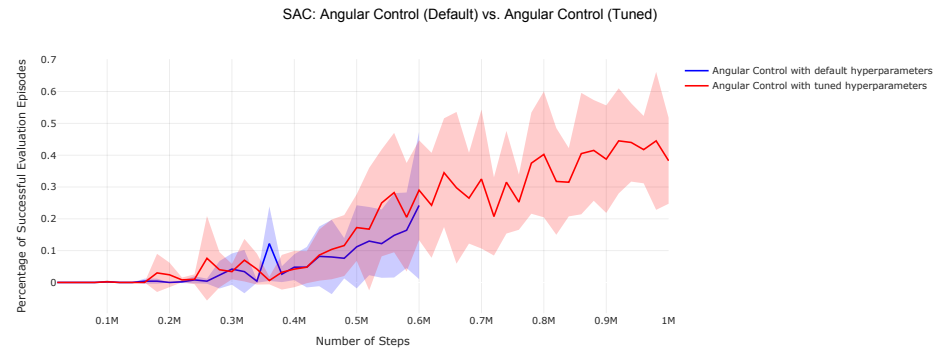
Final Runs Performance: SAC



(a) Mean episodic reward during training rollouts.

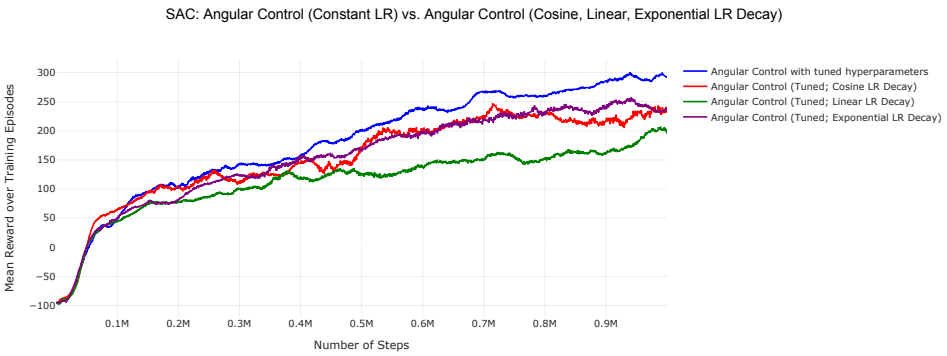


(b) Mean reward over 100 evaluation episodes.

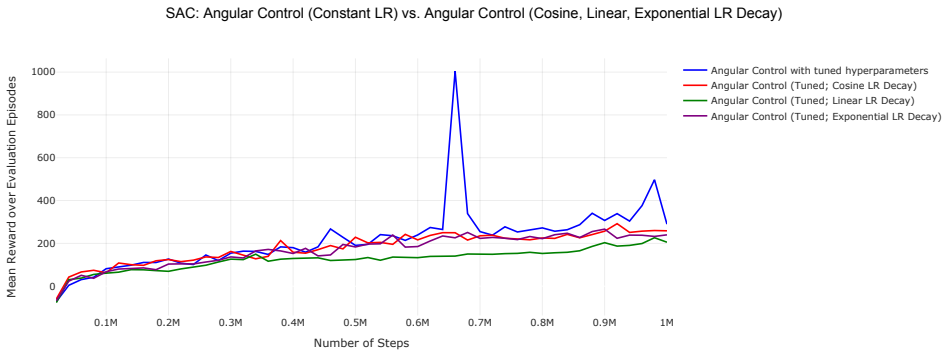


(c) Percentage of successful evaluation episodes over 100 evaluation episodes.

