Deep Reinforcement Learning based Obstacle Avoidance for UAVs under Measurement Uncertainty

Thesis submitted in partial fulfillment
of the requirements for the degree of

Master of Science
in
Computer Science and Engineering
by Research

by

Bhaskar Joshi
2019111002
bhaskar.joshi@research.iiit.ac.in

International Institute of Information Technology
Hyderabad - 500 032, INDIA
June, 2024

International Institute of Information Technology
Hyderabad, India

CERTIFICATE

It is certified that the work contained in this thesis, titled **"Deep Reinforcement Learning based Obstacle Avoidance for UAVs under Measurement Uncertainty"** by Bhaskar Joshi, has been carried out under my supervision and is not submitted elsewhere for a degree.

_____ _____

Date Advisor: Dr. Harikumar Kandath

To my parents and friends.

Acknowledgments

I am profoundly thankful to the entire community at IIIT Hyderabad for their unwavering support throughout my academic journey. Special thanks go to Prof. Harikumar Kandath, whose guidance and support have been invaluable. His mentorship has been a cornerstone of my academic and professional growth, and for that, I am eternally grateful.

I would also like to extend my gratitude to my peers, Dhruv Kapur, Pratyanshu Pandey, Vedansh Mittal, Utkarsh Upadhyay, Aman Rojjha, and Ayush Sharma, for their camaraderie and encouragement. Their support has been a pivotal part of my enriching experience at IIIT Hyderabad.

To my friends, Atharva, Aditya, Arjun, Boron, Prince, Himanshu, Damodar, Harsh, Khush, Amey your presence made every moment at IIIT unforgettable. The joy and laughter we shared have left an indelible mark on my heart, and I cherish every memory made in your company. Your positive spirits have made my time at the institute truly memorable.

A special note of thanks to my family, whose sacrifices and unwavering support have shaped me into the person I am today. To my parents, whose love and encouragement have been my guiding light, and to my brother Tarun Joshi, whose companionship and support mean the world to me, I am deeply grateful. Your sacrifices, love, and belief in me have been the foundation of my achievements.

Lastly, I extend my gratitude to all my extended family members for their love and support. Your faith in me has been a source of strength and motivation.

This journey has been a collaborative effort, and I am fortunate to have been surrounded by such inspiring individuals. Thank you, one and all, for being a part of my story at IIIT Hyderabad.

Abstract

Deep Reinforcement Learning (DRL) has emerged as a prominent method for enhancing the training of autonomous Unmanned Aerial Vehicles (UAVs). At the heart of this advancement lies the critical and complex challenge of obstacle avoidance, which is essential for ensuring that UAVs can navigate safely and efficiently. The incorporation of DRL into UAV training protocols enhances their ability to autonomously navigate through and adapt to diverse environments. This, in turn, is pivotal for expanding the operational capabilities of UAVs, enabling them to tackle a broader array of applications across various challenging and complex scenarios.

In our research, we undertake a multifaceted examination of the impact of measurement uncertainty on the performance of DRL-based waypoint navigation and obstacle avoidance for UAVs covering both static and dynamic obstacles environment scenario. The challenge increases when the obstacles changes from static to dynamic as now these obstacles can change position in time and affect the navigation.

The measurement uncertainty primarily stems from sensor noise, which impacts localization accuracy and obstacle detection capabilities. We model this uncertainty as adhering to a Gaussian probability distribution, characterized by an unknown mean and variance.

Our research unfolds by assessing the performance of a DRL agent, meticulously trained using the Proximal Policy Optimization (PPO) algorithm, operating within an environment characterized by continuous state and action spaces. This investigation takes place in unseen randomized environments, each exposed to varying degrees of state-space noise, effectively emulating the effects of noisy sensor readings. Our primary objective is to pinpoint the threshold at which the policy becomes susceptible to noise, ultimately paving the way for us to explore diverse filtering techniques to mitigate these detrimental effects.

Our findings reveal that the DRL agent exhibits a remarkable degree of inherent robustness against specific types of noise. We leverage this inherent robustness to bolster its performance in scenarios where it would otherwise succumb to the deleterious influence of state-space noise. To empirically validate our research, we undertake extensive training and testing of the DRL agent across a spectrum of UAV navigation scenarios within the PyBullet physics simulator.

In a significant stride towards practical applicability, we port the policy distilled through simulation directly onto a real UAV without any additional modifications, and subsequently, we meticulously verify its performance in a real-world operational setting. This transformative approach holds the potential to inform the selection of sensors with varying biases and variances, and intriguingly, it suggests that artifi-

cially injecting noise into measurements can yield performance enhancements in certain scenarios. The substantiation of this proposition is achieved through rigorous testing on a real UAV, thereby solidifying the practicality and real-world utility of our approach.

In summation, our research contributes to the burgeoning field of UAV autonomy by shedding light on the intricate interplay between measurement uncertainty and Deep Reinforcement Learning-based navigation and obstacle avoidance. The insights garnered from our study not only advance our understanding of UAV operation but also provide actionable guidance for sensor selection and noise injection strategies to bolster real-world performance.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Unmanned aerial vehicles (UAVs) are commonly used for various critical missions that often require navigation in unpredictable environments with obstacles and potential threats to the safety of the UAV[3]. In recent years, the widespread application of unmanned aerial vehicles (UAVs) has become increasingly evident in various civilian tasks such as object tracking[4], 3D reconstruction[5], wireless coverage provision[6], remote sensing[7], wildlife protection[8], search and rescue operations[9], goods delivery[10], security and surveillance[11], agriculture[12], and inspection of civil infrastructure[13]. Despite the numerous applications, navigation remains challenging primarily in dynamic and stochastic environments. Velocity obstacles[14], artificial potential fields[15], fuzzy logic[16] and genetic algorithms [17] are some methods for obstacle avoidance in dynamic environments. Methods like SLAM [18] are effective in high obstacle-density environments, using data from sensors like cameras and LIDAR. This makes it too computationally complex for real-time implementation in UAVs.

An increasingly popular approach for waypoint navigation amidst obstacles is to use Reinforcement Learning (RL)[19]. RL allows an agent to learn the optimal behaviour for a particular task through trial-and-error interactions with the environment. This makes RL a promising method for improving the autonomy of agents in various robotics applications [20].

Our research focuses on examining the impact of measurement uncertainty on the efficacy of DRL-based waypoint navigation and obstacle avoidance in UAVs, accounting for both static and dynamic obstacles. Measurement uncertainty arises from sensor noise affecting localization and obstacle detection, it is assumed to follow a Gaussian probability distribution with an unknown non-zero mean and variance and it makes the relative position of the obstacles and the target from the UAV uncertain. Our research undertakes the evaluation of a Deep Reinforcement Learning (DRL) agent's effectiveness, which has been trained using the Proximal Policy Optimization (PPO) technique. Extensive training and testing of the DRL agent under various UAV navigation scenarios are performed in the PyBullet physics simulator. To evaluate the practical validity of our method, we port the policy trained in simulation onto a real UAV without any further modifications and verify the results in a real-world environment.

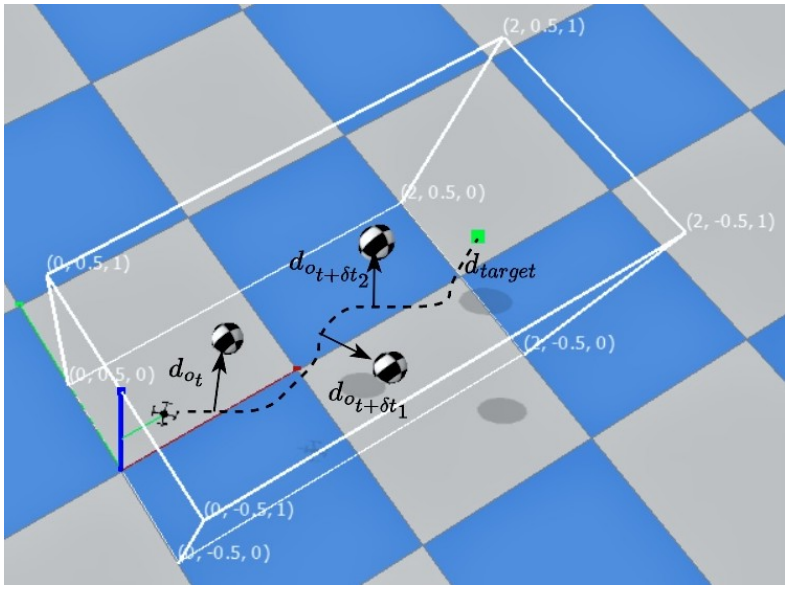


Figure 1.1: **UAV Obstacle Avoidance Scenario**: The UAV starts at one end of the environment, and must make it to the target (indicated by the green square) at the other end. Between them lie obstacles that the UAV must avoid. Further, the UAV's position is constrained at all times by a geofence, indicated by the white lines (figure from the simulated environment).

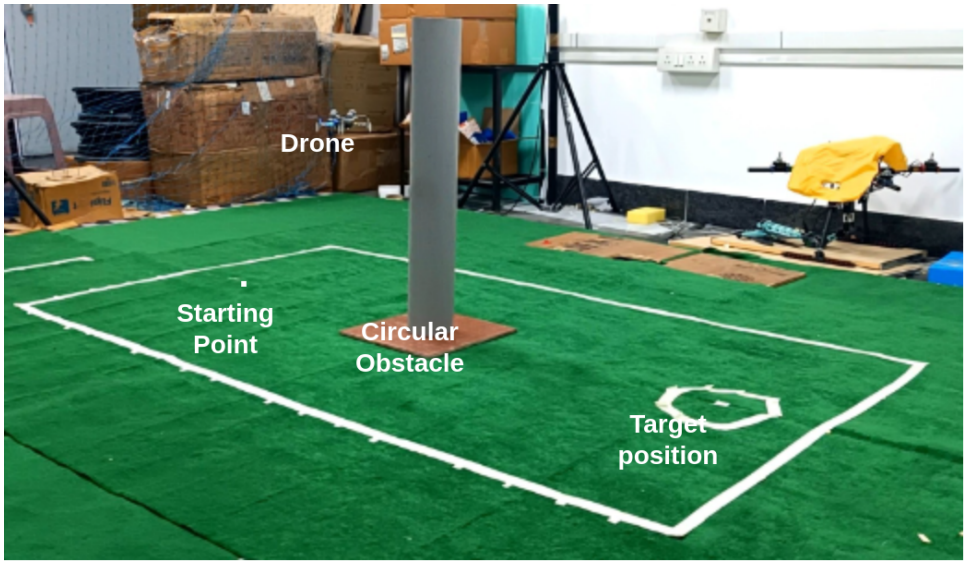


Figure 1.2: **Real World UAV Navigation Test Setup (Side View)**: This image illustrates the experimental setup delineating an operational test area with clear white boundary lines. The setup features a Crazyflie 2.1 drone initiating at a marked starting point, navigating around a cylindrical obstacle, and aiming for a designated target position. The environment simulates potential real-world conditions to assess the UAV's autonomous navigation capabilities.

1.1 Related Works

The advent of Reinforcement Learning (RL) as a prominent approach for waypoint navigation in complex environments marks a significant advancement in the field of robotics, particularly for Unmanned Aerial Vehicles (UAVs). RL facilitates agents in acquiring optimal behavior for specific tasks through trial-and-error interactions with their environment, thereby augmenting the autonomy of agents in diverse robotics applications [21].

Pham et al. [22] provide a framework for using reinforcement learning to enable UAVs to navigate a discretized environment. S. Ouahouah et al. [23] propose probabilistic and Deep Reinforcement Learning based algorithms for avoiding collisions in a discrete space environment.

Expanding the scope to dynamic obstacle avoidance, several studies have introduced innovative RL applications. The work of Guo et al. [24] that presented Layered-RQN, an improved DRL technique for UAVs to navigate in dynamic situations. Yan et al. [25] employ D3QN, a DRL algorithm for real time path planning in dynamic environments. They use a separate action space that includes eight directions around the UAV. Yang et al. [26] proposed a framework based on double deep Q-network with priority experience replay (DDQN-PER) which allows UAVs to navigate and avoid obstacles in a dynamic environment.

To enhance UAV navigation training, Villanueva et al. [27] introduced a technique to improve UAV navigation training by adding Gaussian noise to a DRL agent's action space. This reduced navigation steps and flight time per task. There have been work of Zhao et al. [28] that study generalisation in RL, however it is completely in simulation. TQ Pham (2019)[29] proposed an avoidance assistance system in simulation and W Poomarin et. al [30] used trajectory guidance for automatic docking of wheel mobile robot.

All of the procedures described above presume that the data acquired from the sensors is perfect.

However in real-world scenarios, the sensors and techniques used for UAV localisation result in noise. Some common techniques used to localise UAV in indoor environments are IMU(inertial measurement unit), VO(visual odometry) and SLAM(simultaneous localisation and mapping) [18]. IMUs are usually chosen because they can provide the measurement data at a higher sampling rate. But over time IMU measurements are drifted and generate errors [31]. Because of this IMUs are fused with other methods like visual odometry [32], or GPS [33]. However, cameras used for visual odometry also suffer through noise [34]. UWB (Ultra-wide band) based techniques are also used for localisation in GPS denied environments[35] [36]. UWB signals can experience reflections, diffractions, and scattering from various surfaces and objects in the environment. This leads to noise in obtained data. Yang et al [35] presents a UWB based technique for localisation producing noise with bias and variance upto 0.2 m and 0.07m respectively. Another commonly used technique for UAV localisation involves using RF(radio frequency) [37]. RF, similar to UWB may also face reflections, diffractions, scattering, signal attenuation and interference causing noise in measurements. Ultimately, there are many techniques, and sensors available for UAV localisation, with bias and variance trade-offs.

Recent works have started addressing noise in RL-based UAV navigation. Hu et al[38] use PPO, a DRL algorithm for autonomous obstacle avoidance in a UAS (Unmanned Aircraft System) with moving obstacles considering a continuous state and observation space. The authors consider uncertainty in the UAS position due to various factors such as wind, true speed, and positioning error.However they use fixed levels of noise. Wan et al[39] propose a novel DRL method Robust-DDPG for UAV navigation in dynamic environments. They use Gaussian noise as adversial attacks during training to improve policy robustness.

All these works in RL based navigation for UAVs in complex and stochastic environments consider noisy sensors to allow better generalisation of the RL agent. However, these works do not provide any reasoning or study behind the choice of the noise added, and to the best of our knowledge there is no systematic study on generalisation on RL based navigation for UAVs. There have been works like those of A. Zhang et al. [40], C. Zhang et al. [41] and Cobbe at al. [42] that study generalisation in RL, however all these works are on games that have little to no physical significance. We thus provide a systematic evaluation of policies for UAV navigation in a dynamic environment trained on different levels of noise.

No studies have been conducted on the effects of measurement noise in the context of training and implementation of DRL-based algorithms for UAVs that are deployed from simulation to real world.

Building upon this foundation, our work also extends to the exploration of RL's capacity to mitigate the adverse impacts of time-varying noise, such as those observed with Inertial Measurement Units (IMUs). Narasimhappa et al [43] use a modified Kalman Filter (SHAKF) to incorporate time-varying noise estimator to reduce the drifting measurements of the IMU while Tsiakmakis et al[44] used DRL to learn a policy to correct localization errors from IMUs leading to more precise localization. However we would like to look at reinforcement learning as a method to eradicate the effects of such noise by training an inherently robust policy. To the best of our knowledge, there are no other works that acknowledge such type of noise while training RL based policies for UAV navigation. We also incorporate the study of time varying bias for static obstacle in our study in chapter 4.

1.2 Problem Formulation

The agent's main objective is to reduce the separation between the Unmanned Aerial Vehicle (UAV) and the target (d_{target}) to a level within a predefined acceptable margin of error ($\epsilon_{\text{success}}$). Simultaneously, the agent must ensure a minimum safety distance (ϵ_{safe}) from the closest obstacle d_{o_i} at all times, with the understanding that this constraint trivially holds for other obstacles in the environment like shown in figure 1.3.

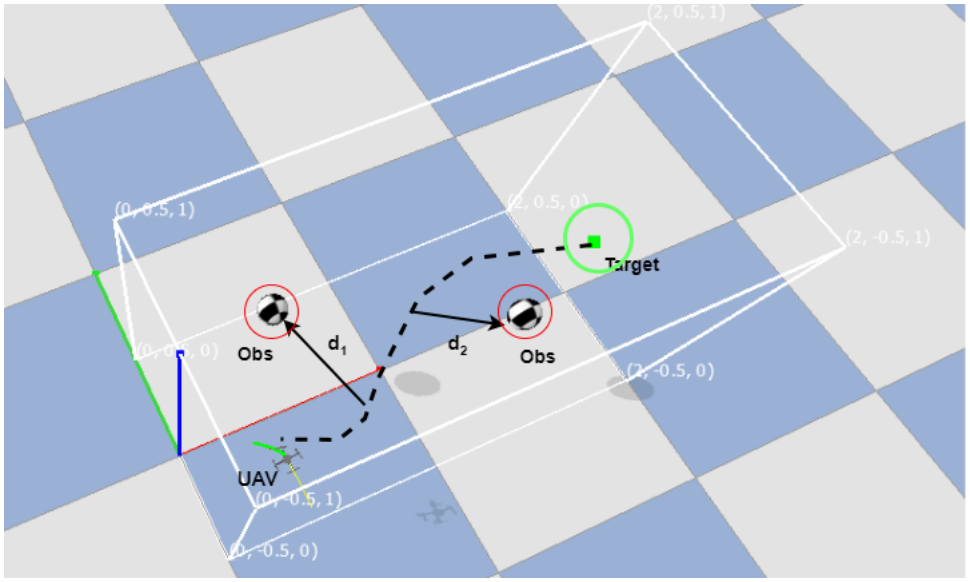


Figure 1.3: **Obstacle Avoidance Challenge**: The UAV is tasked to navigate towards the target, strategically maneuvering around obstacles that lie along its path.

However, the presence of noisy sensors introduces inaccuracies in the UAV's localization estimates. These inaccuracies manifest as deviations in the UAV's intended path, increasing the risk of collisions with obstacles, as depicted in Figure 1.4.

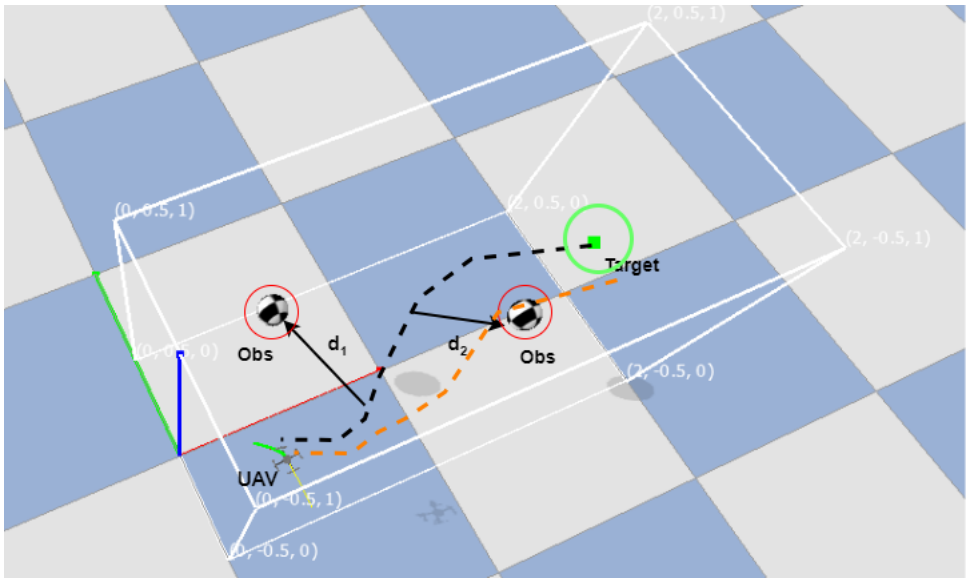


Figure 1.4: **Imprecise Localization Arising from Sensor Noise**: The UAV's trajectory, indicated by the orange dotted line, illustrates the deviations caused by noisy sensor data, increasing the risk of collision with obstacles.

The estimated position variables $\hat{x}$ and $\hat{y}$ are defined as:

$$\hat{x} = x + \eta_x \qquad \hat{y} = y + \eta_y \tag{1.1}$$

$$\eta_x, \eta_y \sim \mathcal{N}(\mu, \sigma) \tag{1.2}$$

Here x and y denote the true position of the UAV and η_x and η_y are localization errors caused due to sensor noise, which conform to a normal distribution $\mathcal{N}$ with parameters μ and σ

The challenge of obstacle avoidance increases because the UAV's estimated position deviates from its actual position due to sensor noise, leading to uncertainty in the observed state at each time step.

Our investigation aims to assess the inherent robustness of a policy learned through deep reinforcement learning to such noise. To tackle this challenge, we leverage the Proximal Policy Optimization (PPO) algorithm, a well-established method in deep reinforcement learning (DRL), known for its stability and efficiency in policy learning. PPO plays a pivotal role in our investigation by allowing us to systematically evaluate and enhance the robustness of the UAV's navigation policy in the face of measurement noise. By training and evaluating the DRL agent under various levels of sensor noise, with unknown mean and variance, we aim to not only assess the inherent resilience of the learned policy but also explore strategies to augment its performance despite the adversities posed by imprecise sensor data. The approach ensures the UAV's proficiency in maintaining safety distances and achieving its navigation objectives under fluctuating levels of environmental uncertainty, highlighting the indispensable value of PPO in developing robust autonomous systems capable of operating reliably in real-world conditions fraught with sensor imperfections.

1.3 Thesis Contribution

The key contributions of this thesis are given below.

- This is the first study that systematically analyzes the effects of noisy sensor inputs on DRL-based waypoint navigation and obstacle avoidance for UAVs.

- The measurement noise is modelled as a random variable sampled from a Gaussian distribution. Both training and evaluation of the well-known DRL algorithm, Proximal Policy Optimization (PPO) is performed in the presence of measurement noise with different levels of the unknown mean and variance. The performance of the DRL agent trained with perfect measurements is compared with other agents trained with different levels of measurement noise.

- We also consider scenarios involving time-varying bias, where the bias μ in the Gaussian noise $\mathcal{N}(\mu, \sigma)$ linearly changes over time. This time-varying bias is often observed in noise originating from position estimates provided by Inertial Measurement Units (IMUs).

- We show that artificially injecting noise with carefully chosen variance into the existing measurement error improves the performance of the DRL agent when the measurement error has some unknown bias.
- The policy trained in simulations can be directly deployed to a real-world environment for waypoint navigation and obstacle avoidance, using a CrazyFlie 2.1.

1.4 Thesis Overview

This thesis presents a comprehensive study of Reinforcement Learning (RL) applied to quadrotors, addressing key concepts, algorithm taxonomy, and experimental validations with both simulations and real-world applications. The main contributions and the structure of the thesis are outlined as follows:

- In Chapter 2, provides background for the study: foundational concepts of RL are discussed, including states, observations, and action spaces, along with a detailed exploration of policy determination, value functions, and the optimization problem intrinsic to RL. UAV model, Mocap, Gaussian Noise, Low Pass Filter (LPF), Kalman Filter (KF), Velocity Obstacle (VO) and Pybullet.
- Chapter 3 details a comprehensive study for navigating static obstacles using RL. The methodology includes a thorough description of the environment, the agent, and the impacts of noise and time-varying bias. The results section evaluates the trained policies under various noise conditions, analyzing mean rewards and episode durations.
- The focus of Chapter 4 is a study on dynamic obstacles. It presents the methodology and environment setup, delves into the agent's structure, and discusses the effects of noise. Results from training in two distinct environments are assessed, with evaluations of the trained policies' robustness against different types of noise presented alongside a discussion section.
- In Chapter 5, the sim-to-real transfer is showcased through experiments with static obstacles in a real-world setup. This includes the implementation and testing protocols, an evaluation of noise impact, performance metrics, and a summary of the experimentation results.
- Finally, in Conclusion section we conclude the thesis, summarizing the key findings, discussing the implications of the research, and suggesting directions for future work.

Chapter 2

Background

Reinforcement Learning (RL) is a machine learning paradigm that function by incentivizing desired behaviours and penalizing undesirable ones in order to instruct agents in decision-making [1]. In reinforcement learning (RL), an agent interacts with its environment to achieve a goal, in contrast to supervised learning where models are trained on a preset set of data with known outputs. The agent discovers, by means of trial and error, the optimal course of action in various circumstances so as to maximize its cumulative reward over time.

The learning process is regulated by the exploration and exploitation principles, in which the agent actively pursues novel strategies and utilizes its prior knowledge to optimize its decision-making. Typically, the environment offers rewards or penalties as feedback, which the agent employs to modify its behaviour. This framework possesses a broad range of applications, encompassing finance, healthcare, games, and robotics. It empowers systems to enhance their performance in dynamic, complex tasks autonomously.

2.1 Key Concepts in RL

The principal entities in Reinforcement Learning (RL) are the agent and the surrounding environment. The global context in which the agent interacts and resides is referred to as the environment. The agent determines the most suitable course of action after perceiving a (potentially incomplete) observation of the state of the world at each stage of interaction. In addition to changes that occur during agent interaction, the environment is also capable of undergoing spontaneous transformations.

The agent also receives a reward signal from the environment, which is a numerical indication of the quality of the current condition of the world. The objective of the agent is to optimize its cumulative reward, known as the return. Reinforcement learning methods refer to strategies employed by an agent to acquire behaviours that enable it to accomplish its objective .

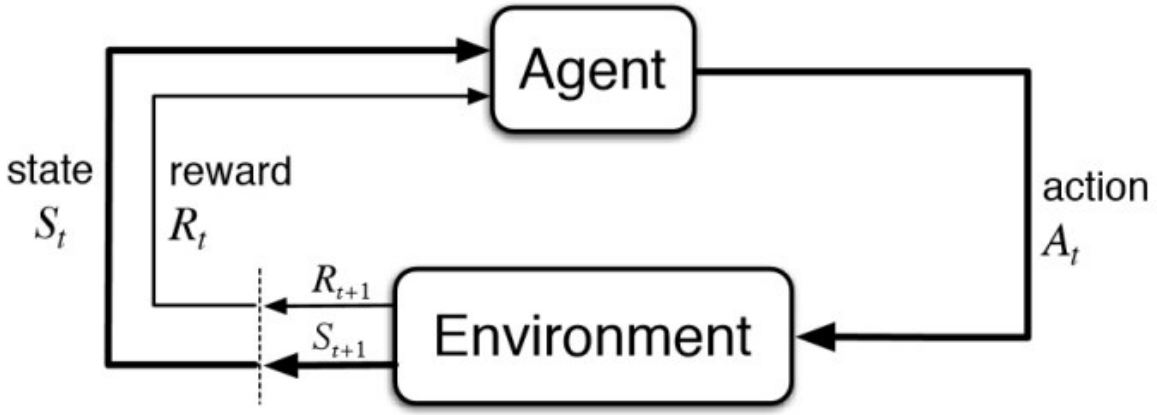


Figure 2.1: **Agent Environment Interaction loop**: The figure depicts the fundamental framework of a Reinforcement Learning model, in which an agent engages with an environment by taking action A_t, the action is then received by the environment which uses that to produce a reward R_{t+1} and a state S_{t+1} at the next moment in time and then this is then passed to the agent for the next action to be decided. Image ref. [1]

In order to provide a more precise explanation of the function of RL, it is necessary to introduce further terminology [1], [45]:

2.1.1 States and observations

A state s, offers a thorough understanding of the world's condition, leaving no aspect of the world undisclosed. Conversely, an observation o provides an incomplete view of a state, potentially excluding some details.

In deep reinforcement learning, states and observations are typically encoded as vectors, matrices, or tensors of real numbers. For example, a visual observation might be encoded as the RGB values of its pixels, while the condition of a robot could be captured by the angles and velocities of its joints.

In reinforcement learning terminology, the symbol for state s, is occasionally used in contexts where it would be more accurate to use the symbol for observation, o. This is particularly evident when discussing the process by which an agent determines an action. The notation often suggests that the action is based on the state, s, whereas in reality, the action is based on the observation, o, because the agent does not have full access to the state. This reflects a simplification or idealization in the notation that does not always align with the practical limitations of the agent's access to information.

If an agent has access to the entirety of the environment's state, the environment is considered fully observable. On the other hand, if the agent only receives partial information about the state, the environment is deemed partially observable.

2.1.2 Action space

Environments in reinforcement learning dictate the types of actions that can be taken, with the entirety of valid actions within a specific environment being referred to as the action space. For some environments, like those in Atari games or the game of Go [46] , the action space is discrete, meaning the agent has a limited set of specific moves it can make. In contrast, environments that involve controlling a robot in the physical world usually feature continuous action spaces, where actions are represented by real-valued vectors and can vary across a wide range.

The distinction between discrete and continuous action spaces significantly impacts the choice and design of algorithms in deep reinforcement learning. Certain algorithm families are tailored exclusively for either discrete or continuous action spaces and require significant modifications to be applicable in the other scenario. This difference fundamentally affects the strategies, techniques, and theoretical approaches employed in deep RL, influencing how algorithms learn, adapt, and perform across various environments.

2.1.3 Policy

A policy π outlines the strategy an agent employs to decide actions based on the current state. Policies are categorized into deterministic and stochastic forms.

2.1.3.1 Deterministic Policies

A deterministic policy π is defined as a function $\pi : S \to A$, where for each state $s \in S$, there is a corresponding action $a \in A$ such that $a = \pi(s)$. This formulation implies a fixed action selection for each state, expressed mathematically as:

$$a = \pi(s) \tag{2.1}$$

$$\pi : S \to A \tag{2.2}$$

2.1.3.2 Stochastic Policies

A stochastic policy, in contrast, is represented as a probability distribution over actions for each state, denoted as $\pi(a|s)$, indicating the probability of selecting action a when in state s. Formally, for each state s, $\pi(\cdot|s)$ is a probability distribution over the action space A, described as:

$$\pi(a|s) = P(A = a|S = s) \tag{2.3}$$

$$\pi : S \times A \to [0, 1], \quad \text{where } S \text{ is the state space, } A \text{ is the action space.} \tag{2.4}$$

This allows the agent to explore actions in a probabilistic manner, which can be particularly beneficial in complex or uncertain environments.

Deterministic Vs Stochastic approaches

The differentiation between *deterministic* and *stochastic* algorithms hinges on the nature of the policies or environments they interact with or learn from.

Deterministic Algorithms

- **Policy Determinism:** Deterministic RL algorithms adopt a deterministic policy, implying that for a specific state, the policy invariably designates the same action i.e the outcome of the action, in terms of the next state, might still be stochastic due to the environment's dynamics, but the choice of action is fixed for any given state. This eradicates randomness in the action selection phase during policy execution.

- **Environment Assumption:** These algorithms often presuppose a deterministic environment, where the subsequent state and reward are precisely determined by the current state and the action executed. Notwithstanding, deterministic policies are also applicable in stochastic environments, aiming to identify the optimal action to maximize expected rewards.

- **Examples:** Deterministic Policy Gradient (DPG), Deep Deterministic Policy Gradient (DDPG), and Twin Delayed DDPG (TD3) exemplify deterministic RL algorithms, focusing on learning a deterministic policy mapping states to actions $\pi : S \rightarrow A$.

- **Application Scenarios:** Primarily suitable for environments where outcomes are predictable and consistent for the same actions and states. Robotics requiring precise control often benefit from deterministic approaches.

Stochastic Algorithms

- **Policy Randomness:** Stochastic RL algorithms develop a stochastic policy, denoting that for a given state, the policy provides probabilities for each potential action. The action is subsequently selected based on these probabilities, introducing randomness into action selection.

- **Environment Assumption:** These algorithms adeptly manage stochastic environments, where transition dynamics (next state and reward given a state and action) are probabilistic. However, they're also deployable in deterministic environments $\pi : SxA \rightarrow [0, 1]$.

- **Examples:** Stochastic Policy Gradient, Proximal Policy Optimization (PPO), and Soft Actor-Critic (SAC) are instances of stochastic RL algorithms. These methods learn policies that probabilistically determine actions, facilitating more effective exploration of the environment and averting local optima pitfalls.

- **Application Scenarios:** Optimal for environments where uncertainty or variability in action outcomes is inherent. This includes complex, dynamic environments like financial markets or adaptive game AI, where exploration and risk management are beneficial.

In our study, we have used PPO which is Stochastic Algorithm. PPO introduces a clipped objective function that prevents excessively large updates to the policy. This clipping mechanism is crucial for maintaining the stability of the learning process over time, reducing the likelihood of catastrophic performance drops after updates (explained in detail in later section). The algorithm's stability and robustness make it suitable for a broad array of tasks, including complex environments where maintaining policy stability is challenging. This is one of the reason we have used it in our study. PPO is the current SotA in Reinforcement Learning in terms of sample efficiency, ease of implementation and quality of results. PPO scales well with high-dimensional state and action spaces, making it applicable to a range of problems, from robotic control. PPO has shown good generalization capabilities across tasks. It can be effectively used in different domains, from simulated environments to real-world applications, by appropriately adjusting its parameters.

2.1.4 Trajectories

In reinforcement learning, a trajectory, (or sometimes called an episode), denoted as τ, is a sequence of states, actions, and rewards observed by an agent throughout an episode. It is formally defined as:

$$\tau = (s_0, a_0, r_1, s_1, a_1, r_2, \ldots, s_{T-1}, a_{T-1}, r_T, s_T) \tag{2.5}$$

where s_t denotes the state at time step t, a_t represents the action taken at time step t, and r_{t+1} is the reward received after executing action a_t in state s_t. The term T refers to the terminal time step of the episode, which can be finite for episodic tasks or infinite for continuing tasks.

2.1.5 Reward and Return

The return, G_t, is the total discounted reward received by the agent from time step t onwards. It can be formulated in several ways depending on the task:

1. **Cumulative Return** for episodic tasks is given by:

$$G_t = \sum_{k=t+1}^{T} r_k \tag{2.6}$$

2. **Discounted Return**, incorporating a discount factor γ to weigh immediate rewards more heavily than future rewards, is defined as:

$$G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \tag{2.7}$$

where $0 \leq \gamma \leq 1$.

2.1.6 Reinforcement Learning Optimization Problem

The objective in reinforcement learning is to discover a policy π^* that maximizes the expected return from the initial state distribution over all possible trajectories. This optimization problem can be formally stated as:

$$\pi^* = \arg\max_{\pi} \mathbb{E}_{\tau\sim\pi}[G_0] \tag{2.8}$$

where $\mathbb{E}_{\tau\sim\pi}[G_0]$ represents the expected return of following policy π from the initial state over all trajectories.

2.1.7 Value Functions

Value functions are fundamental in reinforcement learning as they quantify the expected return for states or state-action pairs under specific policies. These functions are pivotal in policy evaluation and improvement. There are four primary value functions of interest:

1. **The On-Policy Value Function**, $V^\pi(s)$, which provides the expected return when starting in state s and consistently following policy π:

$$V^\pi(s) = \mathbb{E}_{\tau\sim\pi}\left[R(\tau) \mid S_0 = s\right] \tag{2.9}$$

   where $R(\tau)$ denotes the return of the trajectory τ.

2. **The On-Policy Action-Value Function**, $Q^\pi(s,a)$, which indicates the expected return upon taking an action a in state s under policy π, and thereafter always following π:

$$Q^\pi(s,a) = \mathbb{E}_{\tau\sim\pi}\left[R(\tau) \mid S_0 = s, A_0 = a\right] \tag{2.10}$$

   This function is particularly useful for evaluating the potential of actions that are not necessarily derived from policy π.

3. **The Optimal Value Function**, $V^*(s)$, which gives the maximum expected return achievable from state s across all policies:

$$V^*(s) = \max_{\pi} \mathbb{E}_{\tau\sim\pi}\left[R(\tau) \mid S_0 = s\right] \tag{2.11}$$

   This function represents the best possible outcome from any given state, serving as a benchmark for policy performance.

4. **The Optimal Action-Value Function**, $Q^*(s,a)$, which provides the maximum expected return for taking action a in state s when followed by the optimal policy:

$$Q^*(s,a) = \max_{\pi} \mathbb{E}_{\tau\sim\pi}\left[R(\tau) \mid S_0 = s, A_0 = a\right] \tag{2.12}$$

   $Q^*(s,a)$ is critical in many RL algorithms as it directly informs the decision-making process by indicating the quality of taking action a in state s.

2.1.8 The Optimal Q-Function and the Optimal Action

The Optimal Q-Function, denoted as $Q^*(s, a)$, represents the expected return for choosing an action a in state s and subsequently adhering to the optimal policy. The Bellman optimality equation for $Q^*(s, a)$ is:

$$Q^*(s, a) = \mathbb{E}\left[R_{t+1} + \gamma \max_{a'} Q^*(S_{t+1}, a') \mid S_t = s, A_t = a\right] \tag{2.13}$$

This equation captures the essence of the decision-making process at the heart of reinforcement learning, where the goal is to maximize the expected return.

The Optimal Action, denoted $a^*(s)$, is the action that maximizes the expected return from state s:

$$a^*(s) = \arg\max_a Q^*(s, a) \tag{2.14}$$

The optimal policy π^* in state s will select actions according to this principle, potentially choosing randomly among multiple actions that yield the same maximum value from $Q^*(s, a)$. This situation implies that there could be several optimal actions for a given state:

$$\pi^*(s) = \begin{cases} \text{a randomly selected action from } \arg\max_a Q^*(s, a), & \text{if multiple maxima exist} \\ a^*(s), & \text{if a unique maximum exists} \end{cases} \tag{2.15}$$

Note that while there may be multiple actions that maximize $Q^*(s, a)$, and thus the optimal policy could be stochastic when selecting among these, there always exists a deterministic policy that will select an optimal action.

2.1.9 Bellman Equations

Bellman equations are fundamental to reinforcement learning, expressing the recursive relationships inherent in value functions. These equations reflect the principle that the value of a state (or state-action pair) is based on the rewards expected in the future, adjusted by the value of the subsequent states. They ensure self-consistency in value estimates and are key to both policy evaluation and policy improvement.

2.1.9.1 Bellman Expectation Equations

The Bellman expectation equations for the on-policy value functions under a policy π are given by:

$$V^\pi(s) = \mathbb{E}_{a\sim\pi, s'\sim P}\left[r(s, a) + \gamma V^\pi(s')\right], \tag{2.16}$$

for the state-value function, and

$$Q^\pi(s, a) = \mathbb{E}_{s'\sim P}\left[r(s, a) + \gamma \mathbb{E}_{a'\sim\pi}\left[Q^\pi(s', a')\right]\right], \tag{2.17}$$

for the action-value function, where $s' \sim P(\cdot|s,a)$ denotes that the next state s' is sampled according to the environment's transition dynamics given current state s and action a, and $a' \sim \pi(\cdot|s')$ denotes that the next action a' is chosen according to policy π in the next state s'.

2.1.9.2 Bellman Optimality Equations

The Bellman optimality equations for the optimal value functions, which do not condition on a particular policy, are:

$$V^*(s) = \max_a \mathbb{E}_{s' \sim P} \left[r(s,a) + \gamma V^*(s') \right], \tag{2.18}$$

for the optimal state-value function, and

$$Q^*(s,a) = \mathbb{E}_{s' \sim P} \left[r(s,a) + \gamma \max_{a'} Q^*(s',a') \right], \tag{2.19}$$

for the optimal action-value function. The key distinction between the Bellman equations for on-policy and optimal value functions is the maximization over actions, which reflects the goal of selecting the action leading to the highest value.

Bellman Backup The term "Bellman backup" refers to the operation performed on the right-hand side of the Bellman equation. For a state-value function, this is the expected return for a state, including the immediate reward and the discounted value of the next state. For the action-value function, it includes the expected immediate reward plus the discounted value of the subsequent state after taking the best possible action.

2.1.10 Advantage Functions

Advantage functions play a critical role in reinforcement learning (RL) by quantifying how much better it is to take a particular action compared to the average action at a given state under a certain policy. The concept of the advantage function, $A^\pi(s,a)$, helps in understanding the relative utility of each action without needing to consider the absolute quality of that action.

The advantage function for a policy π, denoted as $A^\pi(s,a)$, is defined as the difference between the action-value function $Q^\pi(s,a)$ and the state-value function $V^\pi(s)$. This difference tells us how much more (or less) beneficial it is to take action a in state s over randomly selecting an action according to $\pi(\cdot|s)$, assuming the policy π is followed thereafter. Mathematically, the advantage function is expressed as:

$$A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s) \tag{2.20}$$

This measure is particularly useful in policy gradient methods where it helps to estimate the efficacy of actions relative to a baseline, enabling more informed decisions for policy improvement.

2.2 Taxonomy to RL Algorithms

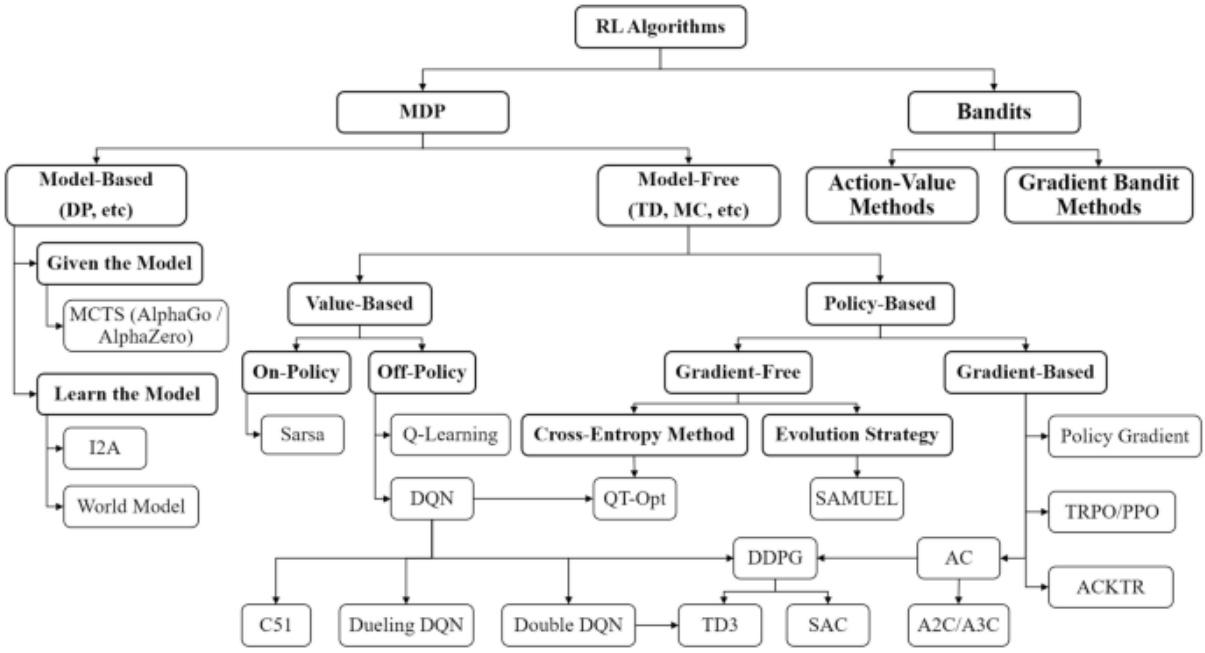


Figure 2.2: **Taxonomy to RL Algorithms**: An informative yet non-exhaustive taxonomy of algorithms in contemporary RL, image ref [2]

2.3 Model-Free vs Model-Based Reinforcement Learning

One of the principal distinctions in reinforcement learning (RL) is whether the algorithm utilizes a model of the environment. Model-based RL methods incorporate a representation that predicts state transitions and rewards, allowing for planning and foresight. This approach can substantially improve sample efficiency over model-free methods, as demonstrated by the success of algorithms such as AlphaZero [47]. However, acquiring a ground-truth model is often infeasible, necessitating the learning of the model from environmental interactions, which introduces challenges such as model bias and potential sub-optimal performance in the actual environment.

Model-free methods, in contrast, learn from direct interaction without an explicit environmental model. These methods can be broadly categorized into policy optimization and Q-learning approaches, each with distinct characteristics and use cases.

2.3.1 Policy Optimization in Model-Free RL

Policy optimization directly adjusts the policy's parameters to maximize the expected return. The policy, expressed explicitly as $\pi_\theta(a|s)$, is optimized either by gradient ascent on the performance objec-

tive $J(\pi_\theta)$, or through surrogate objectives for more stable updates. Almost always, this optimization is performed on-policy, which means that each update utilizes only data gathered while the policy is in effect at the time. Notable algorithms include A2C/A3C and Proximal Policy Optimization (PPO):

- A2C/A3C [48] perform gradient ascent to maximize the expected return directly.
- PPO [49] optimizes a surrogate objective that provides a conservative estimate for policy updates.

2.3.2 Q-Learning in Model-Free RL

Q-learning focuses on learning an approximate Q-function, $Q_\theta(s, a)$, based on the Bellman equation. This method often utilizes off-policy data, enabling the efficient reuse of past experience. Examples of Q-learning algorithms are Deep Q-Networks (DQN) [50] and C51 [51], which models the return distribution.

2.3.3 Trade-offs Between Policy Optimization and Q-Learning

The primary strength of policy optimization methods lies in their principled approach to directly optimizing the desired objective. This direct optimization strategy tends to enhance the stability and reliability of the learning process. Formally, policy optimization techniques focus on adjusting a policy, $\pi(a|s;\theta)$, to maximize the expected return by directly influencing the agent's actions in the environment.

In contrast, Q-learning methods approach the optimization problem indirectly. They aim to train a parameterized action-value function, $Q_\theta(s, a)$, to satisfy a self-consistency Bellman equation. This indirect approach introduces potential failure modes, rendering Q-learning techniques generally less stable than their policy optimization counterparts.

However, when Q-learning methods are effective, they offer a substantial advantage in terms of sample efficiency. This efficiency stems from their ability to reuse data more effectively, as opposed to policy optimization techniques which typically require fresh samples to update the policy. The ability to efficiently utilize data makes Q-learning particularly appealing in environments where acquiring new samples is costly or time-consuming.

2.3.4 Interpolating Between Policy Optimization and Q-Learning

Interpolation between policy optimization and Q-learning has led to algorithms that balance the two approaches, such as Deep Deterministic Policy Gradients (DDPG) and Soft Actor-Critic (SAC), which combine elements of both to improve learning stability and performance.

- DDPG [52] learns a deterministic policy alongside a Q-function using each to improve the other.
- SAC [53] employs stochastic policies with entropy regularization to enhance exploration and learning stability.

2.4 DDPG and TD3

Deep Deterministic Policy Gradient (DDPG) algorithm is a reinforcement learning approach designed for environments with continuous action spaces, learning a Q-function and a policy concurrently through off-policy data and the Bellman equation. DDPG builds on Q-learning principles, aiming to identify the optimal action-value function to dictate the best action in a given state. It incorporates a replay buffer to mitigate issues from correlated data and employs target networks with Polyak averaging [54] for more stable learning, enhancing Q-learning's applicability to continuous domains by using modern deep learning techniques.

Twin Delayed DDPG (TD3) advances DDPG by addressing its susceptibility to hyperparameter sensitivity and overestimation bias through three main improvements: Clipped Double-Q Learning, which reduces overestimation by taking the minimum of two separate Q-function estimates; Delayed Policy Updates to lessen policy update variance; and Target Policy Smoothing, adding clipped noise to actions to avoid overfitting to Q-function peaks. These adjustments make TD3 more robust and effective, significantly outperforming DDPG in various settings. Both DDPG and TD3 are off-policy and support continuous action spaces, but TD3 further enriches exploration by introducing uncorrelated noise to policy actions during training, balancing the exploration-exploitation trade-off more effectively.

2.5 Proximal Policy Optimization (PPO)

Proximal Policy Optimization (PPO) is a policy gradient method for reinforcement learning that has gained popularity due to its effectiveness and simplicity. It is designed to improve upon its predecessor, Trust Region Policy Optimization (TRPO), by seeking to take the largest possible improvement step on a policy using the currently available data without causing a performance collapse, a challenge that TRPO addresses with a complex second-order method.

PPO simplifies this approach and operates by maintaining a balance between exploration and exploitation. It does this by utilizing a clipped objective function that prevents the new policy from deviating too far from the old policy. This ensures that the updates are stable and the policy improves in a more controlled manner.

There are two primary variants of PPO: PPO-Penalty and PPO-Clip.

- **PPO-Penalty:** This variant solves a KL-constrained update like TRPO but introduces a penalty on the KL-divergence in the objective function instead of a hard constraint. The penalty coefficient is adjusted throughout training for appropriate scaling.

- **PPO-Clip:** Unlike PPO-Penalty, PPO-Clip does not have a KL-divergence term in the objective function. It uses a clipping mechanism to modify the objective function, which removes incentives for moving too far from the old policy.

PPO is an on-policy algorithm that can handle environments with either discrete or continuous action spaces.

Key Equations

PPO-Clip updates policies by solving the following optimization problem:

$$\theta_{k+1} = \arg\max_{\theta} \mathbb{E}_{s,a\sim\pi_{\theta_k}} \left[L(s, a, \theta_k, \theta) \right], \tag{2.21}$$

where the objective function L is defined as:

$$L(s, a, \theta_k, \theta) = \min \left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)} A^{\pi_{\theta_k}}(s, a), \text{clip} \left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}, 1 - \epsilon, 1 + \epsilon \right) A^{\pi_{\theta_k}}(s, a) \right). \tag{2.22}$$

In this function, ϵ is a hyperparameter that defines the clipping range and $A^{\pi_{\theta_k}}(s, a)$ is the advantage function, indicating the relative value of taking action a in state s under policy $\pi_k = \pi(\theta_k)$

The clipping function serves as a form of regularization, ensuring that the updated policy does not deviate significantly from the old policy, thus promoting a more stable learning process.

Exploration vs. Exploitation

PPO trains a stochastic policy in an on-policy way, balancing exploration by sampling actions according to the stochastic policy, and exploitation by progressively making the policy less random as it learns to exploit higher rewards. The policy becomes progressively less random over time as it learns to exploit the higher rewards it has found, which could potentially lead to getting trapped in local optima if not managed properly.

Algorithm 1 PPO with Clipping Strategy

Input: initial policy parameters θ_0, initial value function parameters φ_0

for each iteration $k = 0, 1, 2, \ldots$ **do**

Collect trajectories $D_k = \{\tau_i\}$ by executing policy $\pi_k = \pi(\theta_k)$ in the environment.

Calculate rewards-to-go $\hat{R}_t$ for each time step.

Estimate advantages $\hat{A}_t$ using a chosen advantage estimation technique based on the current value function V_{φ_k}.

Optimize the policy parameters by maximizing the clipped objective:

$$\theta_{k+1} = \arg\max_{\theta} \frac{1}{|D_k|\,T} \sum_{\tau \in D_k} \sum_{t=0}^{T} \min\left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)} \hat{A}_t,\ \text{clip}\left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)}, 1-\epsilon, 1+\epsilon \right) \hat{A}_t \right)$$

where ϵ is the clipping hyperparameter.

Update the value function by minimizing the mean-squared error:

$$\varphi_{k+1} = \arg\min_{\varphi} \frac{1}{|D_k|\,T} \sum_{\tau \in D_k} \sum_{t=0}^{T} \left(V_\varphi(s_t) - \hat{R}_t \right)^2$$

end for

Referenced from [49], [55] and `https://spinningup.openai.com/en/latest/algorithms/ppo.html`

2.6 UAV Model

A first-order linear UAV model with position coordinates $[x,\ y]^T$ as output and velocity $[v_x,\ v_y]^T$ as the control input is used here, as given by Eq. (2.23).

$$\dot{x} = v_x, \quad \dot{y} = v_y \tag{2.23}$$

My current work assumes that the altitude z is held constant throughout the flight, constraining the UAV navigation to the XY plane.

2.7 Measurement Noise Model and Denoising Algorithm

The noise used in all of our simulations and experiments is sampled from Gaussian distributions with different combinations of the mean (μ) and standard deviation (σ).

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2} = \mathcal{N}(\mu, \sigma) \tag{2.24}$$

There are several approaches to denoising (reducing the effect of measurement noise) a signal corrupted by Gaussian noise. Our primary focus is on two of them: The Bessel Low Pass Filter [56], and the Kalman Filter [57]

2.7.0.1 Low Pass Filter

A low-pass filter is a filter that passes signals with a frequency lower than a selected cutoff frequency and attenuates signals with frequencies higher than the cutoff frequency. For the denoising aspect of our experiments, we employed a 2^{nd} order Bessel Low Pass Filter, selected for its nuanced balance between phase response and filtering efficacy. The chosen filter, with a cutoff frequency set at 2, is described by the following transfer function as cited from [56]:

$$H(s) = \frac{3}{s^2 + 3s + 3} \tag{2.25}$$

The choice of this LPF was predicated on its proven performance in preserving the temporal integrity of the signal, thus ensuring minimal distortion during the denoising process.

2.7.0.2 Kalman Filter

The Kalman Filter [57] [58] is a recursive algorithm used for estimating the state of a linear dynamic system from a series of noisy measurements. It consists of two main steps: prediction and update.

Prediction

The prediction step projects the current state estimate ahead in time. It also projects the current estimate uncertainty. The equations for the prediction step are:

- **State Prediction Equation:**

$$\hat{x}_{k|k-1} = A\hat{x}_{k-1|k-1} + Bu_k \tag{2.26}$$

  where $\hat{x}_{k|k-1}$ is the predicted state estimate at time k given all measurements up to time $k-1$, A is the state transition model, $\hat{x}_{k-1|k-1}$ is the estimated state at time $k-1$ given all measurements up to time $k-1$, and B is the control input model applied to the control vector u_k.