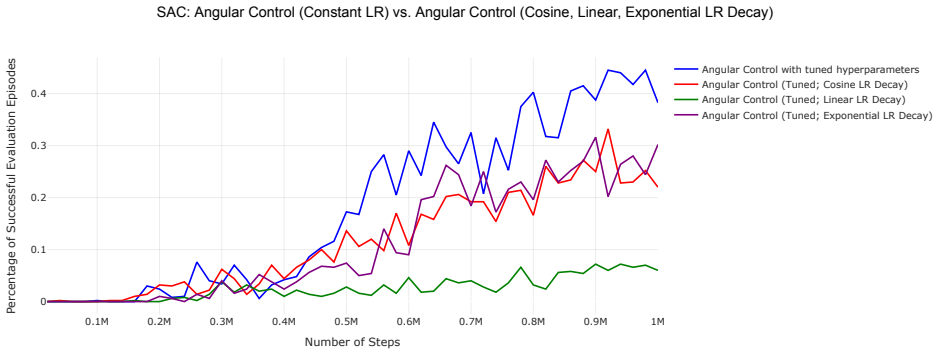
Figure F16: Performance comparison of Angular control under default (blue) and tuned hyperparameters (red) with the SAC algorithm. Shading corresponds to the standard deviation gathered across 3 independant training runs.



(a) Mean episodic reward during training rollouts.

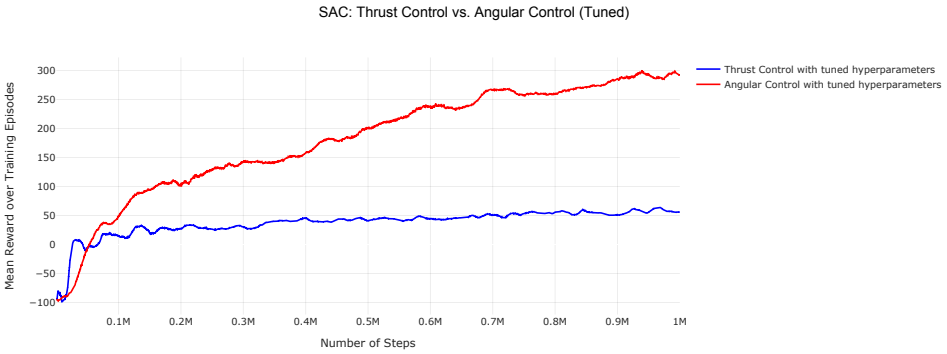


(b) Mean reward over 100 evaluation episodes.

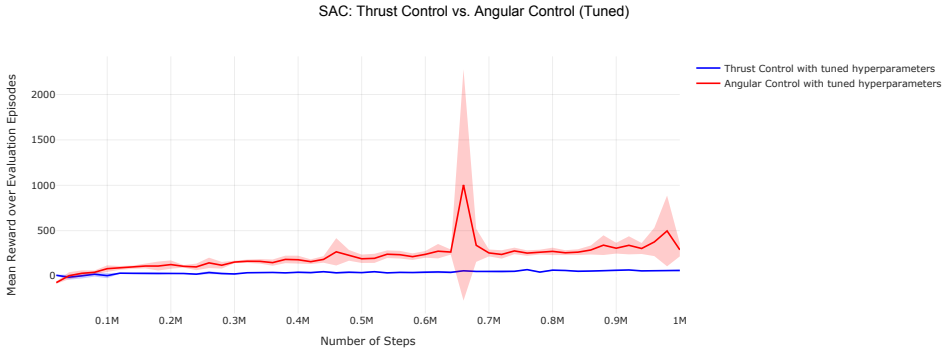


(c) Percentage of successful evaluation episodes over 100 evaluation episodes.

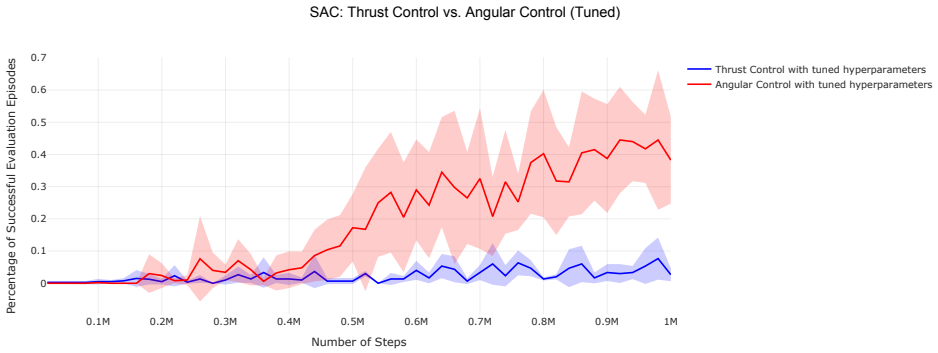
Figure F17: Performance comparison of constant [blue] vs. decay of the learning rate (linear [green], cosine [red], and exponential [purple]) under the SAC algorithm and angular control. All configurations use tuned hyperparameters. No standard deviation is shown to maintain clarity.