- **Covariance Prediction Equation:**

$$P_{k|k-1} = AP_{k-1|k-1}A^T + Q \tag{2.27}$$

  where $P_{k|k-1}$ is the predicted state covariance at time k given all measurements up to time $k-1$, $P_{k-1|k-1}$ is the estimated state covariance at time $k-1$ given all measurements upto time $k-1$ and Q is the process noise covariance matrix.

Update

The update step adjusts the predicted estimate by an actual measurement at time k.

- **Kalman Gain:**

$$K_k = P_{k|k-1}C^T(CP_{k|k-1}C^T + R)^{-1} \tag{2.28}$$

  where K_k is the Kalman Gain at time k, C is the observation model, and R is the observation noise covariance matrix.

- **State Update Equation:**

$$\hat{x}_{k|k} = \hat{x}_{k|k-1} + K_k(z_k - C\hat{x}_{k|k-1}) \tag{2.29}$$

  where $\hat{x}_{k|k}$ is the updated state estimate at time k given all measurements up to time k, and z_k is the measurement at time k.

- **Covariance Update Equation:**

$$P_{k|k} = (I - K_kC)P_{k|k-1} \tag{2.30}$$

  where $P_{k|k}$ is the updated state covariance at time k given all measurements up to time k.

2.8 Velocity Obstacle (VO)

The Velocity Obstacle(VO) is used to devise avoidance maneuvers in dynamic environments[59]. It is used here to design a reward function that allows us to train policies capable of avoidance maneuvers in such dynamic environments. The Velocity Obstacle(VO) of a moving circular obstacle B with respect to a moving circular agent A is defined as set of all velocities of A that may lead to a collision between B. That is,

$$A(t) \cap B(t) = \emptyset \quad \text{if } V_A(t) \notin VO(t) \tag{2.31}$$

where A(t) and B(t) are the trajectories of A and B respectively. The region $VO_B \cap VO_H$ in figure 2.3, accounts for this total Velocity obstacle. To consider only immediate collisions, we can modify the VO by subtracting from VO, a set VO_H defined as eq. 2.32

$$VO_h = \{V_A \mid V_A \in VO, \|V_{A,B}\| \leq \frac{d_m}{T_h}\} \tag{2.32}$$

where V_A is velocity of A and $V_{A,B} = V_B - V_A$, the relative velocity of A with respect to B. VO_H is the set of all velocities of A that lead to a collision within a time horizon T_h. In figure 2.3, region $VO_B = VO - VO_H$ represents such a 'reduced' Velocity Obstacle.

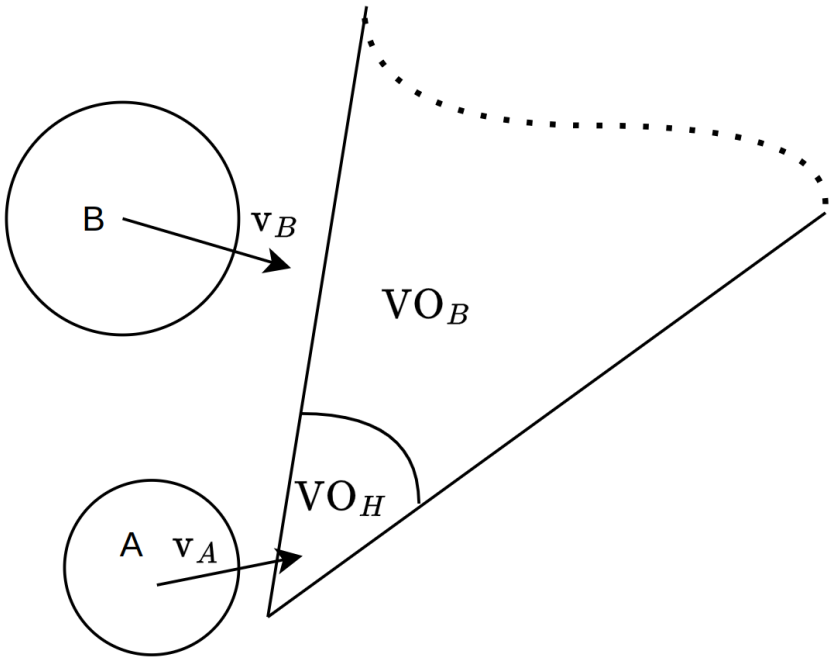


Figure 2.3: VO_B is the velocity obstacle for agent A with respect to object B for some time horizon T.

2.9 Pybullet

PyBullet [60] is a Python module for physics simulation, including robotics, that offers both high performance and accuracy. The reasons to use Pybullet were:

- **Realistic Physics Simulations**: PyBullet is based on the Bullet Physics SDK, which provides highly accurate simulations of real-world physics. This accuracy is crucial for research and RL, especially in robotics, where understanding the dynamics of objects and environments is key to developing effective algorithms and models.

- **Versatility and Flexibility**: It supports a wide range of simulations, from rigid body dynamics to soft body physics. Researchers can simulate complex scenarios, including those with multiple interacting objects, which is often required in robotics and other domains. This versatility allows for a broad range of experiments and studies, making it an invaluable tool in scientific research.

- **Reinforcement Learning Support**: PyBullet is particularly well-suited for RL experiments. It offers a fast and direct way to test algorithms in a controlled, yet realistic environment. Researchers can rapidly prototype and iterate on their RL algorithms, testing them under various conditions and scenarios without the need for real-world testing, which can be costly and time-consuming.

- **Accessibility and Community Support**: Being an open-source project, PyBullet is accessible to everyone, fostering a community of users and contributors who continuously improve and extend its capabilities. This community support also means that researchers can easily find solutions to problems and share their findings with others, advancing the field more rapidly.
- **Integration with Machine Learning Libraries**: PyBullet easily integrates with popular machine learning and data science libraries, such as TensorFlow and PyTorch. This compatibility is essential for researchers in RL, where machine learning algorithms are a core component. It enables seamless transitions between the simulation environment and the learning algorithm, facilitating more efficient development cycles.
- **Benchmark Environments**: PyBullet includes a variety of benchmark environments that are commonly used in RL research. These standardized scenarios allow researchers to compare the performance of their algorithms against established baselines, ensuring that advances in the field are measurable and replicable.

In summary, PyBullet's combination of realistic physics simulation, flexibility, support for reinforcement learning, and integration with machine learning libraries makes it an essential tool in research and RL. Its open-source nature and strong community support further enhance its value, facilitating widespread use and ongoing improvement.

Chapter 3

Navigating UAVs Through Static Environments

In this chapter, we delve into the intricacies of the system architecture designed for navigating Unmanned Aerial Vehicles (UAVs) through static obstacle environments followed by the description of the environment, action space, observation space, reward function, agent and then will cover in detail the effect of noise. The last section of the chapter includes the study for time varying bias for static obstacles.

3.1 System Architecture Overview

This section describes the computational framework for navigating an Unmanned Aerial Vehicle (UAV) through its environment.

The architecture of the methodology followed is shown in Fig. 3.1.

It involves several key steps and components, as outlined below:

- **Current Time Step (t)**: The system operates in discrete time steps, with t indicating the current moment.
- **UAV Position ($\mathbf{x_t}$)**: The current position of the UAV at time step t.
- **Nearest Obstacle Position ($\mathbf{x_{o_t}}$)**: The position of the closest obstacle to the UAV, which must be avoided for successful navigation.
- **Goal Position ($\mathbf{x_{g_t}}$)**: The target location the UAV is attempting to reach.
- **Noisy Position Estimate ($\hat{\mathbf{x}}_\mathbf{t}$)**: To simulate real-world inaccuracies, the UAV's position is perturbed, creating a noisy estimate of its actual position.
- **Denoising and Final Position Estimate ($\tilde{\mathbf{x}}_\mathbf{t}$)**: The noisy position estimate is processed (denoised), considering the previous action A_{t-1} and the previous estimates, to obtain a refined estimate of the UAV's position.

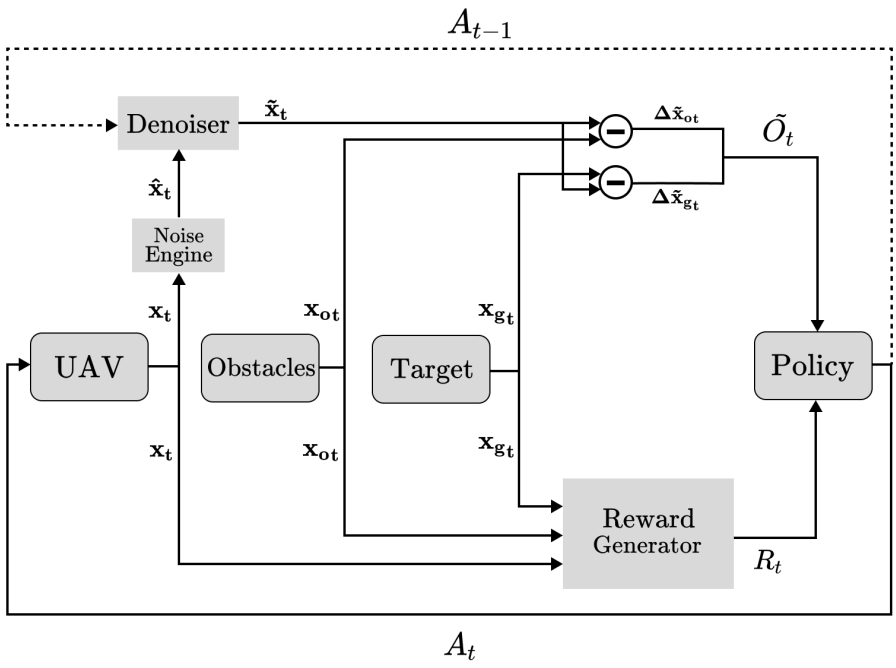


Figure 3.1: **System Architecture**: At the current time step t, we obtain the UAV's position $\mathbf{x_t}$, the position of the nearest obstacle $\mathbf{x_{ot}}$, and the goal position $\mathbf{x_{g_t}}$. $\mathbf{x_t}$ is perturbed to generate the noisy position estimate $\hat{\mathbf{x}}_\mathbf{t}$, which is then denoised along with the previous action A_{t-1} to yield the final position estimate $\tilde{\mathbf{x}}_\mathbf{t}$. The denoised position estimate, along with $\mathbf{x_{ot}}$ and $\mathbf{x_{g_t}}$, is utilized to produce the environment observation $\tilde{O}_t$. The reward R_t is calculated using the true position $\mathbf{x_t}$. The observation and reward are then inputted into the policy, which generates the action to be taken A_t.

- **Environment Observation ($\tilde{O}_t$)**: An observation of the environment is generated using the denoised position estimate, the position of the nearest obstacle, and the goal position. This observation is crucial for the UAV's decision-making process.

- **Reward (R_t)**: The system computes a reward based on the UAV's true position to provide feedback on its navigation performance.

- **Policy and Action (A_t)**: Based on the observation and the reward, the UAV's policy determines the next action to take (A_t), such as adjusting its trajectory or speed to move towards the goal while avoiding obstacles.

This architecture establishes a feedback loop where the UAV continuously updates its position and understanding of the environment, evaluates its progress towards the goal, and adjusts its actions accordingly.

3.1.1 The Environment

The environment consists of multiple obstacles, the target location, and the initial position of the UAV. Each episode commences with the UAV being placed at a random starting position defined as follows:

$$\mathbf{x}_0 = \left[x_{min} + r_{minor}, y_{g_0}, \frac{z_{min} + z_{max}}{2} \right] \tag{3.1}$$

$$y_{g_0} \sim Uniform(y_{min} + r_{minor}, y_{max} - r_{minor}) \tag{3.2}$$

Next, the goal position is set as

$$\mathbf{x}_g = \left[x_{max} - r_{minor}, \frac{y_{min} + y_{max}}{2}, \frac{z_{min} + z_{max}}{2} \right] \tag{3.3}$$

In this setup, m_{min} and m_{max} denote the minimum and maximum limits of the environment along the respective axes ($m \in x, y, z$), ensuring the goal location is reachable while maintaining a consistent challenge level. The positioning also ensures the UAV begins within the bounds of the environment, avoiding any immediate out-of-bounds penalties, with r_{minor} acting as a safety margin to prevent boundary infringements.

3.1.

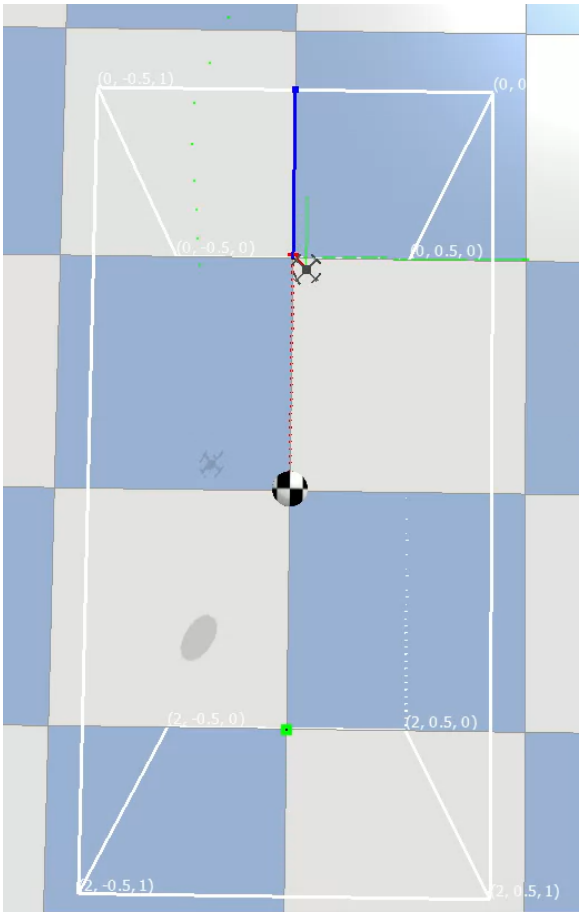


Figure 3.2: **Simulated Environment**: This is a Pybullet simulated environment with x_{min} as 0, x_{max} as 2, y_{min} as -0.5, y_{max} as 0.5 and z axis fixed at 0.5 and r_{minor} is 0.15 (all values in meter)

The environment's complexity is further augmented by the introduction of obstacles. The process begins with the determination of the total number of obstacles, constrained within a predefined upper and lower limit. These obstacles are then systematically distributed throughout the environment. Their placement along the x-axis follows a uniform distribution, ensuring an even spatial spread. Along the y-axis, a normal distribution is employed to simulate varying densities and clustering of obstacles, reflecting more realistic scenarios. The altitude for each obstacle is set at a constant, determined by the midpoint of the z-axis range. And the obstacles in the study are considered to be spherical with diameter d_{obs}.

Note that we refer to the Environment with Static Obstacles (SO) with "Environment 1" and Dynamic Obstacles(DO) with "Environment 2" in our study.
Chapter 3 consists with study with Static Obstacles, (Environment 1) and Chapter 4 consists with the study with Dynamic Obstacles (Environment 2).

3.1.1.1 Observation Space

The observation space within the environment is conceptualized as a refined, continuous representation of the current state, capturing only the essential information necessary for the agent to devise an effective policy. This concept is mathematically formalized as:

$$O_t = \begin{bmatrix} \Delta x_{g_t} & \Delta y_{g_t} & \Delta x_{ot} & \Delta y_{ot} \end{bmatrix} \tag{3.4}$$

Here, Δx_{g_t} and Δy_{g_t} denote the distances to the goal along the x and y axes at time t, respectively. Similarly, Δx_{ot} and Δy_{ot} represent the distances to the closest obstacle or wall along the x and y axes, respectively.

Due to the presence of localization noise, the actual positions $(\hat{x}, \hat{y})$ are influenced by noise, leading to perturbations in the observed state as described by:

$$\hat{O}_t = \begin{bmatrix} \Delta \hat{x_{g_t}} & \Delta \hat{y_{g_t}} & \Delta \hat{x_{ot}} & \Delta \hat{y_{ot}} \end{bmatrix} \tag{3.5}$$

Upon the application of a denoiser, the noisy position estimates $(\hat{x}, \hat{y})$ are denoised to yield positions $(\tilde{x}, \tilde{y})$, resulting in the denoised form of the observation:

$$\tilde{O}_t = \begin{bmatrix} \Delta \tilde{x_{g_t}} & \Delta \tilde{y_{g_t}} & \Delta \tilde{x_{ot}} & \Delta \tilde{y_{ot}} \end{bmatrix} \tag{3.6}$$

The denoised observation, $\tilde{O}_t$, is the feedback provided by the environment to the agent at each timestep t, facilitating the agent's learning and navigation through the task environment.

3.1.1.2 Action Space

The action space within our environment is designed as continuous, accommodating any action vector delineated by:

$$A = \begin{bmatrix} v_x & v_y & v_{mag} \end{bmatrix} \tag{3.7}$$

In this, v_x and v_y represent the velocities of the Unmanned Aerial Vehicle (UAV) in the x and y axes, correspondingly. The term v_{mag} denotes the overall magnitude of the velocity vector, which is a crucial component for determining the UAV's speed.

The methodology to derive the velocity command from the given action vector is as follows:

$$\vec{v} = \frac{\begin{bmatrix} v_x & v_y \end{bmatrix}}{\sqrt{v_x^2 + v_y^2}} \times v_{mag} \tag{3.8}$$

This equation normalizes the velocity components (v_x, v_y) to a unit vector, which is then scaled by the magnitude v_{mag}. This process effectively translates the action vector into a precise velocity command for the UAV, ensuring that its movement is accurately governed by the inputs within the continuous action space.

3.1.1.3 Reward Function

The environment employs a densely structured reward function designed to offer the agent continuous feedback, essential for its learning and adaptation. The reward allocated at each time step t is defined as:

$$R_t = \begin{cases} R_s, & \text{if distance to target} < \epsilon_{\text{success}} \\ R_f, & \text{if collision with an obstacle occurs} \\ R_f, & \text{if the agent is out of time} \\ (-R_d \left\| \begin{bmatrix} \Delta x_t & \Delta y_t \end{bmatrix} \right\| - R_{major} 1_{major} - R_{minor} 1_{minor}) & \text{otherwise} \end{cases} \tag{3.9}$$

where:

- $R_s > 0$ is the reward for successfully reaching the target.
- $R_f < 0$ signifies the penalty for either colliding with an obstacle or failing to reach the target within the stipulated time.
- $R_d > 0$ is the distance penalty coefficient, encouraging the agent to minimize the distance to the target.
- $R_{\text{major}} > 0$ and $R_{\text{minor}} > 0$ are penalties assigned for breaching major and minor safety boundaries, respectively.
- $\|[\Delta x_t, \Delta y_t]\|$ denote the euclidean distance between the agent's current position and the traget position.
- 1_{major} and 1_{minor} are indicator functions that are activated when the respective safety boundaries are violated.

An agent is considered "out of time" if it has not achieved its objective or encountered an obstacle within a predefined timeframe. This reward structure ensures the agent is continuously guided towards optimizing its path to the target, efficiently minimizing travel distance and avoiding safety boundary breaches.

Additionally, graphical representations of training outcomes, such as the progression of mean rewards and episode lengths, offer valuable insights into the effectiveness of different training policies 3.4.

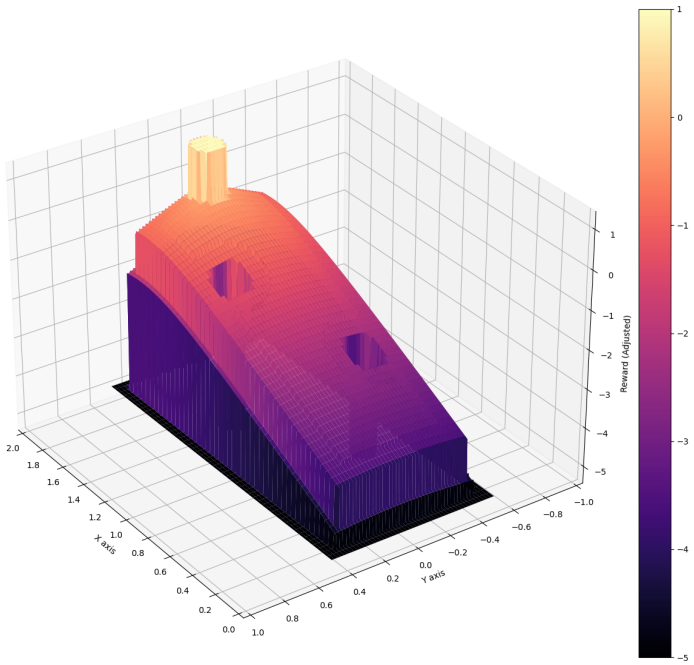


Figure 3.3: **Normalized Reward Function plot for Static Obstacle Environment (Environment 1) :** You can see that the reward continuously increases as we go closer to the goal point, then suddenly increases within the success region of the goal. The two sudden drops in reward mark the presence of the obstacles, indicating a bad reward being given for collision with them. The reward also plumates at regions close to the boundaries.

3.1.2 The Agent

The agent in our study is based on the Proximal Policy Optimization (PPO) algorithm, utilizing the implementation from **stablebaselines3**[61]. The architecture comprises two main components: the actor and the critic, both of which receive the observation from the environment as input. The critic is tasked with learning the state value function, outputting the perceived value of the input state. Conversely, the actor focuses on policy learning, aiming to predict the mean value for each element in the action vector based on the given state. An action value is then sampled using this mean from a Gaussian distribution.

It is important to highlight that the primary goal of our research is not to develop the most efficient policy for navigating the environment. Instead, we aim to investigate the impact of noise on a policy that demonstrates an acceptable success rate. For this purpose, we have selected a vanilla PPO implementation from **stablebaselines3** without any modifications for our experiments.

3.1.3 The Effects of Noise

The cornerstone of our investigation revolves around the impact of observation space noise on policy performance. Initially, multiple policies were trained across varying levels of unbiased noise to scrutinize their training outcomes, including mean rewards and mean episode durations over a set of 100 episodes. Subsequent to establishing baseline performances in a noise-free environment ($\mu = 0, \sigma = 0$), these policies were then assessed under different noise conditions:

1. Unbiased noise ($\mu = 0, \sigma \neq 0$)
2. Bias-only noise ($\mu \neq 0, \sigma = 0$)
3. Biased noise ($\mu \neq 0, \sigma \neq 0$)

While the effects of unbiased noise with unknown standard deviation (σ) can be mitigated quite effectively with the use of denoisers, as shown by our results in Section 3.2, the same cannot be said about bias-only noise and biased noise with unknown mean (μ).

Through our experimental analysis, it emerged that policies conditioned in the aforementioned manner exhibit superior performance under conditions of biased noise compared to bias-only noise. This suggests that the adverse impact of bias-only noise on policy performance can be alleviated by superimposing a carefully injection of unbiased noise onto the biased localization estimates.

Again, it is important to note here that we am not treating biased noise in the same way as the other two noises; rather, we treat it as two separate noises — the first is the biased noise with none-to-low variance plaguing sensor readings, which cannot be fixed by using a filter, and the second is some unbiased noise that we inject into the existing sensor noise to improve the performance of the policy.

3.1.4 Sim-to-Real Transfer

In our experimental setup, we leverage the Crazyflie 2.1 quadcopter, with motion capture markers to facilitate real-time, high-precision localization. This setup enables us to exercise comprehensive control over the perturbations introduced into the observation data. The localization data, captured via motion capture technology, is relayed to our computational infrastructure. Here, it undergoes a process of intentional corruption with simulated noise, followed by a denoising procedure when deemed necessary, before the derivation of the observation vector. This vector is subsequently fed into the policy algorithm, which, in turn, generates the requisite action commands.

These action commands are translated into velocity instructions that the UAV is programmed to execute, marking the completion of a single timestep in our experimental framework. An episode is classified as successful upon the UAV's arrival within a predefined proximity to the designated target. Notably, the transition from simulation to real-world application of the trained agent network onto the UAV is achieved without necessitating any modifications to the training regime.

This seamless sim-to-real transfer underscores the robustness of our training methodology, evidencing the practical applicability of our research in real-world scenarios. It demonstrates the commitment to bridging the gap between theoretical models and their tangible implementation in our study.

3.2 Results

This section delves into the experimental outcomes garnered from both the training phase and the evaluative sessions conducted in simulated settings. For Physical settings, we have another dedicated chapter later. The training was executed on an RTX 2080 Ti GPU, with the entire process spanning approximately 5 hours for a comprehensive 5 million timesteps. Subsequent evaluation of each policy, encompassing 1000 episodes across various configurations of μ, σ, and the presence or absence of a denoiser, was completed within an estimated timeframe of 10 minutes.

Details pertaining to the specific environmental variables deployed during these experiments are systematically catalogued in Table 4.1.

Variable Name	**Value**
R_s	1000
R_f	-1000
R_d	4
R_{major}	5
R_{minor}	1
r_{major}	0.1 (in m)
r_{minor}	0.2 (in m)
d_{obs}	0.1 (in m)
$\epsilon_{success}$	0.1 (in m)

Table 3.1: **Environment Variables: Chapter 3**

3.2.1 Training

Within our research, we embarked on the training of three distinct policies under varied conditions of unbiased noise to compare the effects of noise added to the observation during training. The configurations were as follows:

1. **Policy 1**: No Noise ($\mu = 0$, $\sigma = 0$)

2. **Policy 2**: Low Noise ($\mu = 0$, $\sigma = 0.1$)

3. **Policy 3**: High Noise ($\mu = 0$, $\sigma = 1.0$)

For each policy, the training regimen extended over 5 million timesteps within environments characterized by 1 to 3 randomly positioned obstacles at the commencement of each episode. The selection of policy hyperparameters, as delineated in Table 4.2, was informed by extensive preliminary experimentation to optimize learning efficacy.

Hyperparameter	**Value**
Actor Layers	$64 \times 64 \times 3$
Critic Layers	$64 \times 64 \times 1$
Output Activation Function	tanh
Optimizer	Adam

Table 3.2: **Policy Training Hyperparameters: Chapter 3**

The evaluation of the training quality for these policies hinged on two pivotal metrics: mean reward over the last 100 episodes and mean episode duration over the last 100 episodes. These criteria served not only as indicators of policy robustness but also facilitated a comparative analysis of policy adaptability across varying noise levels.

3.2.1.1 Mean Reward Analysis

An examination of Figure 3.4(a) reveals a progressive increase in the mean reward for all three policies throughout the duration of the training. This upward trend substantiates the effective acquisition of the targeted behaviors, notably obstacle avoidance and optimal target acquisition with minimal steps.

Remarkably, Policy 2, represented in orange, achieves the highest mean reward, slightly surpassing Policy 1. This superior performance can be attributed to the introduction of a minimal level of noise ($\sigma = 0.1$) during its training phase. This strategic insertion of noise facilitated enhanced exploration capabilities, thereby refining the policy's effectiveness. Conversely, Policy 3, depicted in green, exhibits the least favorable outcomes. The substantial variance ($\sigma = 1.0$) associated with this policy significantly impedes its ability to discern patterns within its observations, resulting in a comparatively lower mean reward.

This analysis underscores the delicate balance between noise introduction and policy performance. A moderate amount of noise, as demonstrated by Policy 2, can serve as a catalyst for improved policy exploration and learning efficiency. However, excessive noise, as with Policy 3, proves to be detrimental, overwhelming the policy's learning mechanism and hindering its performance.

3.2.1.2 Analysis of Episode Duration

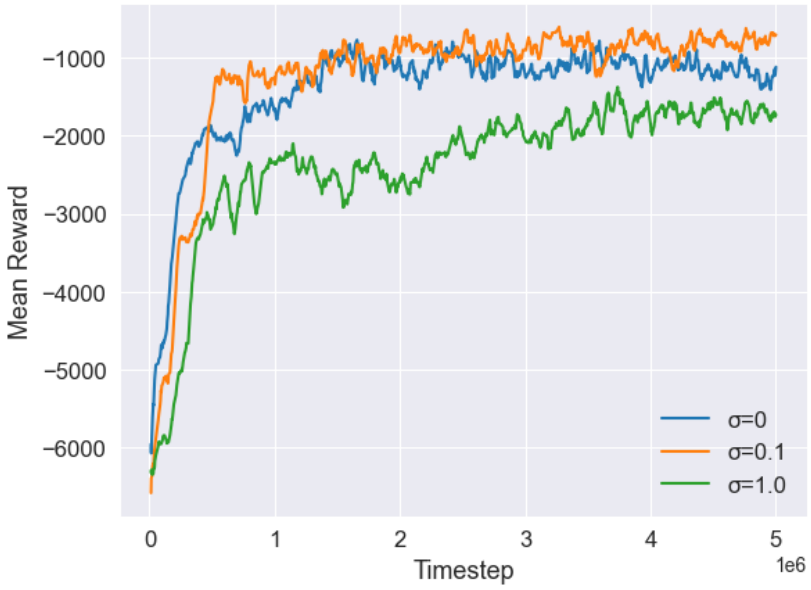
The Figure 3.4(b) illustrates a notable reduction in the mean duration of episodes as training progresses. This trend is a direct consequence of the training environment's design, where each non-terminal timestep incurs a negative reward. Such a penalty mechanism inherently motivates the agent to expedite the completion of an episode, thereby minimizing the accumulation of negative rewards.

This strategic imposition of negative rewards for prolonged episode durations serves a dual purpose. It instills a sense of urgency in the agent, compelling it to seek the most efficient path to the episode's objective. The observed decrease in episode duration is a testament to the effectiveness of this approach, reflecting the agent's increasing proficiency in navigating the environment swiftly and effectively.

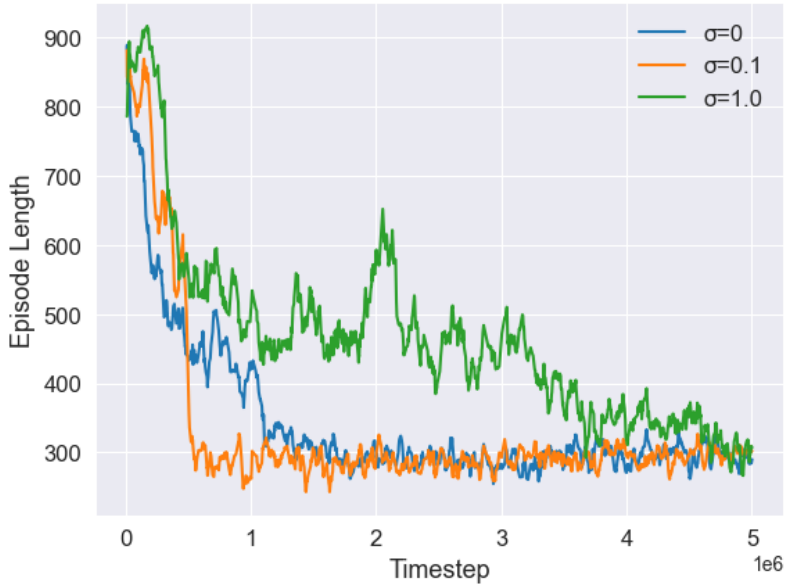
3.2.2 Evaluations of Trained Policies

My rigorous evaluation of the trained policies was conducted across a spectrum of environments, each characterized by distinct levels of bias and noise variance. These assessments were meticulously designed to include both scenarios: with and without the implementation of denoising techniques, namely the Low Pass Filter (LPF) and the Kalman Filter (KF). The environments for these evaluations were dynamically generated, featuring zero to three obstacles randomly placed at the onset of each episode. This approach, encompassing 1000 episodes per evaluation, was strategically chosen to ensure a high degree of statistical reliability in our findings.

For the denoising aspect of our experiments, we employed a 2^{nd} order Bessel Low Pass Filter, selected for its nuanced balance between phase response and filtering efficacy. The chosen filter, with a cutoff frequency set at 2, is described by the following transfer function as cited from [56]:



(a)



(b)

Figure 3.4: **Policy Training Results**: Figure (a) depicts the mean reward progression, and figure (b) illustrates the episode length variation through the training period. Blue, orange, and green plots correspond to Policy 1 (trained without noise), Policy 2 (trained with moderate noise $\mu = 0, \sigma = 0.1$), and Policy 3 (trained with significant noise $\mu = 0, \sigma = 1.0$), respectively.

$$H(s) = \frac{3}{s^2 + 3s + 3} \tag{3.10}$$

This choice of LPF was predicated on its proven performance in preserving the temporal integrity of the signal, thus ensuring minimal distortion during the denoising process. The implementation of this filter, alongside the Kalman Filter, provides a comprehensive overview of the effects of different denoising strategies on the performance of policies subjected to various noise profiles.

Metric Values

For the base metric of **Policy 1** of unbiased noise with a mean (μ) of 0 and a standard deviation (σ) of 0.0, when no denoiser is applied, the success rate is 61.50%, the collision rate is 38.50%, the mean reward is -929.98, and the mean episode length is 298.123.

When a Low-Pass Filter (LPF) is used as the denoiser under the same conditions, the success rate slightly decreases to 60.20%, while the collision rate increases to 39.80%. The mean reward also decreases to -957.45, and the mean episode length increases to 300.106.

Using a Kalman Filter (KF) as the denoiser results in further reduction of the success rate to 58.90%, with an increase in the collision rate to 41.10%. The mean reward decreases more significantly to -982.55, while the mean episode length is 297.165.

For the base metric of **Policy 2** of unbiased noise with a mean (μ) of 0 and a standard deviation (σ) of 0.0, when no denoiser is applied, the success rate is 71.60%, the collision rate is 28.40%, the mean reward is -747.05, and the mean episode length is 298.034.

When a Low-Pass Filter (LPF) is used as the denoiser under the same conditions, the success rate decreases to 69.40%, while the collision rate increases to 30.60%. The mean reward also decreases to -795.71, and the mean episode length increases to 299.356.

Using a Kalman Filter (KF) as the denoiser results in an improvement in the success rate to 74.80%, with a reduction in the collision rate to 25.20%. The mean reward improves significantly to -690.14, while the mean episode length is 299.124.

These metrics highlight the performance differences when applying different denoisers in different policies under the same conditions.

3.2.2.1 Unbiased Noise Evaluation

In this section, we delve into the comprehensive analysis of our three meticulously trained policies under the influence of unbiased noise, characterized by a mean (μ) of 0 and a standard deviation (σ) ranging from 0 to 3.0, with increments of 0.1. The scenario where $\sigma = 0$ serves as the *baseline* benchmark for subsequent comparisons.

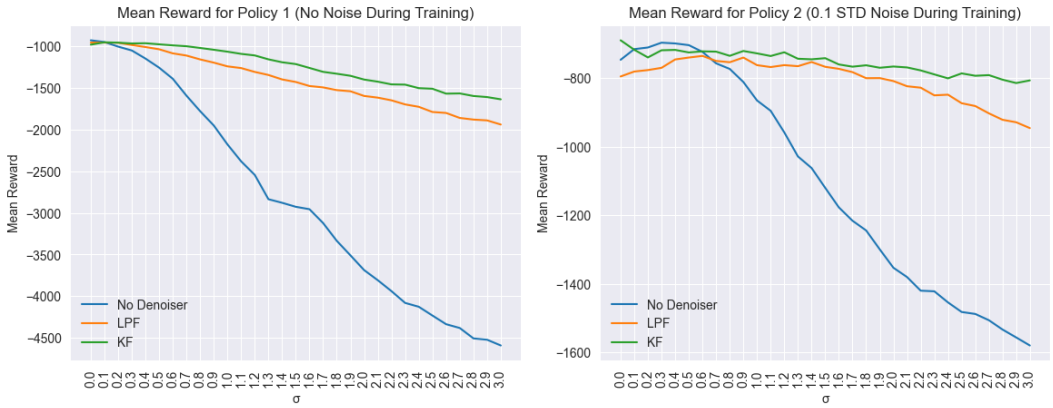
Policy 1 : As delineated in Figure 3.6(a), Policy 1 commences with a 60% success rate at the baseline. However, a precipitous decline in performance is observed as σ increases, plummeting to nearly 0% by

$\sigma = 2.0$. The incorporation of Low Pass and Kalman Filters mitigates this decline, sustaining a 30% success rate even at $\sigma = 3.0$, showcasing the efficacy of denoising in preserving policy performance under increased noise levels.

Policy 2 : Figure 3.6(b) reveals that Policy 2 not only surpasses Policy 1 at the baseline with a 72% success rate but also demonstrates an enhanced performance peak of 76% at $\sigma = 0.5$. Despite a subsequent performance decrement with escalating σ, Policy 2, even at $\sigma = 3.0$, maintains a 56% success rate, effectively doubling the performance of Policy 1 under similar conditions. The strategic application of a denoiser further bolsters Policy 2's resilience, maintaining a robust success rate of approximately 70% across varying levels of noise.

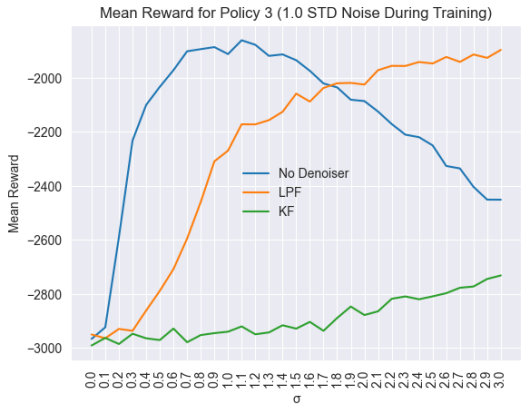
Policy 3 : As per Figure 3.6(c), Policy 3 exhibits a near-zero success rate at baseline, which starkly transitions to 50% at a σ of 0.3, before experiencing a gradual decline with increasing σ. Interestingly, Policy 3 outperforms Policy 1 in higher noise scenarios, aligning closely with Policy 1's denoised performance for σ values exceeding 1.8. The integration of an LPF with Policy 3 yields a modified response curve, indicating a delayed but steadier ascent to a 50% success rate and a more gradual performance reduction thereafter, maintaining a near-constant success rate past a σ of 0.1. Contrarily, the Kalman Filter does not exhibit a similar trend within the tested experimental range.

This intricate analysis underscores the nuanced effects of unbiased noise on policy performance, highlighting the critical role of denoisers in enhancing policy robustness against environmental noise perturbations. The findings from these evaluations not only contribute to the understanding of noise resilience in autonomous systems but also underscore the potential of denoising techniques in improving operational efficiency under diverse noise conditions.



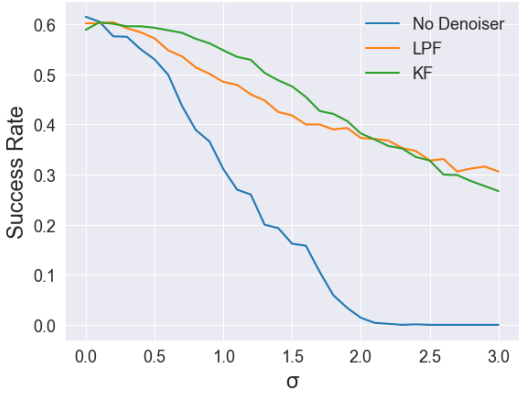
(a) Policy 1

(b) Policy 2

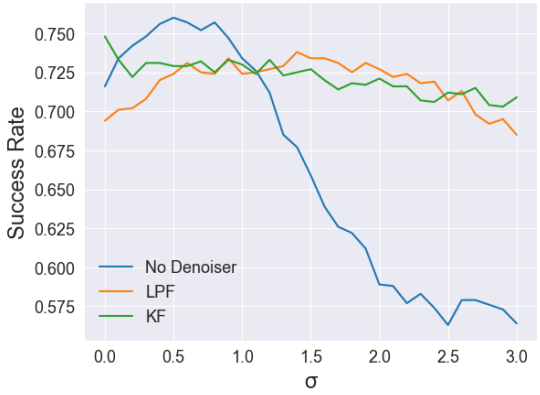


(c) Policy 3

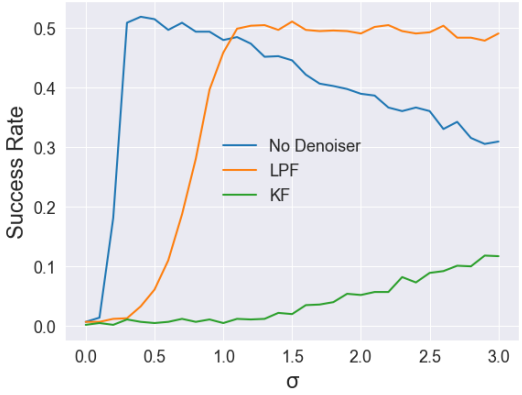
Figure 3.5: **Mean Reward against Unbiased Noise** ($\mu = 0, 0 \leq \sigma \leq 3.0$ in meters) The images depict line graphs showing the mean reward over different noise levels σ for three separate policies in UAV obstacle avoidance. The first image (a) shows Policy 1 with no noise during training, the second (b) shows Policy 2 with 0.1 standard deviation (STD) noise during training, and the third (c) shows Policy 3 with 1.0 STD noise during training. In each graph, there are lines representing the performance without a denoiser, with a Low Pass Filter (LPF), and with a Kalman Filter (KF).



(a) Policy 1



(b) Policy 2



(c) Policy 3

Figure 3.6: **Success Rate against Unbiased Noise** ($\mu = 0, 0 \leq \sigma \leq 3.0$ in meters): (a) Policy 1's success rate decreases sharply with an increase in σ, reaching a 0% success rate at around $\sigma = 2$. The decline is slowed down by the introduction of a denoiser. (b) Policy 2's success rate initially increases with a small increase in σ, then drops off quickly, still outperforming Policy 1 for all tested values of σ, even with a denoiser. The addition of a denoiser results in a nearly constant success rate of around 70% throughout the σ range. (c) Policy 3 starts with a near 0% success rate in the presence of very low noise but jumps to a 50% success rate at $\sigma = 0.3$, then decreases gradually. The Low Pass Filter (LPF) follows this trend with a lag, while the Kalman Filter (KF) is consistently poor.

3.2.2.2 Bias-only Noise Evaluation

Within this analytical segment, we scrutinize the performance dynamics of the three rigorously trained policies under conditions dominated by bias-only noise, with bias levels (μ) extending from 0 to 0.3, while maintaining zero variance ($\sigma = 0$), and incrementing μ by 0.01. This examination

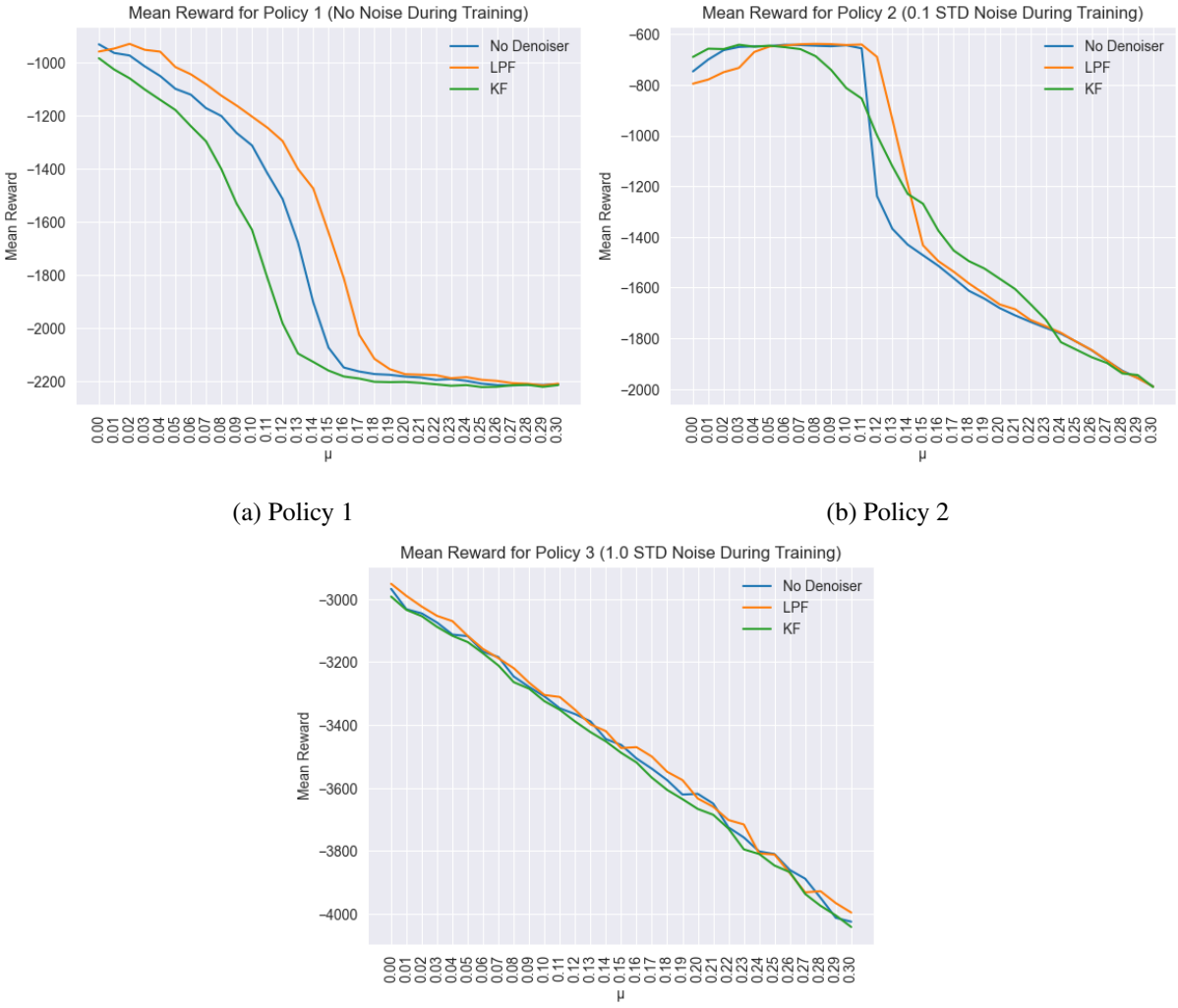
seeks to elucidate the impact of bias on policy efficacy, both in the presence and absence of denoising mechanisms.

Empirical evidence, as depicted in Figure 3.8, categorically illustrates the inherent limitations of denoisers in counteracting the deleterious influence of bias-only noise. This limitation manifests uniformly across all policies, underscoring a pivotal challenge in noise compensation strategies. **Policy 1** : As illustrated in Figure 3.8(a), there is a discernible decrement in success rate corresponding with incremental bias for Policy 1. This trend unambiguously indicates the susceptibility of Policy 1 to bias, revealing a correlation between increasing bias and diminishing performance.

Policy 2 : Contrary to Policy 1, Policy 2 exhibits a remarkable resilience to bias-induced performance degradation, as shown in Figure 3.8(b). This policy maintains robust performance metrics up to a μ of 1.2, post which a precipitous decline is observed. Such resilience underscores the intrinsic adaptability of Policy 2 to bias, up to a critical threshold.

Policy 3 : The performance of Policy 3, detailed in Figure 3.8(c), presents a challenging scenario for systematic analysis due to its significantly lower success rates. The variability observed here is predominantly attributed to random factors, rather than systematic policy responses to bias adjustments.

This comprehensive analysis accentuates the criticality of understanding the nuanced effects of bias-only noise on autonomous policy performance. It highlights the imperative for advanced denoising strategies or bias compensation mechanisms that extend beyond the conventional capabilities of current denoisers. Furthermore, it brings to light the variable resilience of different policies to bias, emphasizing the need for tailored approaches in policy design and training to mitigate the impacts of bias in noise-dominant environments.

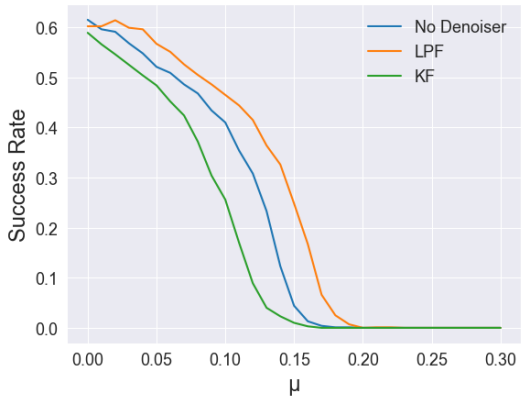


(a) Policy 1

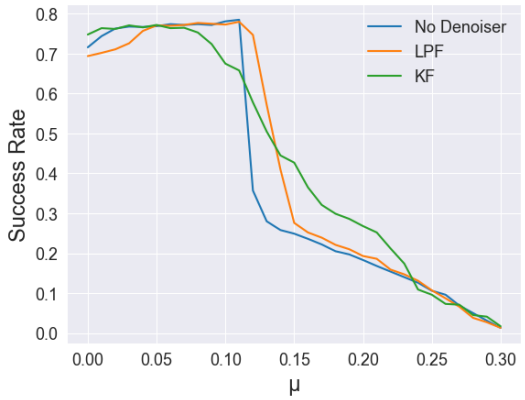
(b) Policy 2

(c) Policy 3

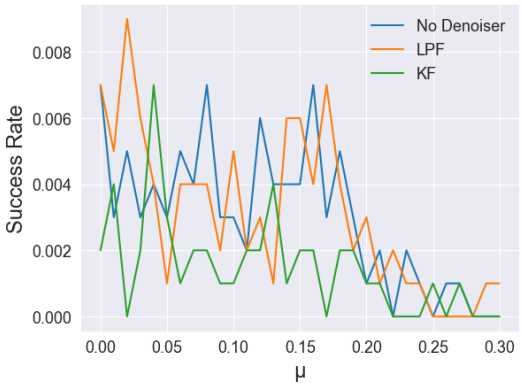
Figure 3.7: **Mean Reward against Biased-Only Noise** ($0 \leq \mu \leq 0.3, \sigma = 0$) The images are line graphs representing the mean reward for three different policies when tested with varying levels of bias (denoted as μ). Each graph depicts the results for a policy under a specific training condition with different levels of bias from 0 to 0.30 The first image (a) is for Policy 1, which was trained with no noise. The second image (b) is for Policy 2, which was trained with 0.1 STD noise. The third image (c) is for Policy 3, which was trained with 1.0 STD noise.



(a) Policy 1



(b) Policy 2



(c) Policy 3

Figure 3.8: **Success rate against bias-only noise** ($0 \leq \mu \leq 0.3, \sigma = 0$): (a) Policy 1's success rate drops off quickly with an increase in μ, hitting 0% around $\mu = 0.16$. (b) Policy 2 manages to maintain a consistent success rate up to $\mu = 0.11$, before a sudden drop-off, (c) Policy 3's success rate is near zero for all values of μ, with any fluctuation in the success rate being attributed to randomness. We see in all three cases that the denoisers can provide no assistance to the policies when faced with bias-only noise.

3.2.2.3 Biased Noise Evaluation

In this rigorous examination, we present the outcomes of evaluating the three distinct policies in environments subjected to a spectrum of biased noise, characterized by a bias (μ) ranging from 0 to 0.3, and a variance (σ) from 0 to 3.0, incrementing μ by 0.01 and σ by 0.1. This assessment specifically eschews the use of denoisers to isolate and understand the intrinsic resilience of each policy to biased noise.

Policy 1 : Analysis of Figure 3.10(a) delineates three pivotal regions. Initially, a minor blue region in the lower left corner underscores the policy's inefficacy under conditions of high bias but low variance.

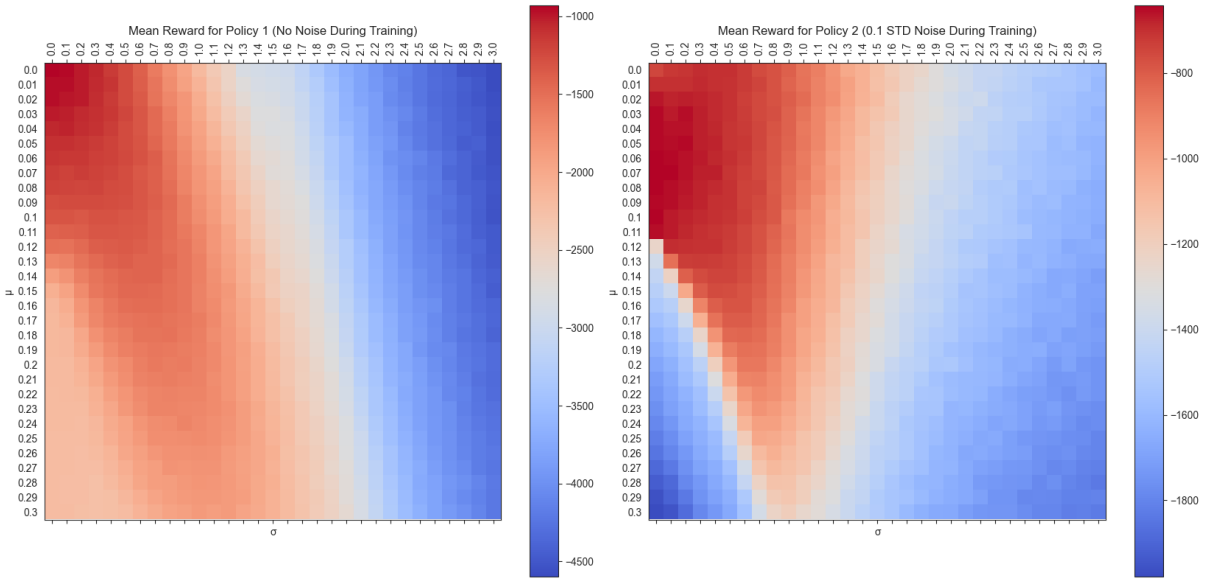
Subsequently, a broader blue region to the right signifies the policy's collapse under high variance noise. Intriguingly, a red region descending diagonally suggests a mitigative strategy against high bias through the strategic introduction of unbiased noise at selected σ levels. For instance, at $\mu = 2.0$, a zero success rate at $\sigma = 0$ can be elevated to over 40% success by choosing an optimal σ of 1.2. This effect tapers with increasing μ, highlighting the diminishing returns of this approach against escalating bias.