(a) Mean episodic reward during training rollouts.



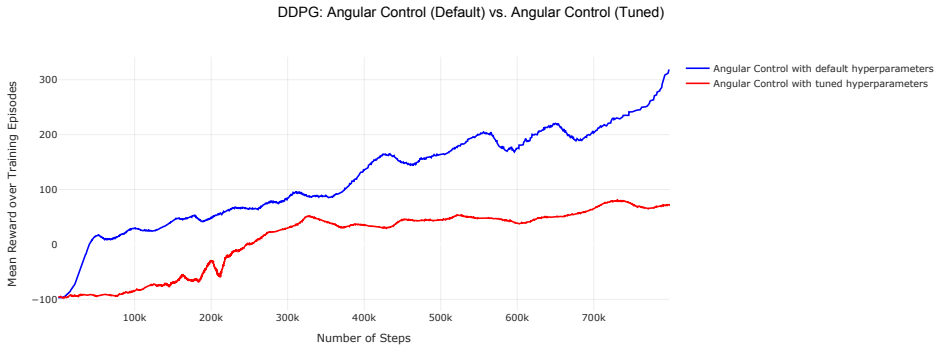
(b) Mean reward over 100 evaluation episodes.



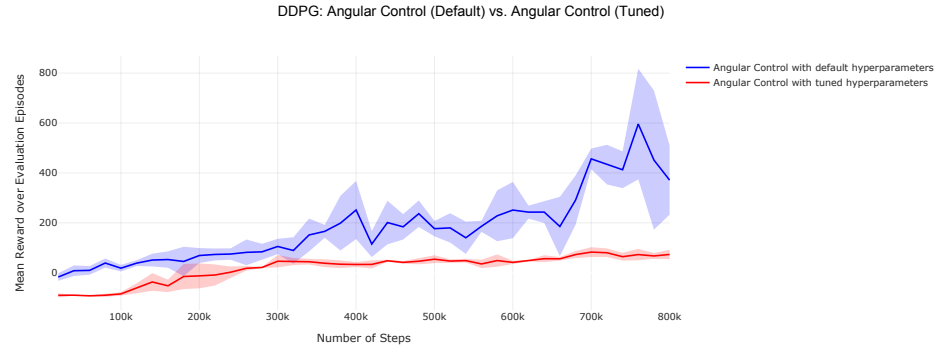
(c) Percentage of successful evaluation episodes over 100 evaluation episodes.

Figure F18: Performance comparison of both command designs: Angular Control (red) and Motor Thrust Control (blue) under the SAC algorithm. Shading corresponds to the standard deviation gathered across 3 independant training runs.

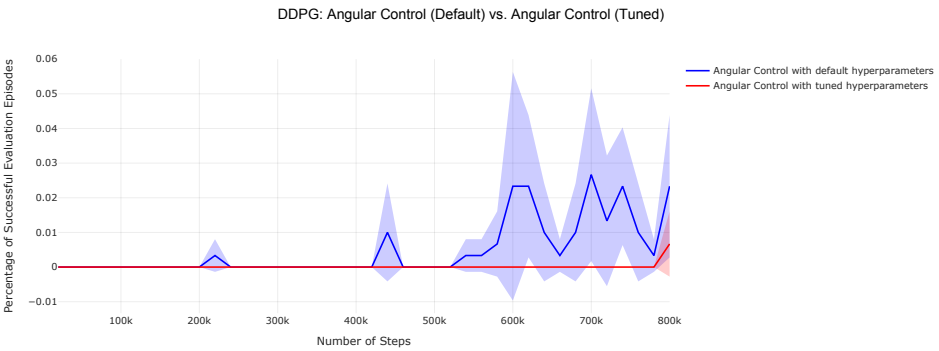
Final Runs Performance: DDPG



(a) Mean episodic reward during training rollouts.

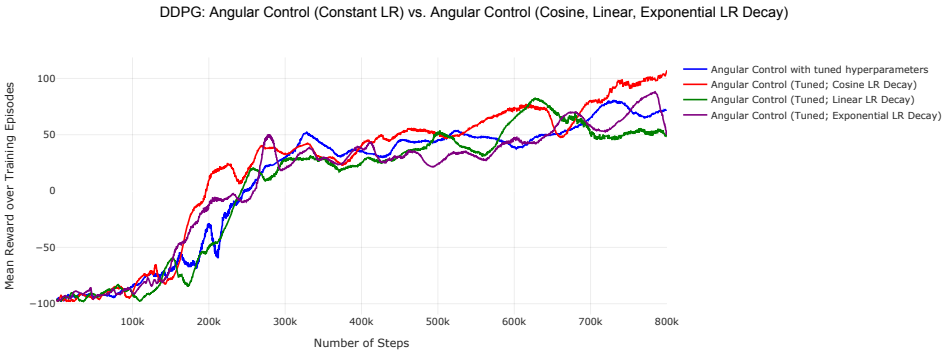


(b) Mean reward over 100 evaluation episodes.

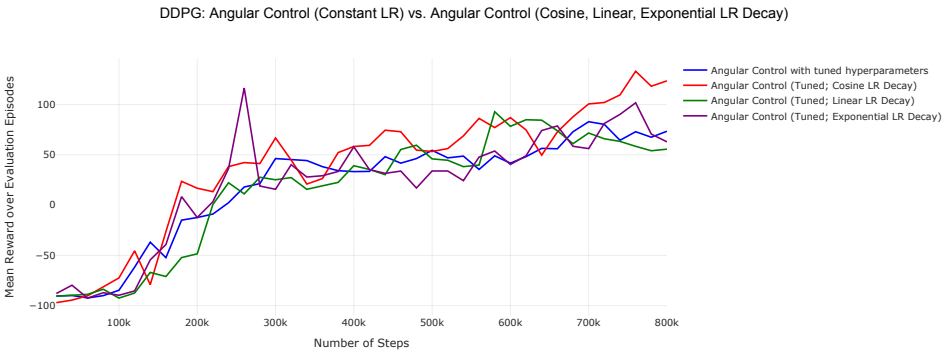


(c) Percentage of successful evaluation episodes over 100 evaluation episodes.

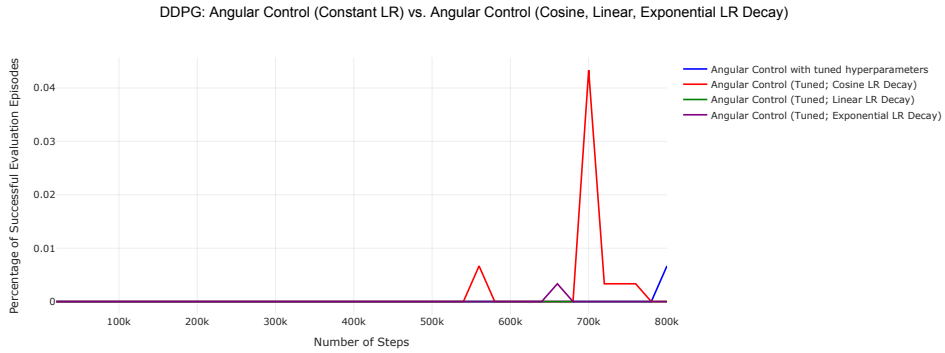
Figure F19: Performance comparison of Angular control under default (blue) and tuned hyperparameters (red) with the DDPG algorithm. Shading corresponds to the standard deviation gathered across 3 independant training runs.



(a) Mean episodic reward during training rollouts.