Policy 2 : Figure 3.10(b) showcases Policy 2's robustness across a wide bandwidth of μ and σ values. The convergence of the blue region at the bottom left with the drop-off in performance at $\mu = 1.2$ from Figure 3.8(b) is notable. Yet, the surrounding red contour indicates the potential to counteract bias effects by fine-tuning σ, enhancing the success rate dramatically from 35% to beyond 70%. As bias and variance intensify, the success rate predictably declines, as depicted by the gradient towards the diagram's lower and right peripheries.

Policy 3 : Reflecting trends observed in Figure 3.6(c), Policy 3's performance is markedly poor for minimal σ values, as illustrated by the left-sided blue strip in Figure 3.10(c). Beyond a σ threshold of 0.3, a significant uptick in success rates is observed for lower μ values, with performance deteriorating radially outward. This decline is more pronounced with increases in μ compared to σ, aligning with Policy 3's historical consistency at high variance levels but marked underperformance across all bias levels.

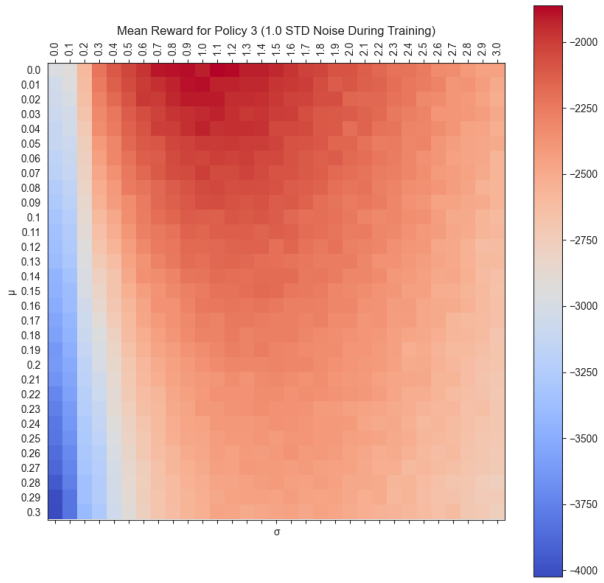
This nuanced analysis underscores the complex interplay between bias and variance in shaping policy performance. It highlights the potential of leveraging unbiased noise as a strategic tool to ameliorate the adverse effects of bias, albeit with diminishing efficacy against higher bias levels. These insights pave the way for future explorations into noise-resilient policy development and the strategic use of noise in enhancing autonomous system performance in noise-pervasive environments.

[0]The code for training and evaluation of the policies, with detailed results, can be found at https://github.com/BhaskarJoshi-01/DroneControl



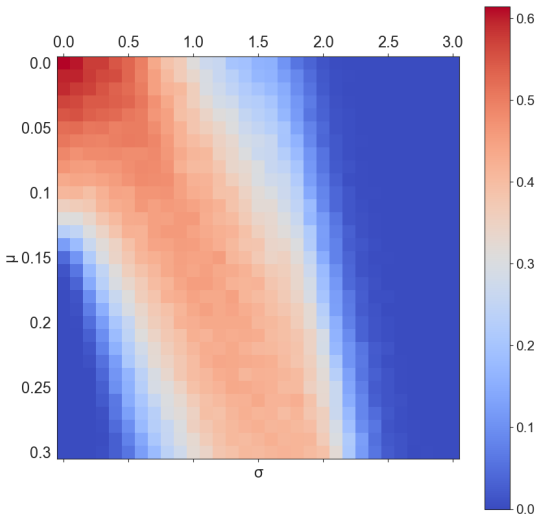
(a) Policy 1

(b) Policy 2

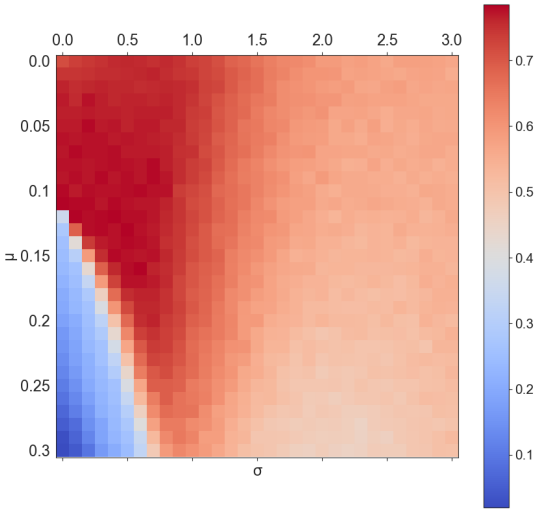


(c) Policy 3

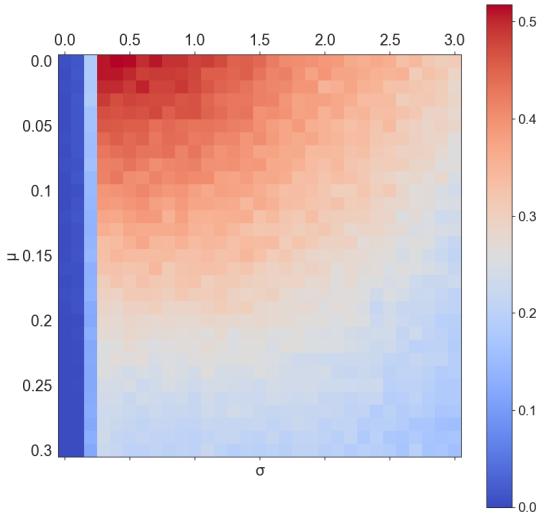
Figure 3.9: **Mean Reward against Biased Noise** ($0 \leq \mu \leq 0.3, 0 \leq \sigma \leq 3.0$) The heatmaps provided illustrate the performance of three different policies for UAV obstacle avoidance with respect to biased noise during training. Each heatmap shows the mean reward on the vertical axis (representing the bias μ) against the standard deviation of noise σ on the horizontal axis. Darker red indicates higher rewards, while darker blue signifies lower rewards.



(a) Policy 1



(b) Policy 2



(c) Policy 3

Figure 3.10: **Success rate against Biased Noise** ($0 \leq \mu \leq 0.3, 0 \leq \sigma \leq 3.0$): The colour scheme of the heatmap maps red to high success rate and blue to low success rate. Each heatmap's spectrum scale is presented as a colorbar to the right of the plot. (a) Policy 1 shows a somewhat linear trend — the negative effects of bias-only noise can be mitigated to great effect by carefully choosing a value for σ to inject, but performance becomes worse if either or both of them increase. (b) Policy 2, already being robust to variance, improves upon the success rate of Policy 1 in regions with high values of σ. Interestingly, the success rate at $\mu = 0.12$ jumps from 35% to over 70% when increasing σ from 0 to 0.1. (c) Policy 3 performs poorly in the presence of low variance, indicated by the blue strip on the left side. Success rate drop-off is steeper along the μ axis than along the σ axis, indicating higher robustness to variance than bias, but showing no benefit of injecting unbiased noise.

3.3 Time Varying Bias

The study for time varying Bias is done on Static Obstacles (SO) i.e Environment 1.

Environment 1 The first environment encompasses N static obstacles $O \in$ SO, with the quantity N being variably selected for each episode from a set $\{0, 1, 2, 3\}$. The distribution of these obstacles is methodically arranged: uniform along the x-axis, Gaussian along the y-axis, and positioned at a consistent elevation, the median, along the z-axis.

3.3.0.1 Effect of Noise : Environment 1

For environment 1, we trained three policies with noise having linearly dependent time varying bias and different fixed levels on variance. Such a type of noise is generally produced by IMU readings. Then we compared their training results, such as mean reward and mean episode duration over 100 episodes. We then tested the three learnt policies in the same environment for noise with the same time varying bias and fixed levels of variance.

3.3.0.2 Training: Environment 1

we trained three policies each having noise with the same time varying bias as (20)

$$\mu = \pm 0.3 \cdot \left(\frac{\text{timesteps}}{L_{\text{max}}} \right) \tag{3.11}$$

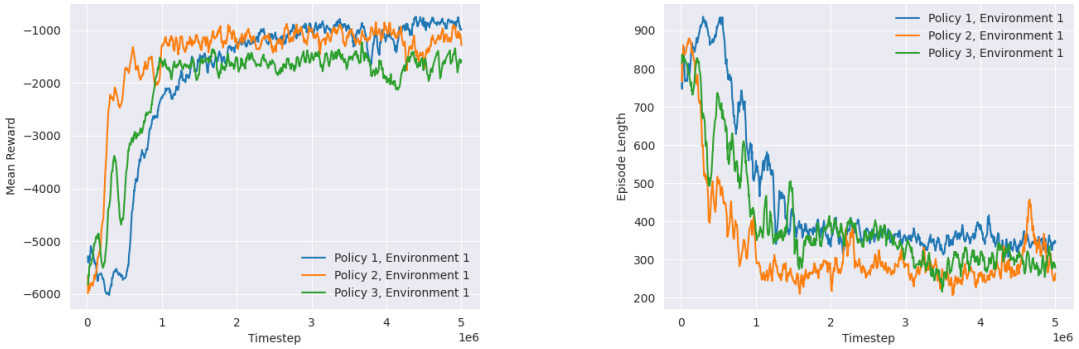
Where L_{max} is the fixed maximum possible length for an episode. The positive or negative deviation is randomly chosen at start of every episode. For our study, each step is 1/48 seconds long, thus effectively modelling the bias as

$$\mu(k) = \pm 0.3 \cdot kT, \tag{3.12}$$

where $k = 0, 1, 2, \ldots$ and $T = \frac{1}{48}$ seconds.
Furthermore, each policy has three different fixed level of variance, giving us 3 policies as,

1. **Policy 1**: No variance (time varing μ, $\sigma = 0$)
2. **Policy 2**: Low Variance (time varying μ, $\sigma = 0.1$)
3. **Policy 3**: High High Variance (time varying μ, $\sigma = 1.0$)

To allow better robustness against obstacles, at least one fixed obstacle was ensured to be present during training. The maximum number of obstacles as described in the methodology section, are three.



(a) (b)

Figure 3.11: **Policy Training Results with noise having linearly dependent time varying bias**: (a) indicate the mean reward progression for the policies trained in environment 1, while (b) indicate episode length progression through the training process for policies trained in environment 1. The mean reward increases in all cases indicating improved obstacle avoidance behaviour, and episode length decreases indicating the agent is learning how to avoid obstacles and reach the goal well before the episode time limit.

3.3.0.3 Evaluation of Trained Policy: Time Varying Biased Noise

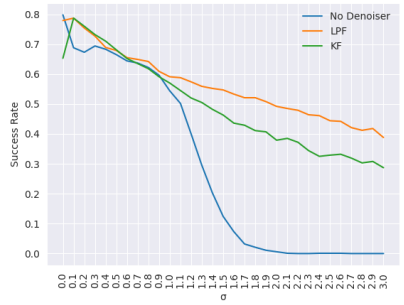
Policies 1,2 and 3 trained in environment 1 containing noise with time varying bias and fixed variance of 0, 0.1 and 1.0 were tested in the same environment with the same time varying bias but on a wider range of variances ($0 \leq \sigma \leq 3.0, \Delta\sigma = 0.1$). We also evaluate the denoising capacity of LPF and Kalman filter during testing. The evaluation plots can be found in figure 3.12 and can be described as follows:

Policy 1: Policy 1 has a success rate of 0.8 for the no denioser and LPF case, and around 0.65 for the KF. The performance for the no denoiser case dropes and reaches zero for $\sigma \geq 2.1$. Both denoisers assist in reducing the drop in performances giving a success of 0.3 for KF and 0.4 for LPF at $\sigma = 3.0$.

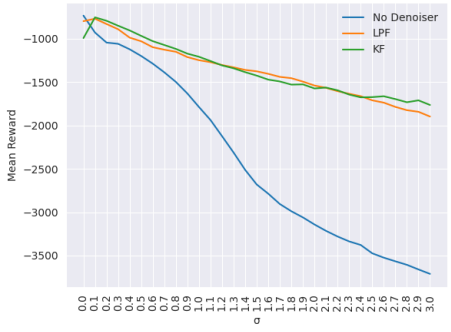
Policy 2: Policy 2 behaves very similar to policy 1 but starts off with a lower success rate of 0.6 at $\sigma = 0$ and decreases till near zero at $\sigma = 3.0$ for no denoisers. Presence of denoisers again decreases the fall in performance giving a success rate of about 0.45 at $\sigma = 3.0$ for KF and 0.35 for LPF.

Policy 3: The policy starts off with a success rate between 0.35 and 0.4 for both cases, with and without denoisers. The denoisers maintain the success rate for all values of σ till $\sigma = 3.0$ but the for the no denoiser case, the performance quickly starts to plumet after $\sigma = 1.3$, falling down to almost 0 at $\sigma = 3.0$

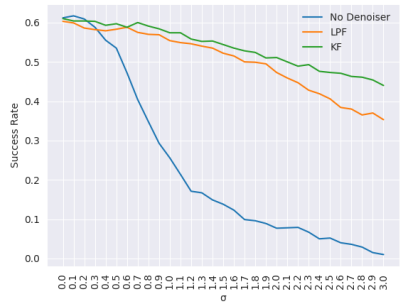
We can effectively conclude from the plots that adding variance over time varying bias during training deprecates the performance of the policy against noise with low fixed variance and time varying bias.



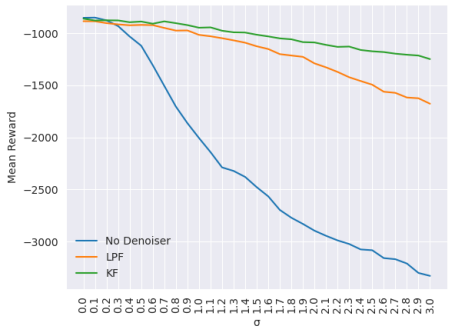
(a) Policy 1, Environment 1



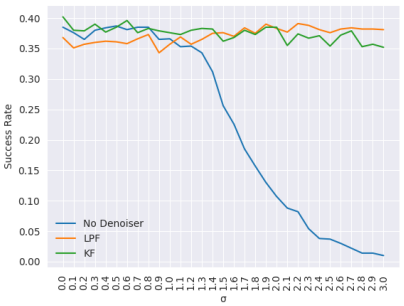
(b) Policy 1, Environment 1



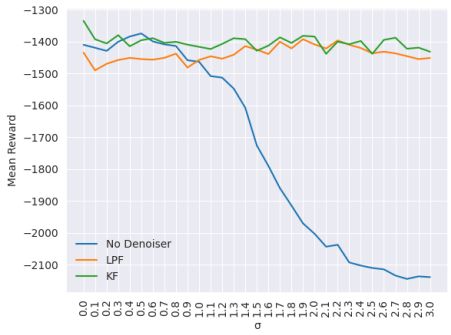
(c) Policy 2, Environment 1



(d) Policy 2, Environment 1



(e) Policy 3, Environment 1



(f) Policy 3, Environment 1

Figure 3.12: **Success Rate and Mean Reward Analysis with Time-Varying Bias ($\mu = kT, 0 \leq \sigma \leq 3.0$) in Meters:** (a, b) Policy 1 demonstrates a decreasing success rate with an increase in noise standard deviation σ, nearing a 0% success rate at approximately $\sigma = 2$. The denoisers (LPF and IF) attenuate the rate of decline. The mean reward also shows a steep decline with increasing σ, mitigated by denoisers. (c, d) Policy 2 shows a similar trend in success rate, beginning at a lower success rate of 0.6 and experiencing a decline as σ increases. Here too, denoisers help in reducing the fall in success rates and mean rewards. (e, f) Policy 3 maintains a more constant success rate between 0.35 and 0.4, but begins to plummet post $\sigma = 1.3$. Denoisers are effective in preserving the initial success rate and mean reward level across the noise range.

However for noise with high variance, adding variance during training improves the performance of the policies. In both cases, denoisers assist in improving the performance.

We have firstly trained a policy under time varying bias and three levels of fixed variance in a static environment, then tested on a wider range of variances along with the time varying bias. We observed that for low fixed variance and varying bias, the policy trained with no variance performed the best and denoisers did not have much effect (fig 4.5 (a)), while in the case of high variance, it is better to train with a small variance and use denoisers during deployment. Such results can be greatly utilised while making training choices for RL based navigation while relying on IMUs as sensors, if the nature of the varying bias of IMU is known.

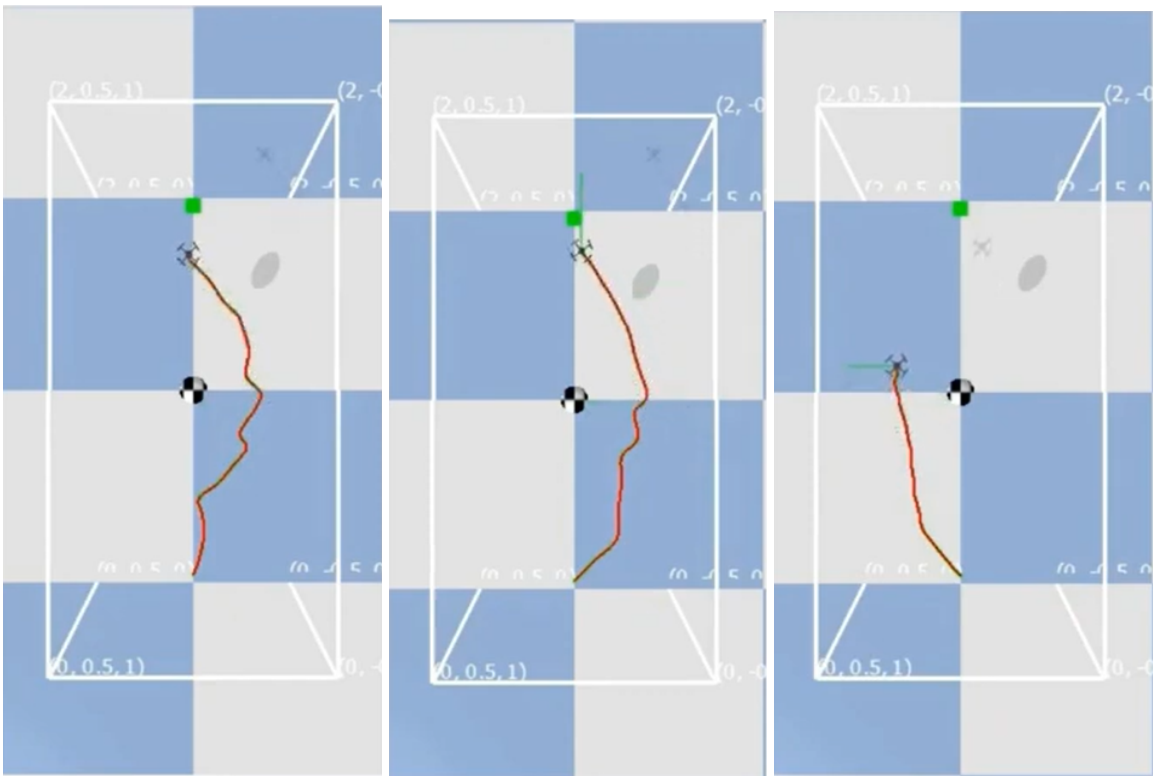
3.4 Simulation Trajectories

3.4.1 Effects of Noise on Training : No noise, No Denoiser

Figure 3.13 illustrates the simulation trajectories within an environment devoid of both noise and denoising mechanisms. The depicted trajectories are highlighted by a red line for each scenario. In scenarios (a) and (b) of Figure 3.13, the trajectory successfully reaches the designated target. However, in scenario (c) of Figure 3.13, the trajectory concludes upon reaching the maximum number of steps, leading to the termination of the episode. Additionally, a green line emanates from the drone (in some figures), indicate the maintained safe distance from the nearest obstacle throughout its journey.

3.4.2 Effects of Noise on Training : $\mu = 0$, $\sigma = 1.5$ and No Denoiser

This section evaluates the performance of various policies in an environment characterized by unbiased noise with $\mu = 0$, $\sigma = 1.5$, and the absence of a denoising mechanism. Observations from Figure 3.14 reveal that policies represented in scenarios (b) and (c) are trained with a non-zero σ, enabling them to adapt and perform effectively in the presence of unbiased noise. In contrast, the policy depicted in Figure 3.14 (a), which was not trained under noisy conditions, exhibits poor performance when exposed to high levels of unbiased noise.

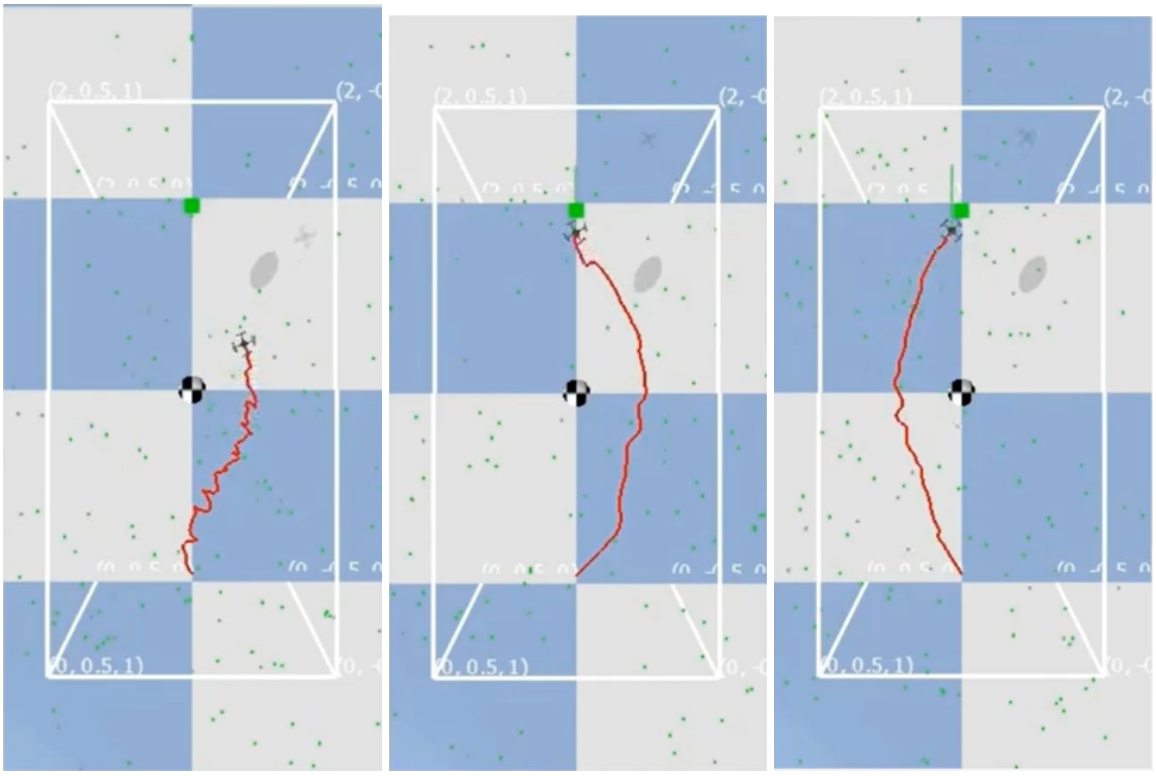


(a) Policy 1 $\sigma = 0$

(b) Policy 2 $\sigma = 0.1$

(c) Policy 3 $\sigma = 1$

Figure 3.13: **Simulation Trajectories : No noise, No Denoiser**

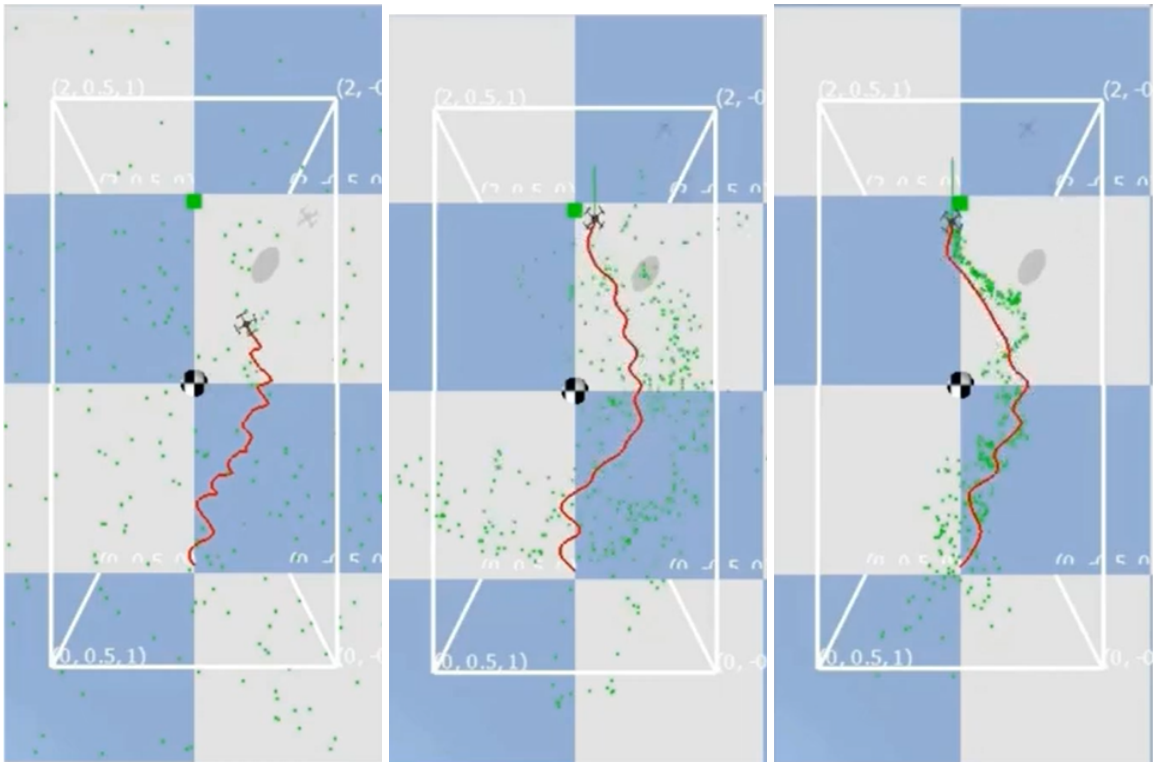


(a) Policy 1 $\sigma = 0$ (b) Policy 2 $\sigma = 0.1$ (c) Policy 3 $\sigma = 1$

Figure 3.14: **Simulation Trajectories for Effects of Unbiased Noise on Training Without Denoising**

3.4.3 Unbiased Noise and Filters

This subsection presents an evaluation of Policy 1, under conditions of unbiased noise ($\mu = 0$, $\sigma = 1$), with the incorporation of filtering techniques. Figure 3.15 illustrates the trajectory outcomes when Policy 1 is subjected to these conditions. Notably, scenarios (b) and (c) in Figure 3.15 have Low Pass Filter (LPF) and Kalman Filter (KF), respectively. These filters enable successful target achievement in an environment influenced by unbiased noise with parameters $\mu = 0$ and $\sigma = 1$. The efficacy of these filtering methods is evident, highlighting their utility in enhancing navigational accuracy under noisy conditions.