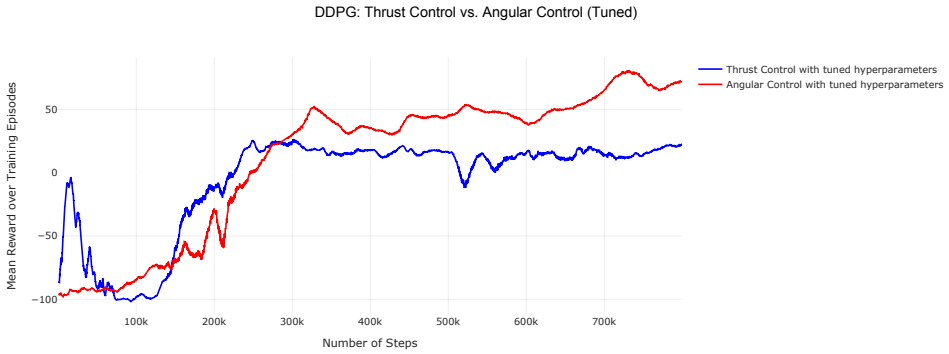


(b) Mean reward over 100 evaluation episodes.

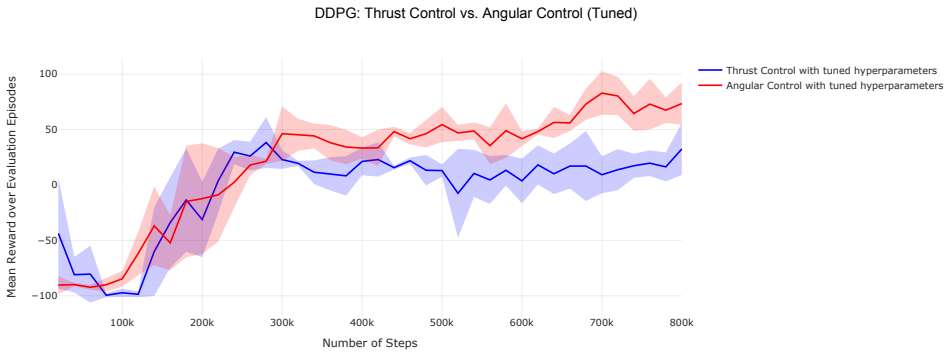


(c) Percentage of successful evaluation episodes over 100 evaluation episodes.

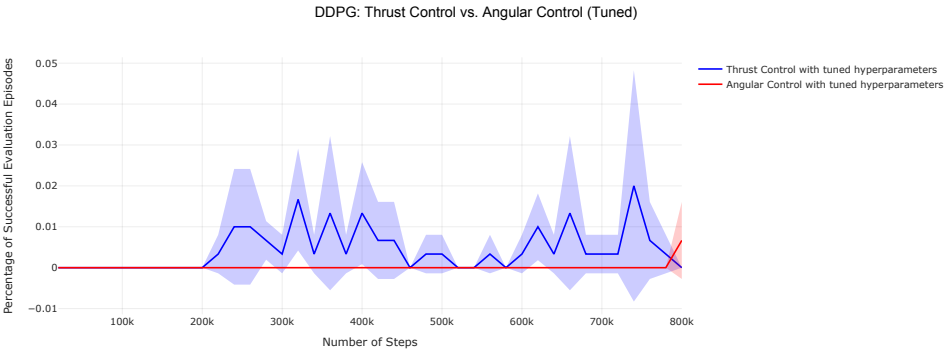
Figure F20: Performance comparison of constant [blue] vs. decay of the learning rate (linear [green], cosine [red], and exponential [purple]) under the DDPG algorithm and angular control. All configurations use tuned hyperparameters. No standard deviation is shown to maintain clarity.



(a) Mean episodic reward during training rollouts.



(b) Mean reward over 100 evaluation episodes.