(a) **Policy 1**
No Denoiser

(b) **Policy 1**
LPF

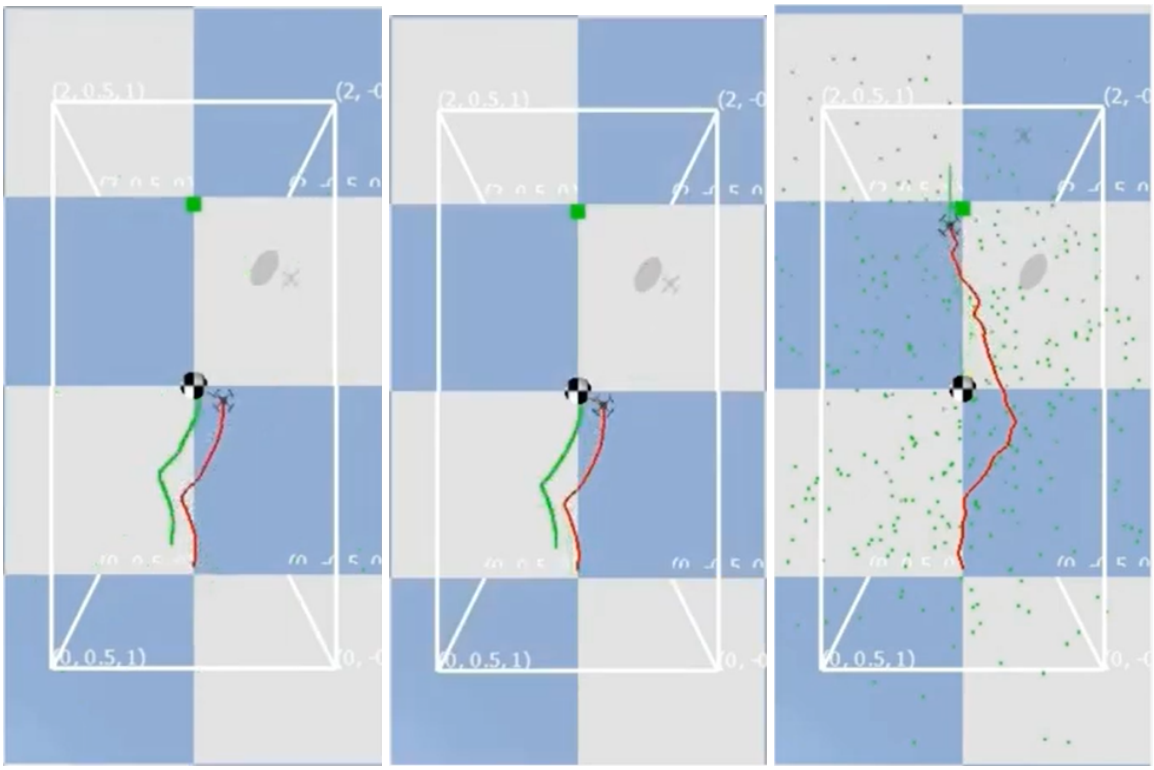
(c) **Policy 1**
KF

Figure 3.15: **Simulation Trajectories for Unbiased Noise and Filters**

3.4.4 Handling Biased Noise $\mu = 0.1$ and $\sigma = 0$

This subsection explores the effect of biased noise ($\mu = 0.1$ and $\sigma = 0$) on navigational strategies and evaluates the effectiveness of denoising approaches. In Figure 3.16, scenarios (a) and (b) demonstrate trajectories under biased noise conditions without filter and with the application of a Kalman Filter (KF), respectively. The trajectory in Figure 3.16(b), utilizing KF, appears smoother compared to Figure 3.16(a), which lacks a denoising mechanism. However, both scenarios exhibit an inability to circumvent obstacles effectively.

Introduction of unbiased noise ($\sigma = 0.5$), as depicted in Figure 3.16(c), significantly enhances obstacle avoidance, suggesting that the adverse effects of bias can be mitigated by introducing a specific magnitude of noise variance. This finding underscores the potential for strategic noise manipulation to improve navigational outcomes under biased conditions.



(a) **Policy 1**
No Denoiser

(b) **Policy 1**
KF

(c) **Policy 1**
Unbiased Noise Injection
$\sigma = 0.5$

Figure 3.16: **Simulation Trajectories for Handling Biased Noise**

Chapter 4

Navigating UAVs Through Dynamic Environments

In this chapter, we delve into the enhancements and adaptations made to UAV navigation systems for effectively dealing with dynamic obstacles environment under measurement uncertainty.

4.1 System Architecture Overview

The architecture of the methodology followed is shown in Fig. 4.1.

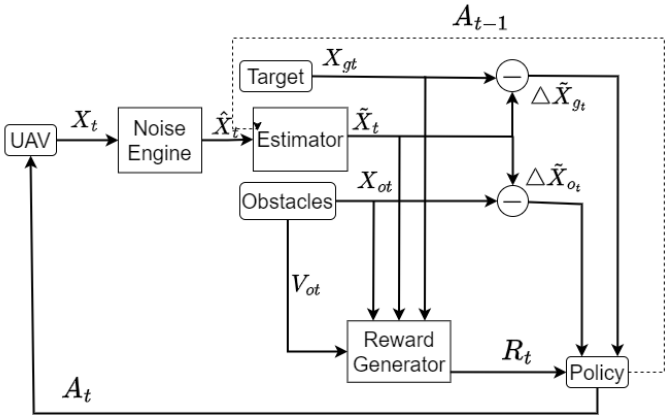


Figure 4.1: **System Architecture**: At time step t, We have access to the UAV's position $\mathbf{X_t}$, the nearest obstacle's position $\mathbf{X_{ot}}$, and the target position $\mathbf{X_{g_t}}$. We introduce noise to $\mathbf{X_t}$ to create a noisy position estimate, denoted as $\hat{\mathbf{X}}_\mathbf{t}$, which is then denoised to obtain the final position estimate $\tilde{\mathbf{X}}_\mathbf{t}$, taking into account the previous action A_{t-1}. The denoised position estimate, along with $\mathbf{X_{ot}}$ and $\mathbf{X_{g_t}}$, are used to generate the environmental observation $\tilde{O}_t$. The reward R_t is computed based on the true position $\mathbf{X_t}$ for Environment 1 and both the position $\mathbf{X_t}$ and velocity $\mathbf{V_t}$ for Environment 2. The policy takes the observation and reward as inputs and produces the action to be taken, denoted as A_t.

4.1.1 The Environment

Note: Evironmnent will dynamic obstacles will be stated as Environment 2 and with static obstacles will be referred as Enviornment 1.
For Dynamic Obstacle (DO) environment also the spawn position and dimentions are same as Chapter 3, at the beginning of each episode, the UAV is placed randomly at:

$$\mathbf{X}_0 = \left[x_{min} + r_{minor}, y_{g_0}, \frac{z_{min} + z_{max}}{2} \right] \tag{4.1}$$

$$y_{g_0} \sim Uniform(y_{min} + r_{minor}, y_{max} - r_{minor}) \tag{4.2}$$

and, the goal position is set as:

$$\mathbf{X}_g = \left[x_{max} - r_{minor}, \frac{y_{min} + y_{max}}{2}, \frac{z_{min} + z_{max}}{2} \right] \tag{4.3}$$

Here, m_{min} and m_{max} denote the environment's dimensions along the m-axis (where m could be x, y, or z), and r_{minor} represents the safety margin to avoid boundary exceedance.

- **Environment 2** This consists of two dynamic obstacles (DO), one exhibiting simple harmonic motion along the y-axis as per:

  $$\frac{d^2y}{dt^2} = -\omega^2 \dot{y} \tag{4.4}$$

  Wechoose amplitude as eq. 4.5

  $$\frac{y_{min} + y_{max}}{2} \tag{4.5}$$

  such that it is always between the bounds of the Geofence. Value of ω is taken as 0.72 rad/s. The other moving with a constant linear velocity v_i of 0.005 m/s along the x-axis as follows:

  $$\mathbf{v} = v_i \hat{i} \tag{4.6}$$

  The first obstacle is spawned randomly on the x axis, while for the second obstacle, the initial y coordinate is distributed based on the normal distribution and the initial x coordinate is fixed as x_{max}.

  The action space, observation space and the reward functions for Dynamic obstacle environment are as follows:

4.1.1.1 Action Space

Both environments (Static and Dynamic) share the same action space, where a valid action can be any vector with the following format ref eq 4.7

$$A = \begin{bmatrix} v_x & v_y & v_{mag} \end{bmatrix} \tag{4.7}$$

v_x and v_y represent the velocities of the Unmanned Aerial Vehicle (UAV) in the x and y axes, correspondingly. The term v_{mag} denotes the overall magnitude of the velocity vector. The velocity command can be calculated based on the action as follows ref eq 4.8:

$$\vec{v} = \frac{\begin{bmatrix} v_x & v_y \end{bmatrix}}{\sqrt{v_x^2 + v_y^2}} \times v_{mag} \tag{4.8}$$

Wehave the following observation space and reward function for Environment 2 (i.e with Dynamic obstacles) .

4.1.1.2 Observation Space

Observation space represents a condensed, continuous representation of the current state similar for both the environments (Static and Dynamic) as described in Chapter 3 3.1.1.1 . Observation space selectively encodes essential information for the agent to acquire an effective policy as Eq. 4.9

$$O_t = \begin{bmatrix} \Delta x_{g_t} & \Delta y_{g_t} & \Delta x_{ot} & \Delta y_{ot} \end{bmatrix} \tag{4.9}$$

Here, Δx_{g_t} and Δy_{g_t} denote the distances to the goal along the x and y axes at time t, respectively. Similarly, Δx_{ot} and Δy_{ot} represent the distances to the closest obstacle or wall along the x and y axes, respectively. Because of the noise in localisation of the UAV and noise in localisation of the obstacles as well as goal positions, the relative positions would also contain noise, giving us the the noisy observation space $\hat{O}_t$ as Eq. 4.10

$$\hat{O}_t = \begin{bmatrix} \Delta \hat{x_{g_t}} & \Delta \hat{y_{g_t}} & \Delta \hat{x_{ot}} & \Delta \hat{y_{ot}} \end{bmatrix} \tag{4.10}$$

And during evaluation Wetest the effectiveness of various denoising methods, giving us the denoisesd estimates of the observation space $\tilde{O}_t$ as Eq. 4.11

$$\tilde{O}_t = \begin{bmatrix} \Delta \tilde{x_{g_t}} & \Delta \tilde{y_{g_t}} & \Delta \tilde{x_{ot}} & \Delta \tilde{y_{ot}} \end{bmatrix} \tag{4.11}$$

4.1.1.3 Reward Function

For the reward function for environment 2, We have considered a velocity obstacle reward $R_O(t)$ for both moving obstacles as follows:

$$R_O(t) = \begin{cases} R_{collide} & \text{if } V \in VO_O(t) \\ 0 & \text{otherwise} \end{cases} \tag{4.12}$$

Where V is the velocity of the UAV and $VO_O(t)$ is the velocity obstacle [59] with time horizon T_H of obstacle O with respect to the UAV at time step t. This reward incentivizes the agent to avoid velocities that may result in future collisions.

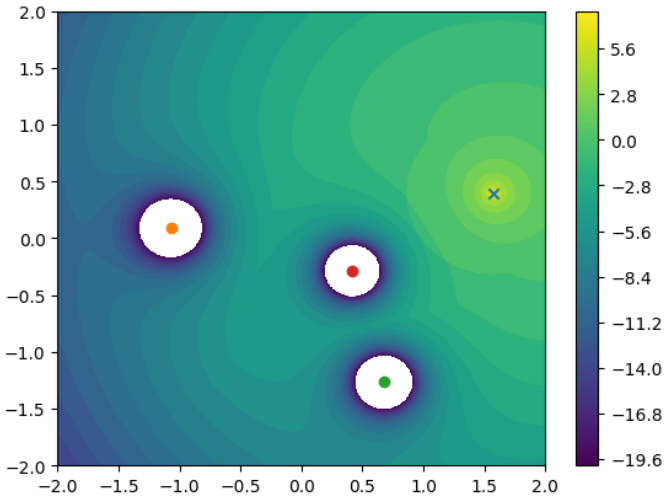
The reward, for *Environment 2* is then given by adding R_O for both moving obstacles to R_t defined in Eq. 4.13.

$$R'_t = R_t + \sum_{\forall O \epsilon DO} R_O(t) \tag{4.13}$$

Where, R_t is the same as eq 3.9 and is given here by 4.14 :

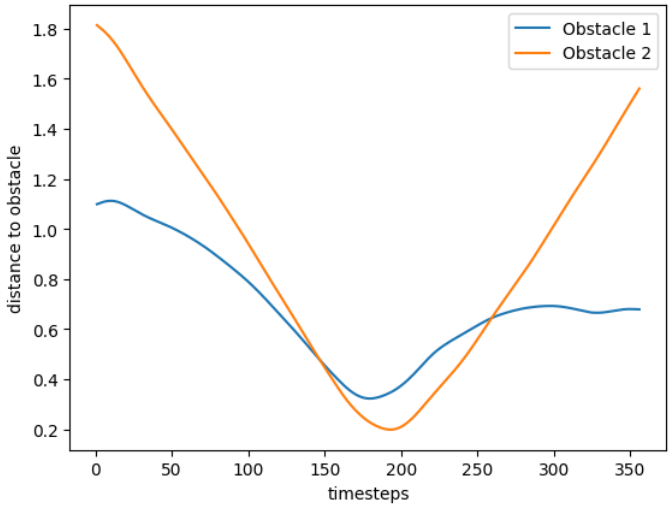
$$R_t = \begin{cases} R_s, & \text{if distance to target} < \epsilon_{\text{success}} \\ R_f, & \text{if collision with an obstacle occurs} \\ R_f, & \text{if the agent is out of time} \\ (-R_d \left\| \begin{bmatrix} \Delta x_t & \Delta y_t \end{bmatrix} \right\| - R_{major} 1_{major} - R_{minor} 1_{minor}) & \text{otherwise} \end{cases} \tag{4.14}$$

$R_s > 0$ represents the reward for success, $R_f < 0$ is the penalty for failure, $R_d > 0$ is the coefficient for the distance penalty, and $r_{minor} > 0$ and $r_{major} > 0$ are penalties for breaching minor and major bounds, respectively. t is here is total number of time steps and L_{max} is a chosen maximum time after an episode terminates by default.The variables 1_{major} and 1_{minor} are binary indicators set to 1 when the major and minor safety bounds with radii R_{major} and R_{minor}, respectively, are violated. They are set to 0 when no violation occurs. Additionally, a slight negative reward, determined by the UAV's distance from the target, is assigned to each non-terminal time step. This is done to incentivize the agent to reach the target in as few steps as possible.

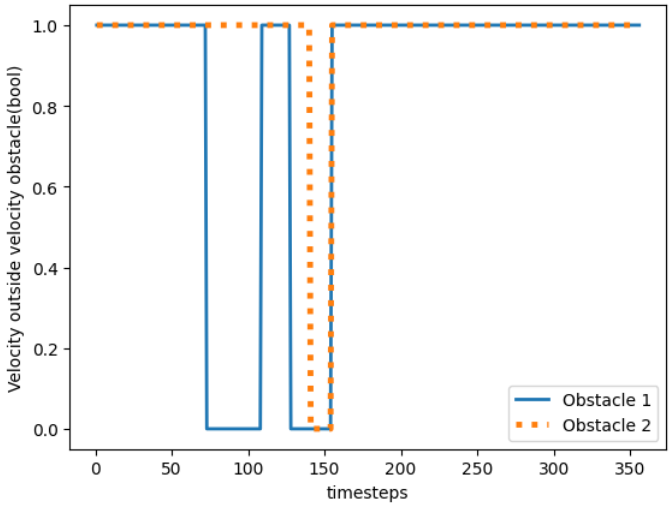


(a)

Figure 4.2: **Visualization of the Reward Function:** This figure displays a heatmap visualization of the reward function utilized within the context of drone navigation. Lower rewards are indicated by darker shades, whereas higher rewards are denoted by bright yellow areas. Obstacles within the environment are marked with "o", and the designated target location is indicated by an "x". The gradient of colors from dark to bright illustrates the increasing reward values, guiding the drone's movement from regions of lower reward to higher reward. This visualization aids in understanding how the drone is incentivized to navigate around obstacles towards its target.



(a) Distance to Obstacle VS Timestep for UAV



(b) Velocity outside Velocity obstacle (bool) VS Timestep for UAV

Figure 4.3: **Visualization of Velocity Obstacle for UAV Obstacle Avoidance**

From the figure 4.3 Initially the UAV is focussed on reaching the goal and chooses velocities in that direction. As the obstacles get closer, the velocities intersect with the velocity obstacles of the obstacles.The UAV, as observed in the plots shifts its focus to obstacle avoidance, choosing new velocities outside the velocity obstacles. This maneuver guides the UAV away from the obstacles. As seen from the plots the distance to obstacles starts increasing soon after the UAV selects velocities outside the velocity obstacle.

4.1.2 The Agent

We have chosen to use an unmodified implementation of vanilla PPO (already described in chapter 3, Section 3.1.2) for our experiments as our focus is on examining the influence of noise on a general policy that achieves an acceptable success rate.

4.1.3 The Effects of Noise

our primary emphasis in this research is on analyzing how noise in the observation space impacts the policy's performance. We assume that all elements in the observation spaces in both environments are subject to noise from the same normal distribution $\mathcal{N}(\mu, \sigma)$.

4.1.3.1 Environment 2: Dynamic Obstacles

For this environment, We train three policies on unbiased noise with different levels of variance and compare the training results. Subsequently, We conducted baseline tests for each trained policy by assessing their performance in a noise-free environment ($\mu = 0, \sigma = 0$). After establishing these baselines, We proceeded to evaluate these policies in environments characterized by different types of measurement noise, as outlined below:

1. Unbiased noise ($\mu = 0, \sigma \neq 0$)
2. Bias-only noise ($\mu \neq 0, \sigma = 0$)
3. Biased noise ($\mu \neq 0, \sigma \neq 0$).

4.2 Results

This section covers the experimental results obtained during training and during evaluation. Polices in environment 1 were trained for 5 million steps each on an RTX 3050 GPU, while policies in environment 2 were trained for 5 million steps each on an RTX 2080 Ti GPU. The process of evaluating a policy for 1000 episodes, with each episode having a maximum of 1000 timesteps, on a particular combination of μ, σ, and denoiser, requires approximately 10 minutes. You can find the detailed environment variables used in our experiments in TABLE 4.1. All distance measurements in TABLE 4.1 are in meters.

4.2.1 Training

4.2.1.1 Environment 2

We have trained the following 3 policies with different degrees of unbiased noise:

Hyperparameter	Value	Hyperparameter	Value
R_s	1000	R_{minor}	1
R_f	-1000	r_{major}	0.1
R_d	4	r_{minor}	0.2
R_{major}	5	$\epsilon_{success}$	0.1
$R_{collide}$	-0.5	T_h	5
L_{max}	1000	x_{min}	0
x_{max}	2	y_{min}	-0.5
y_{max}	0.5	z_{min}	0
z_{max}	1		

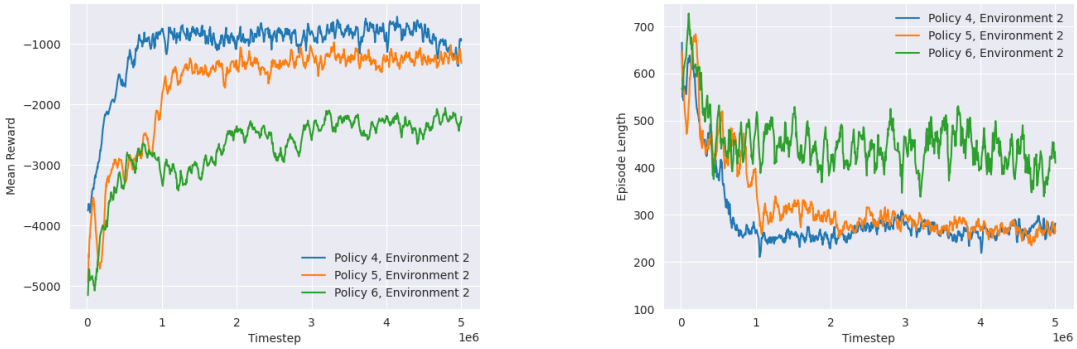
Table 4.1: **Environment Variables: Chapter 4**

1. **Policy 4**: No Noise ($\mu = 0$ and $\sigma = 0$)
2. **Policy 5**: Low Noise ($\mu = 0$ and $\sigma = 0.1$)
3. **Policy 6**: High Noise ($\mu = 0$ and $\sigma = 1.0$)

Policies 1, 2 and 3 were trained in Environment 1 while policies 4, 5 and 6 were trained in Environment 2. All policies were trained for 5 million steps. Policy hyperparameters are mentioned in TABLE 4.2.

Hyperparemeter	Value
Actor	$64 * 64 * 3$
Critic	$64 * 64 * 1$
Output Activation Function	tanh
Optimizer	Adam

Table 4.2: **Policy Training Hyperparameters: Chapter 4**



(a) (b)

Figure 4.4: **Policy Training Results for Environment 2**: (a) indicate the mean reward progression for the policies trained in environment 2, while (b) indicate episode length progression through the training process for policies trained in environment 2 .The mean reward increases in all cases indicating improved obstacle avoidance behaviour, and episode length decreases indicating the agent is learning how to avoid obstacles and reach the goal well before the episode time limit.

We have employed two criteria to assess the effectiveness of our training process for the policies: the mean reward obtained over the last 100 episodes and the mean duration of episodes over the last 100 episodes. Episode duration is defined as number of time steps before terminal state is reached.

- **Mean Reward:** From Figure 4.4(a), it's evident that the mean reward for all policies demonstrates a gradual increase throughout the training process. This trend suggests that the policies are effectively learning the desired behavior of avoiding obstacles and reaching the target in the most efficient manner possible.
- **Episode Duration:** Figure 4.4(b) the mean episode duration exhibits a decreasing trend over the course of training. This phenomenon can be attributed to the fact that each non-terminal time step within the environment results in a negative reward for the agent. Consequently, the agent is incentivized to complete each episode as swiftly as possible.

4.2.2 Evaluations of Trained Policies

We have assessed the performance of the trained policies in the identical environment where they underwent training, under various levels of bias and variance in the noise, both with and without the application of a denoiser (Low Pass Filter and Kalman Filter). The denoiser was applied to both noisy observation spaces as defined in eq. 4.10 to get the denoised observation space as eq 4.11 for both environments. The Low Pass Filter (LPF) utilized in our experiments is a 2^{nd} order Bessel filter with a cutoff frequency of 2. Its transfer function is given by [56] as given in equation 4.15.

$$G(s) = \frac{3}{s^2 + 3s + 3} \tag{4.15}$$

4.2.2.1 Unbiased Noise

This subsection presents the evaluation results of three policies trained in *Environment 2* (referred to as **Policy 4, 5, and 6**), which includes dynamic obstacles. These policies were tested in the same environment under varying levels of unbiased noise ($\mu = 0, 0 \leq \sigma \leq 3.0, \Delta\sigma = 0.1$), with comparisons made between their performances with and without the implementation of a denoiser. We will use the term *baseline* to refer to the scenario where there is no standard deviation in the noise ($\sigma = 0$) for the remainder of this subsection.

Policy 4: Figure 4.5(a) visualizes the performance of Policy 4. At baseline, Policy 4 achieves a success rate of around 50%. This is followed by a slight increase till 60% at 0.2 standard deviation then gradually drops till near-zero success at standard deviation of 3.0. The drop-off is further slowed down significantly by the Low Pass and Kalman Filters, still achieving around 30% success rate at $\sigma = 3$.

Policy 5: From Figure 4.5(c), Policy 5 outperforms Policy 4 at baseline, achieving around 60% success rate. The success rate drops as σ increases, but not at as rapidly as Policy 1. With a denoiser this drop is reduced significantly, and the success rate gradually decreases to around 40% at $\sigma = 3$. Policy 5 shows a higher and more consistent success rate as compared to Policy 4 across the range of σ values.

Policy 6: Figure 4.5(e) visualizes the performance for Policy 6 . The performance starts off with baseline success rate of around 50%. In the case without a denoiser, a small increase in success rate is seen until $\sigma = 0.8$, and then proceeds to rapidly decrease.This can be attributed to the fact that this policy was trained with noise around that range($\sigma = 1.0$) .With the denoisers, the drop in success rate is slowed down significantly, with a success rate of around 30% and 40% at $\sigma = 3$ for the Low Pass Filter and the Kalman Filter respectively.

4.2.2.2 Bias-only Noise

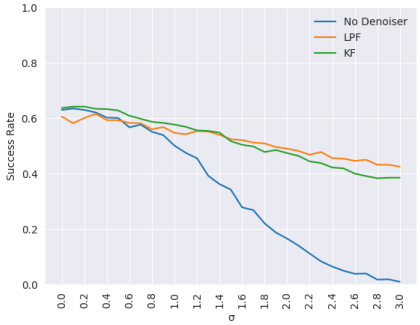
This subsection contains the evaluation results of the three trained policies in *environment 2* (**Policy 4, Policy 5 and Policy 6**) with varying degrees of bias-only noise ($0 \leq \mu \leq 0.3, \sigma = 0, \Delta\mu = 0.01$), comparing their performances with and without a denoiser.

Figure 4.6 confirms the fact that the denoisers have no way of mitigating the negative effects of bias, as we see the same trend across the three cases for each policy. The success rate for Policy 4 and 5 decreases as bias increases (Figure 4.6(a), (b)).

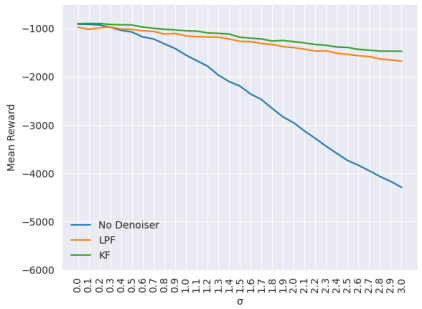
Policy 4:Figure 4.6 (a,b) shows the result of Policy 4 against biased noise. The Policy starts off with a success rate of around 0.7 at no bias and drops to around 0.1 at $\mu = 0.3$. The behaviour of the plots for the denoiser as well as no de-noiser case is simlar indicating that deniosers have no way of mitigating biased noised.