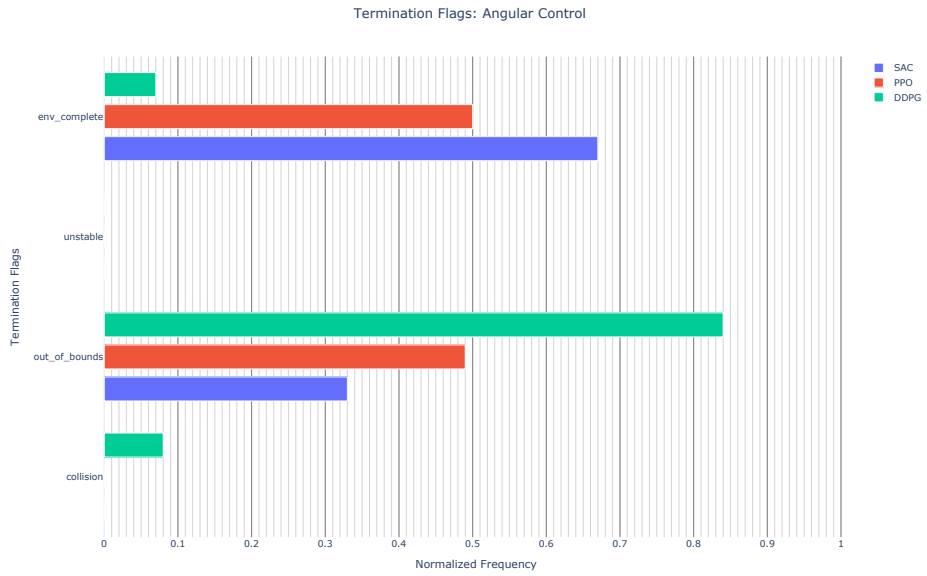
(c) Percentage of successful evaluation episodes over 100 evaluation episodes.

Figure F21: Performance comparison of both command designs: Angular Control (red) and Motor Thrust Control (blue) under the DDPG algorithm. Shading corresponds to the standard deviation gathered across 3 independant training runs.

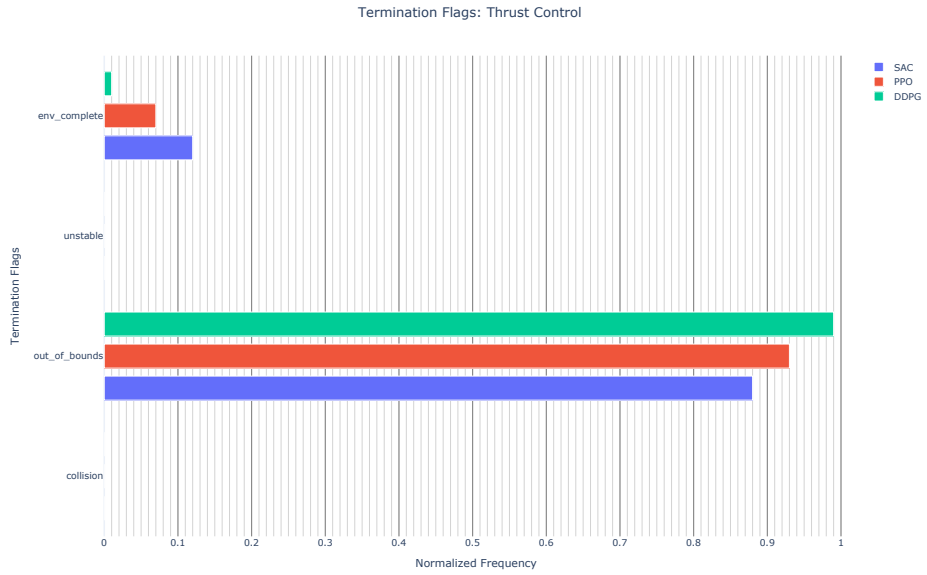
Functions for Learning rate decay

Learning Rate Schedules	
Schedule	**Formula**
Linear Schedule	$\alpha_{\text{lin}}(t) = \alpha_{\text{init}} \cdot t$
Exponential Schedule	$\alpha_{\text{exp}}(t) = \alpha_{\text{init}} \cdot \gamma^{(1-t)}$
Cosine Annealing Schedule	$\alpha_{\cos}(t) = \alpha_{\min} + (\alpha_{\text{init}} - \alpha_{\min}) \cdot 0.5 \cdot (1 + \cos(\pi \cdot (1-t)))$

Table A1: Learning rate schedules used for the training of PPO, SAC, and DDPG.



(a) Termination flags for Angular Control.



(b) Termination flags for Thrust Control.

Figure F22: Comparison of termination flags across both command designs. The plots show the normalized frequency for PPO, SAC, and DDPG. Termination can occur due to task completion (env_complete), angular velocity exceeding the predefined threshold (unstable), the agent colliding with the ground (collisions), or the UAV moving outside the flight dome (out_of_bounds).