(a) Policy 4, Environment 2

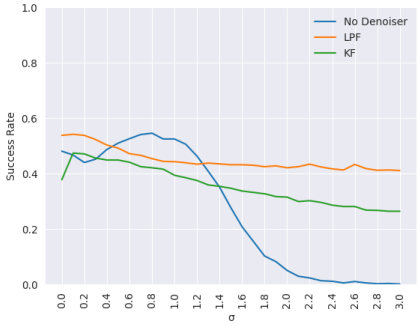
(b) Policy 4, Environment 2



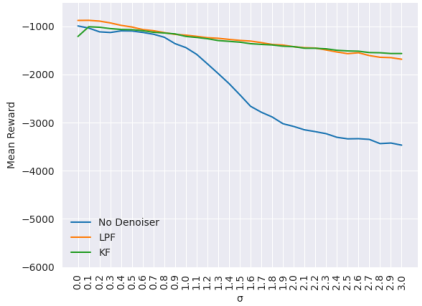
(c) Policy 5, Environment 2



(d) Policy 5, Environment 2



(e) Policy 6, Environment 2



(f) Policy 6, Environment 2

Figure 4.5: **Success Rate and Mean Reward Analysis with Unbiased Noise** ($\mu = 0, 0 \leq \sigma \leq 3.0$) **in Meters:** (a, b) Policy 4's success rate begins relatively stable but declines sharply as σ increases, reaching a 0% success rate near $\sigma = 2.4$. The implementation of denoisers (LPF and KF) appears to moderate the decline. The mean reward also diminishes with higher σ, with denoisers mitigating the downward trend. (c, d) Policy 5 experiences a decrease in success rate as σ rises, maintaining a moderate initial success rate but deteriorating more gradually with the aid of denoisers. Similarly, the mean reward decreases less sharply when denoisers are applied. (e, f) Policy 6 starts off with a success rate of 0.4, increases slightly to 0.55, then plumates to 0 at $= 2.4$. Denoisers smoothen the small peak, but also slow down the drop in performance. The mean reward reflects a similar pattern, with denoisers reducing the rate of decline.

Policy 5: Figure 4.6 (c,d) shows the result of Policy 5 against biased noise. Policy 5 starts with a success rate of around 0.6 at $\mu = 0.6$ then drops to near 0 as μ approaches 0.3. The drop is more steeper than that of Policy 4. Denoisers, again, seem to have no effect on improving the drones performance.

Policy 6: Figure 4.6 (e,f) shows the result of Policy 6 against biased noise. Policy 6, trained on noise with variance of 1.0 starts off at a success rate of around 0.5 and does not drop as in the case of Polcies 4 and 5. We can conclude that training on noise with variance helps in immunity against noise containing only a bias and no variance. The LPF performs very similar to the no denoiser case, while the KF performs a little worse than the two others. Conclusively, the denoisers do not help improve performance in this case as well.

4.2.2.3 Biased Noise

In this subsection, We present the evaluation outcomes of Policies 4, 5, and 6 within *Environment 2*, where the noise levels vary under biased conditions ($0 \leq \mu \leq 0.3, 0 \leq \sigma \leq 3.0, \Delta\mu = 0.01, \Delta\sigma = 0.1$). We compare the performances of these policies without the application of a denoiser.

Policy 4: From the plot 4.7 (a), we can infer that in general, the performance decreases with increase in either of σ or μ. However there is a region of special interest at $0.2 \leq \mu \leq 0.3$. We can observe that at this region, additional σ improves the performance till about $\sigma = 1.0$. This can be used to our advantage by adding artificial noise to improve the performances of the policies. We discuss this more detail in the discussions section. Another such region is encompassed in $0.7 \leq \sigma \leq 1.2$, where additional μ improves the performance till $\mu \approx 0.12$.

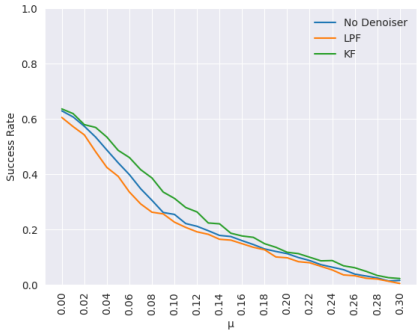
Policy 5: From the plot 4.7 (b), a similar observation can be made for Policy 5, with performances dropping with increase of either of μ or σ. Another region of interest as described for Policy 4, can be seen for $0.05 \leq \mu \leq 0.12$

Policy 6: The heat map for Policy 6 4.7 (c) trained with standard deviation of 1 is very different from the previous 2 policies. We can observe that the performance exclusively depends only on σ, decreasing as σ increases. Although there are no special regions as described for Policies 4 and 5, we can see that this Policy is almost immune to bias in noise.

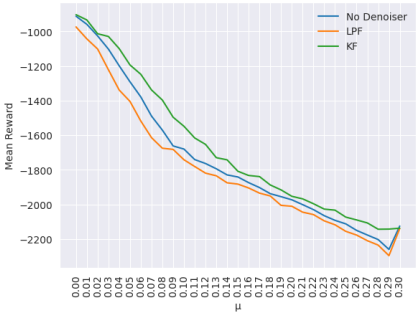
The plot for mean reward during training is in 4.8 for each of the cases.

(a) Policy 4, Environment 2

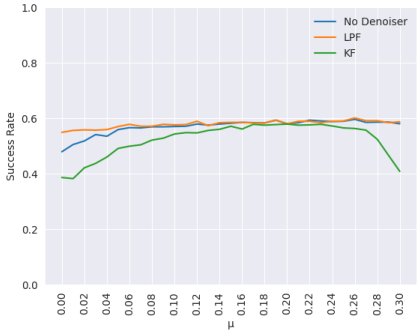
(b) Policy 4, Environment 2



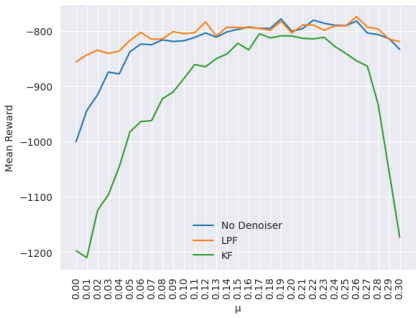
(c) Policy 5, Environment 2



(d) Policy 5, Environment 2

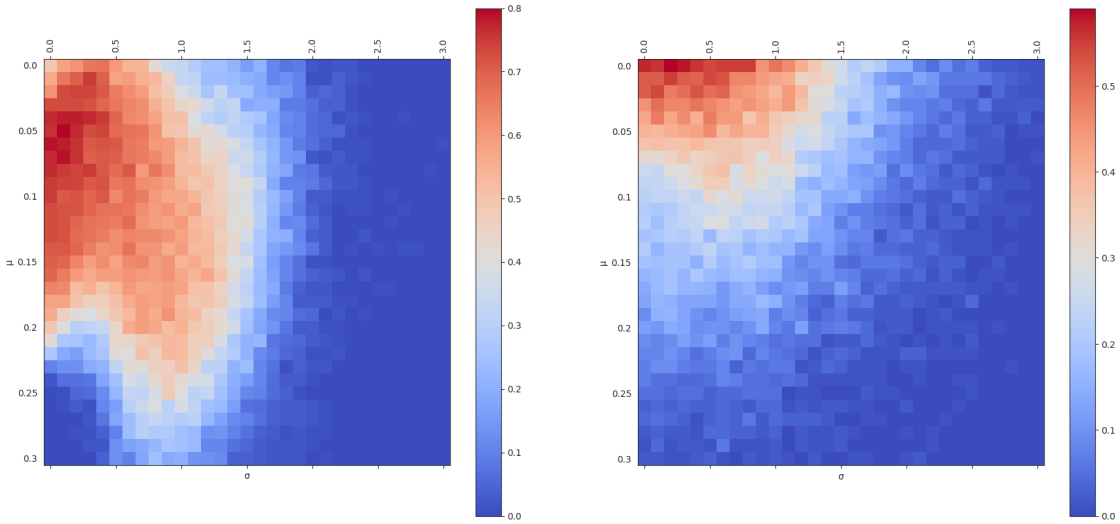


(e) Policy 6, Environment 2



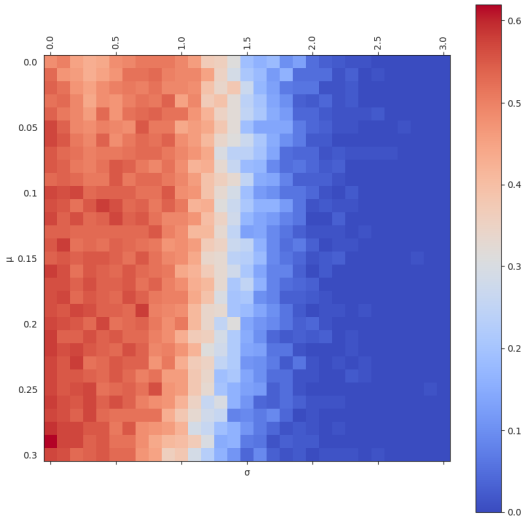
(f) Policy 6, Environment 2

Figure 4.6: **Success Rate and Mean Reward Analysis against Bias-Only Noise** ($0 \leq \mu \leq 0.3, \sigma = 0$): (a) Policy 4's success rate drops slowly with an increase in μ, hitting around 0.1% at $\mu = 0.30$. (c) Policy 5 starts off worse with a succes rate of around 0.6, and drops to lower values faster than policy 1. (e) Policy 6's success rate is between 0.5 and 0.6 thoughtout, except a slight underperformance using KF. We see in all three cases that the denoisers can provide no assistance to the policies when faced with bias-only noise.



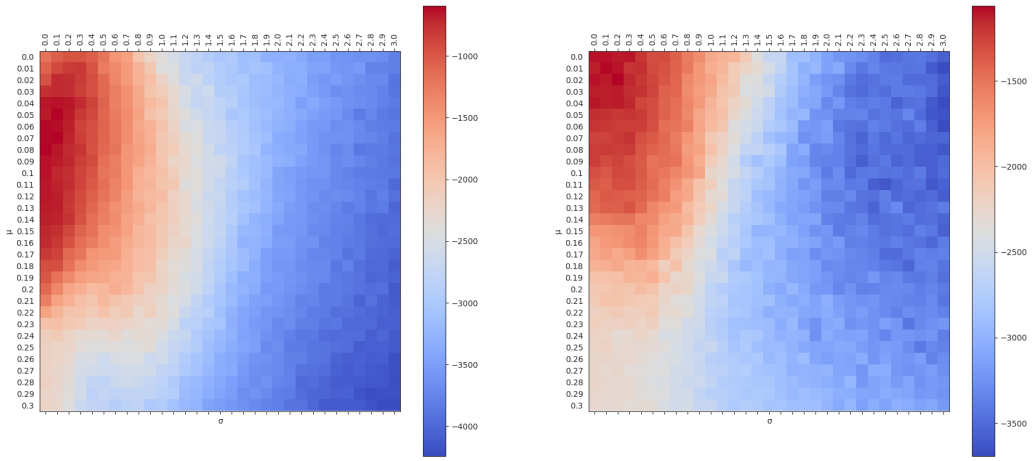
(a) Policy 4, Environment 2

(b) Policy 5, Environment 2



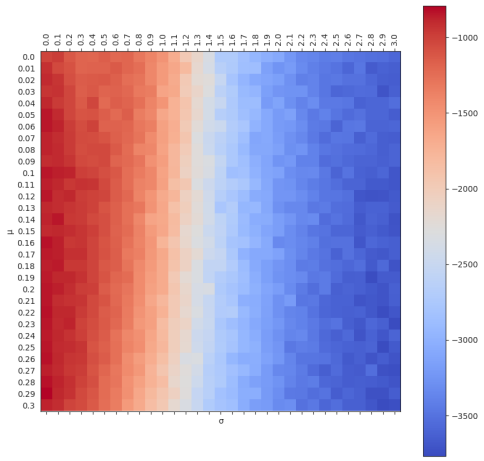
(c) Policy 6, Environment 2

Figure 4.7: **Rate of Success in the Presence of Biased Noise** ($0 \leq \mu \leq 0.3, 0 \leq \sigma \leq 3.0$): This heatmap uses a color gradient where red indicates a high success rate and blue signifies a lower success rate. The color gradient for each heatmap is detailed by a colorbar positioned to the right of the chart. (a) In policy 4, we observe a region interest at $0.2 \leq \mu \leq 0.3$ where adding where adding additional σ improves performance. This can be utilised to improve performance of the agent. (b) Policy 5 shows an increase in performance with both μ and σ with a small region of interest at $0.05 \leq \mu \leq 0.12$ (c) Policy is almost immune to changes in σ while showing linear with increase in μ



(a) Policy 4, Environment 2

(b) Policy 5, Environment 2



(c) Policy 6, Environment 2

Figure 4.8: **Mean Reward Plots in the Presence of Biased Noise** ($0 \leq \mu \leq 0.3, 0 \leq \sigma \leq 3.0$): This heatmap uses a color gradient where red indicates a high success rate and blue signifies a lower success rate. The color gradient for each heatmap is detailed by a colorbar positioned to the right of the chart. These are corresponding heatmaps for 4.7

4.3 Discussion

We have trained policies with noise of three different variances and tested their performance against different combinations of bias and variance (fig 4.6). We have also tested the effectiveness of denoisers for unbiased and bias-only noise. Fig 4.7 confirms that denoisers have no way of assisting in helping with bias-only bias, while from fig 4.5 we can conclude that denoisers in general help in improving performance for unbiased noise. Fig 4.6 gives the trade offs between bias and variance. This can be

leveraged greatly in two ways, first- choosing amongst sensors and techniques for localisation yielding noise in observation with different biases and variances, and second- artificially injecting appropriate noise to data received from noisy sensors to improve performance of a trained policy. The noise in localisation affects the performance of a trained RL policy as shown by the plots, thus making sensor choosing an important aspect to consider while utilising RL for UAV navigation. We propose an RL policy being trained for UAV navigation using any method can be evaluated on varying bias and variance of observation space noise, and obtain plots similar to those of fig 4.6. These plots can then be utilized to choose between various sensors, and localisation techniques having different levels of bias and variance.Moreover, the policy's performance can be enhanced by deliberately introducing suitable noise, as indicated by these plots. For example in fig 4.6 (a) for the region $0.2 \leq \mu \leq 0.3$ of special interest as described above, we can observe that, additional σ improves the performance till about $\sigma = 1.0$. Thus if our sensor has noise with a low variance and bias in this region, we can artificially add unbiased noise with appropriate standard deviation derived from the plot to existing sensor noise in order to improve performance. This section of the research was entirely simulated. It is our intention to put this to the test in the future for the scenario with a dynamic environment.

4.4 Simulation Trajectories

In this section, We show some simulation trajectory for the models above. Since the environment is dynamic, We have provided snapshot of 4 figures to capture the trajectory with more intuition as the obstacle's location changes so does the trajectory for drone. Kindly note that the snapshots are from one episode and not different episode. It is the trajectory from starting point to target.

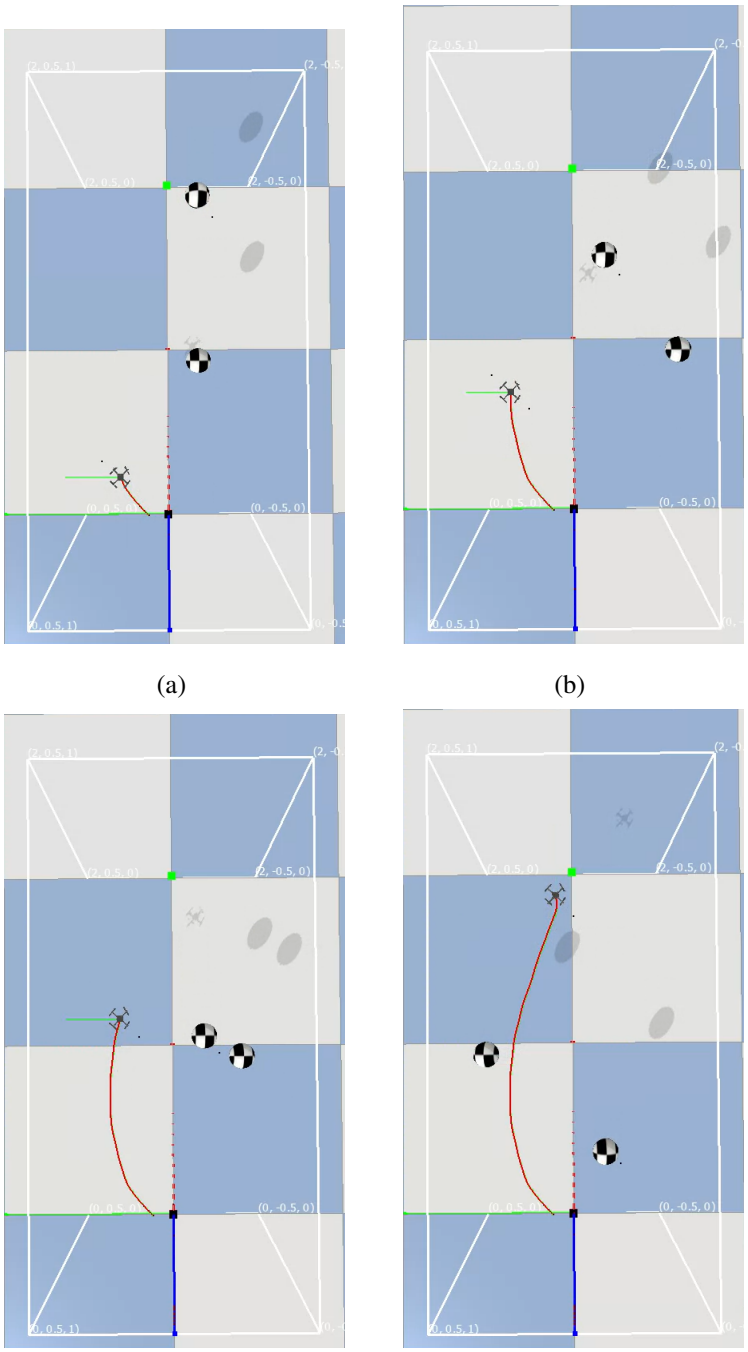
4.4.1 Effects of Noise on Training : No noise, No Denoiser

Figure 4.9 4.10 4.11 illustrates the simulation trajectories within an environment 2, with dynamic obstacles devoid of both noise and denoising mechanisms. The depicted trajectories are highlighted by a red line for each scenario. In each of these 3 scenarios the trajectory successfully reaches the designated target. Additionally, a green line emanates from the drone (in some figures can also be yellow or red), indicate the maintained safe distance from the nearest obstacle throughout its journey.

4.4.2 Effects of Noise on Training : $\mu = 0$, $\sigma = 1.5$ and No Denoiser

This section evaluates the performance of various policies in an environment characterized by unbiased noise with $\mu = 0$, $\sigma = 1.5$, and the absence of a denoising mechanism. Observations from Figure, 4.12, 4.13 and 4.14 reveal that policies represented in scenarios Fig. 4.13 and 4.14 are trained with a non-zero σ, enabling them to adapt and perform effectively in the presence of unbiased noise. In

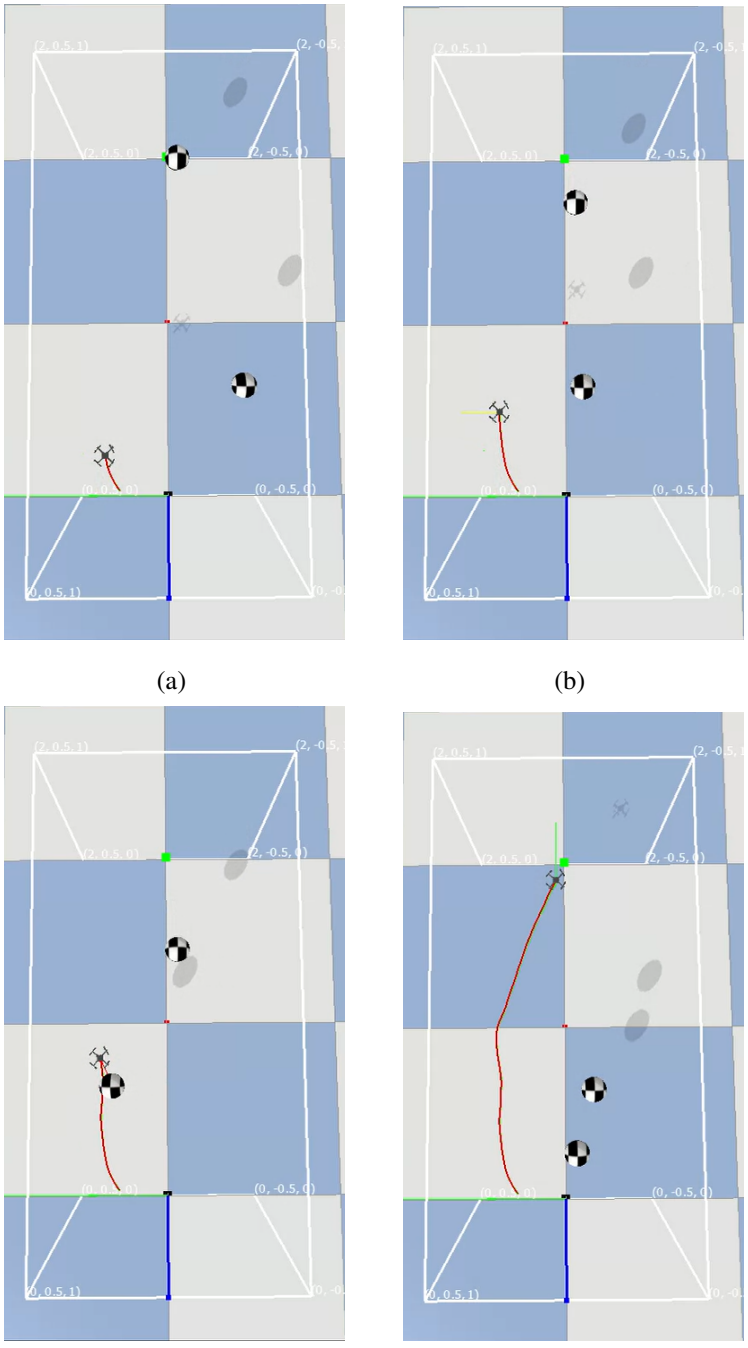
[0]The code for training and evaluating the policies, together with comprehensive results, may be accessed at https://github.com/BhaskarJoshi-01/DroneControl-Dynamic



(a) (b)

(c) (d)

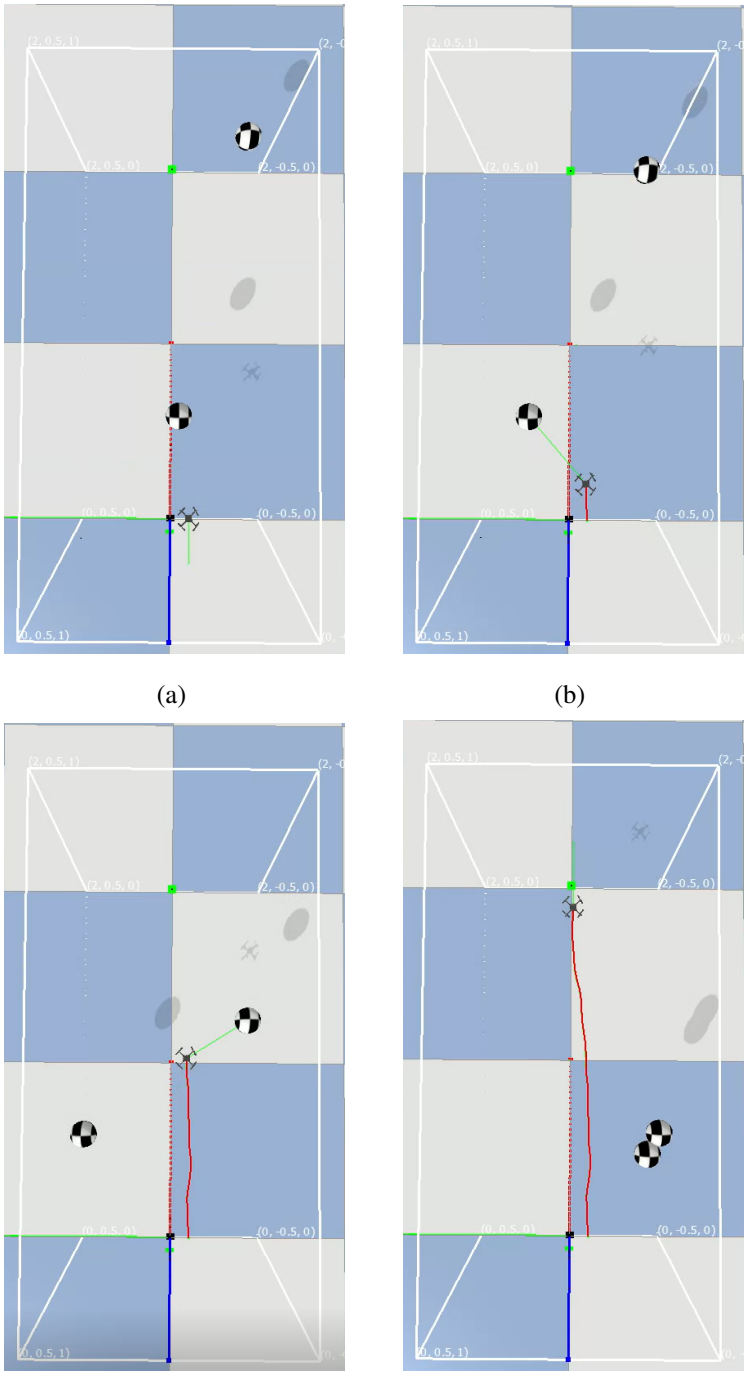
Figure 4.9: **Simulation Trajectory for Environment 2: Model 4, with No noise and No Denoiser**



(a) (b)

(c) (d)

Figure 4.10: **Simulation Trajectory for Environment 2: Model 5, with No noise and No Denoiser**



(a) (b)

(c) (d)

Figure 4.11: **Simulation Trajectory for Environment 2: Model 6, with No noise and No Denoiser**

contrast, the policy depicted in Figure 4.12, which was not trained under noisy conditions, exhibits poor performance when exposed to high levels of unbiased noise.

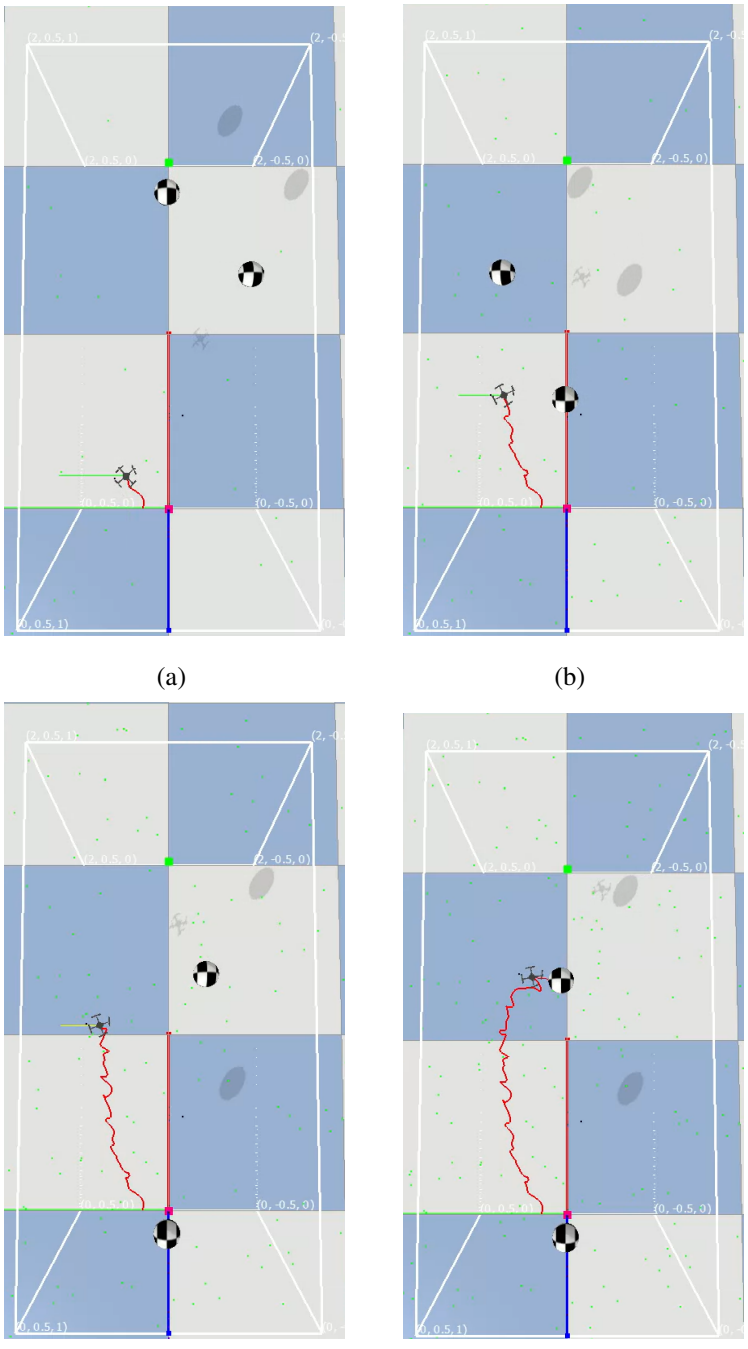
4.4.3 Unbiased Noise and Filters

This subsection presents an evaluation of Policy 4, under conditions of unbiased noise ($\mu = 0$, $\sigma = 0.1$), with the incorporation of filtering techniques. Figure. 4.15, 4.16 and 4.17 illustrates the trajectory outcomes when Policy 4 is subjected to these conditions. Notably, scenarios 4.16 and 4.17 have Low Pass Filter (LPF) and Kalman Filter (KF), respectively. These filters enable successful target achievement in an environment influenced by unbiased noise with parameters $\mu = 0$ and $\sigma = 0.1$. The efficacy of these filtering methods is evident, highlighting their utility in enhancing navigational accuracy under noisy conditions.

4.4.4 Handling Biased Noise $\mu = 0.1$ and $\sigma = 0$

This subsection explores the effect of biased noise ($\mu = 0.1$ and $\sigma = 0$) on navigational strategies and evaluates the effectiveness of denoising approaches. In Figure 4.18, 4.19 and 4.20 scenarios: 4.18 and 4.19 demonstrate trajectories under biased noise conditions without filter and with the application of a Low Pass Filter (LPF), respectively. The trajectory in Figure 4.19, utilizing LPF, appears smoother compared to Figure 4.18, which lacks a denoising mechanism. However, both scenarios exhibit an inability to circumvent obstacles effectively.

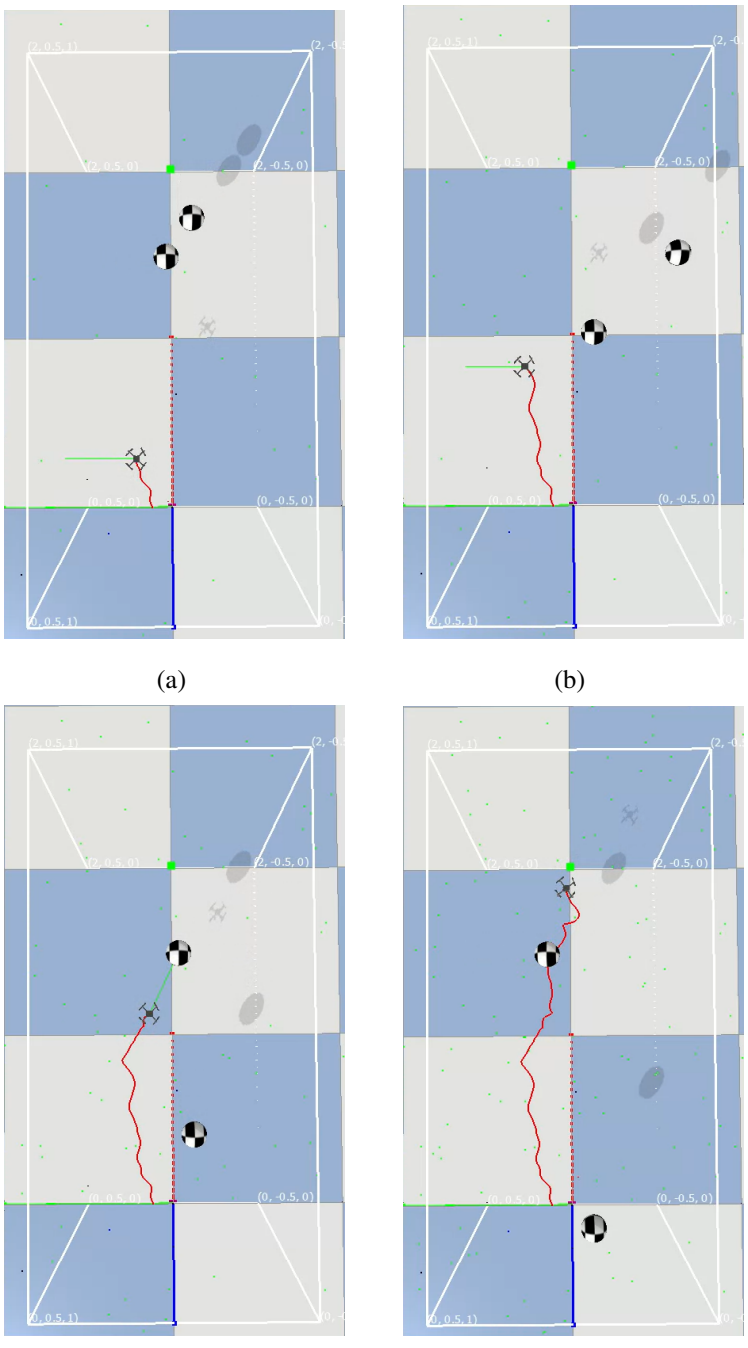
Introduction of unbiased noise ($\sigma = 0.7$), as depicted in Figure 4.20, significantly enhances obstacle avoidance, suggesting that the adverse effects of bias can be mitigated by introducing a specific magnitude of noise variance. This finding underscores the potential for strategic noise manipulation to improve navigational outcomes under biased conditions.



(a) (b)

(c) (d)

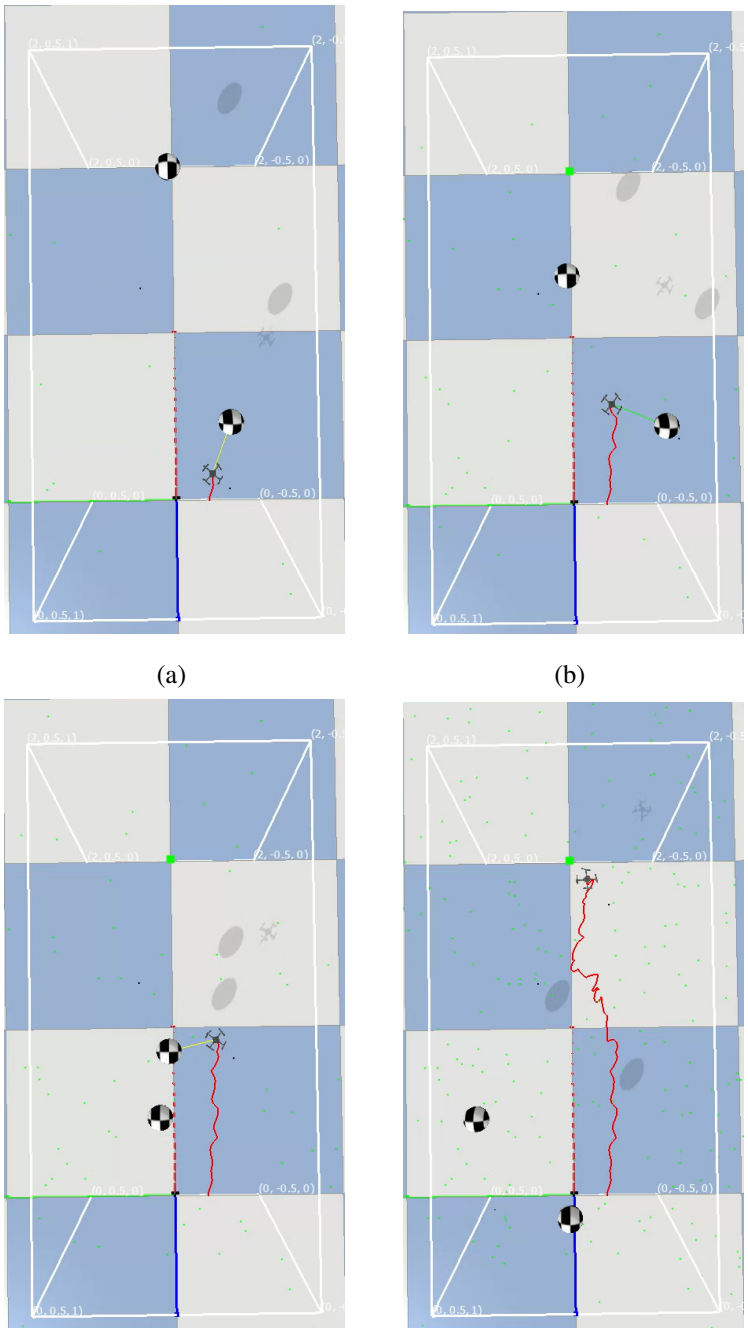
Figure 4.12: **Simulation Trajectory for Environment 2: Model 4 with $\mu = 0$, $\sigma = 1.5$ and No Denoiser**



(a) (b)

(c) (d)

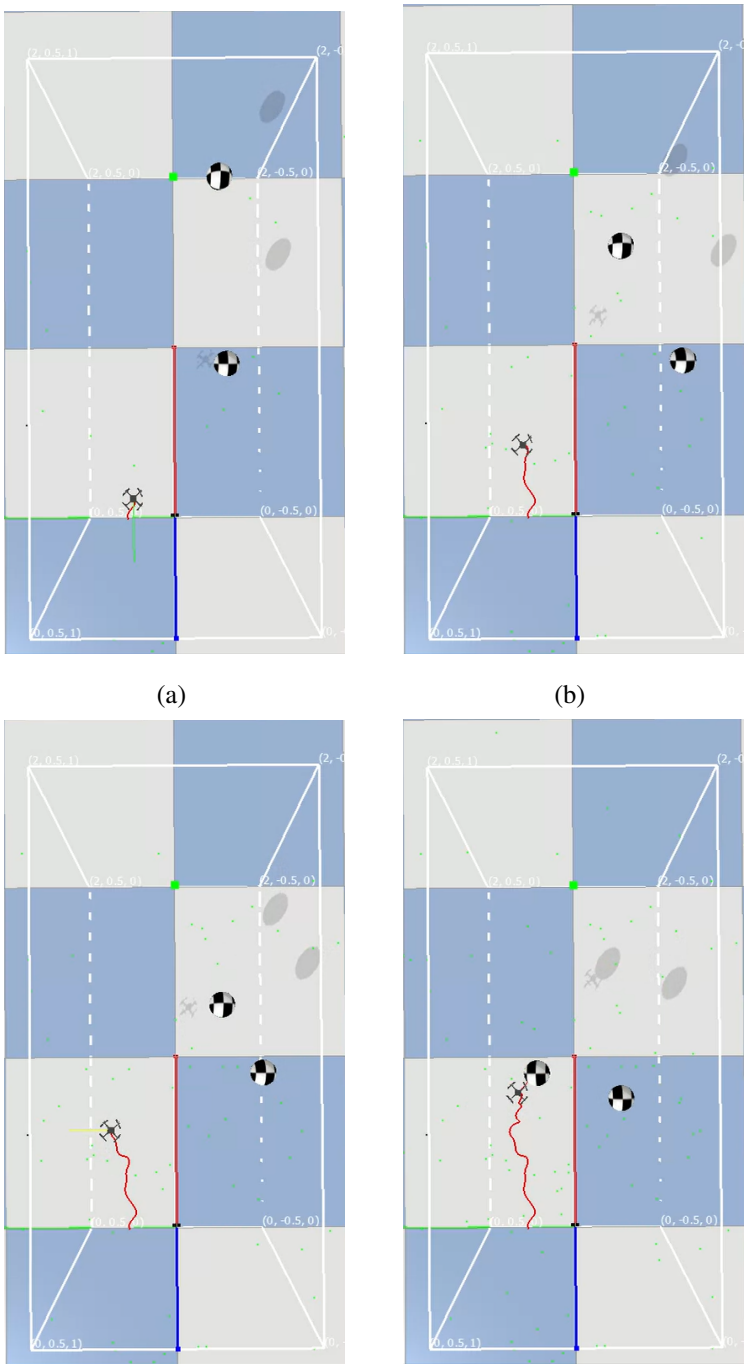
Figure 4.13: **Simulation Trajectory for Environment 2: Model 5 with $\mu = 0$, $\sigma = 1.5$ and No Denoiser**



(a) (b)

(c) (d)

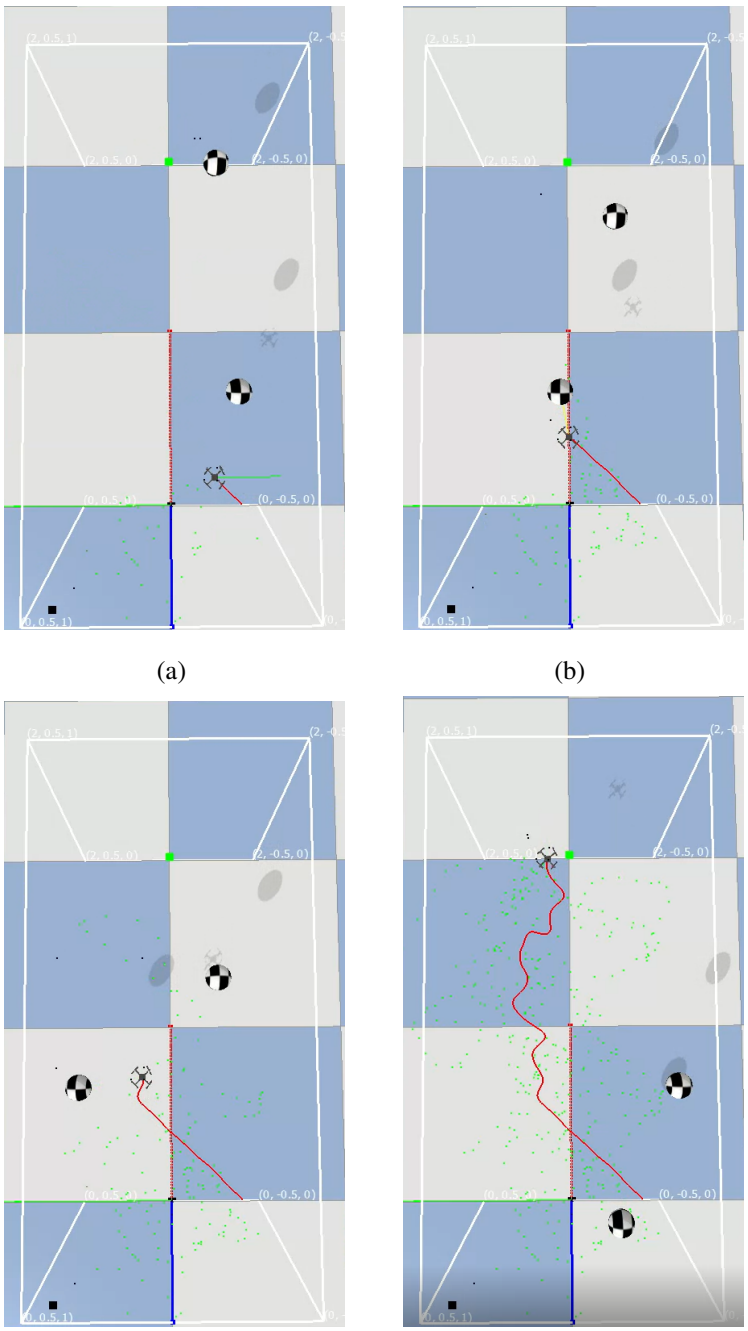
Figure 4.14: **Simulation Trajectory for Environment 2: Model 6 with $\mu = 0$, $\sigma = 1.5$ and No Denoiser**



(a) (b)

(c) (d)

Figure 4.15: **Simulation Trajectory for Environment 2: Model 4 with $\mu = 0$, $\sigma = 0.1$ and No denoiser**



(a) (b)

(c) (d)

Figure 4.16: **Simulation Trajectory for Environment 2: Model 4 with $\mu = 0$, $\sigma = 0.1$ and LPF**

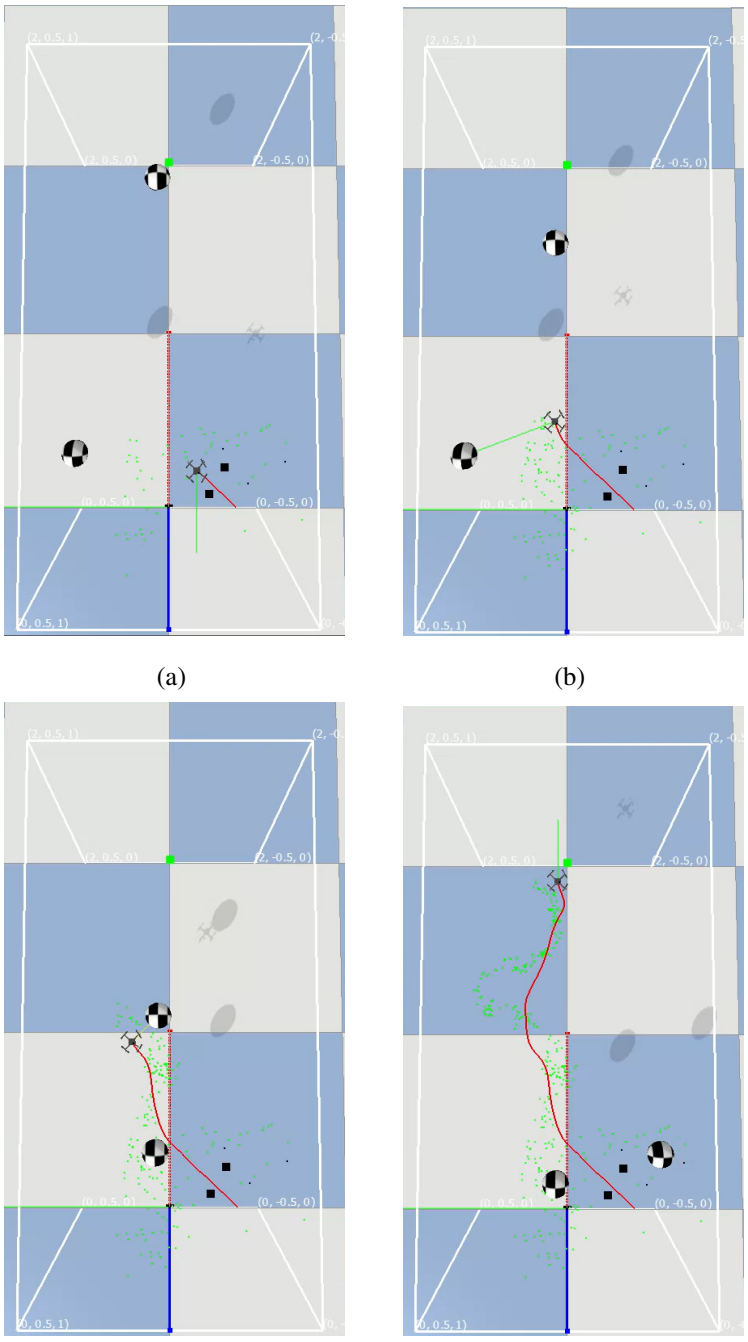
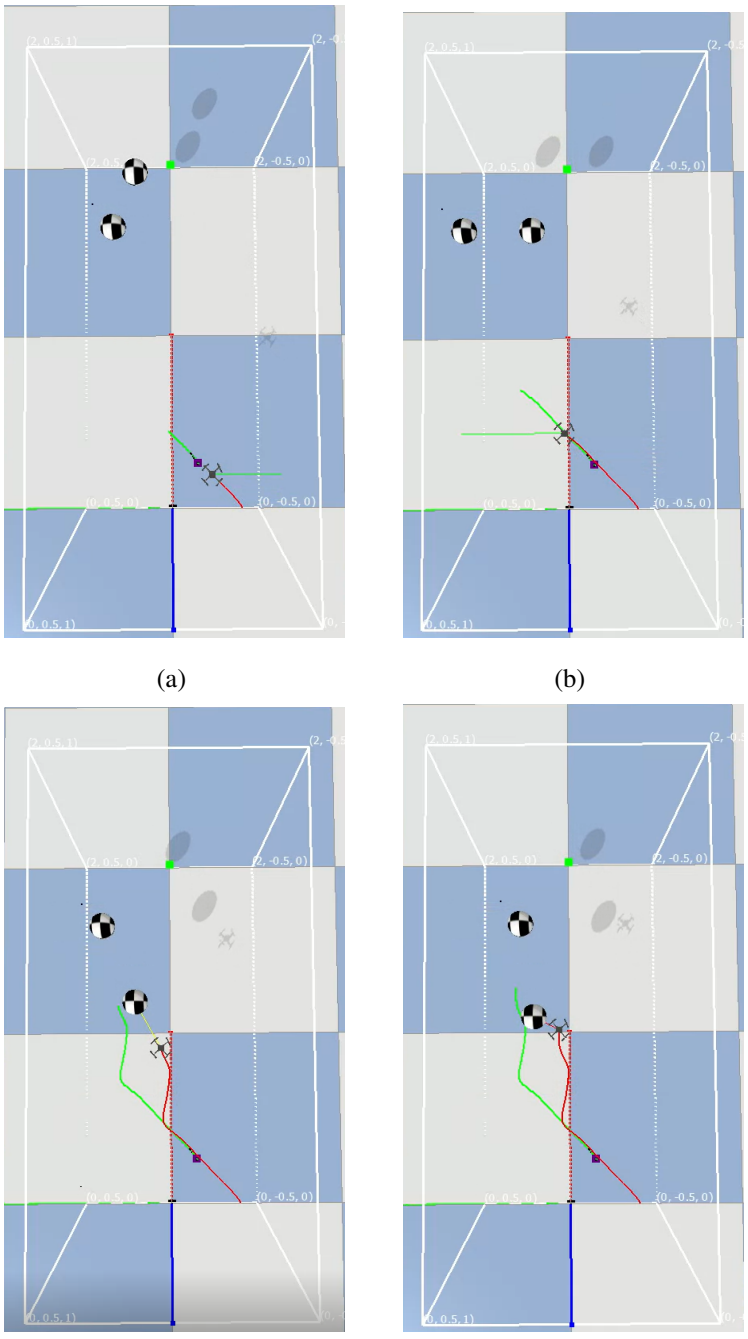


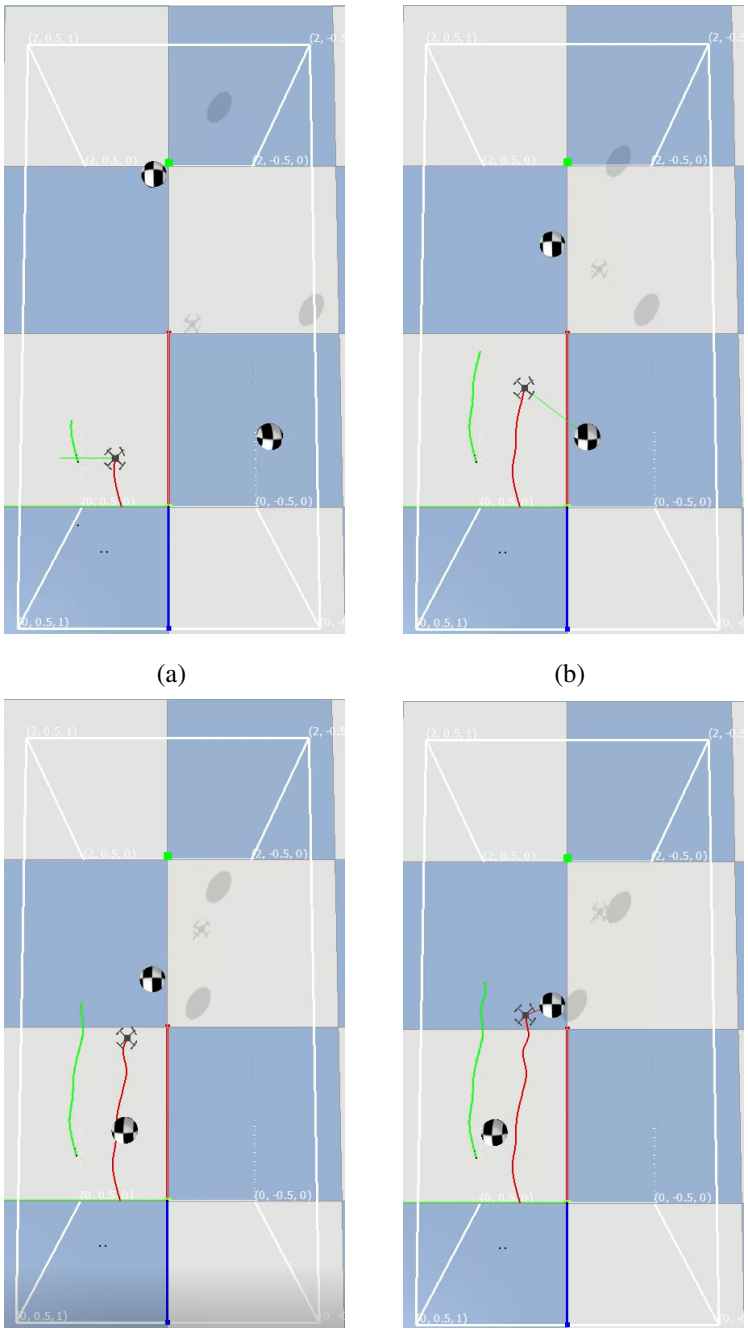
Figure 4.17: **Simulation Trajectory for Environment 2: Model 4 with $\mu = 0$, $\sigma = 0.1$ and KF**



(a) (b)

(c) (d)

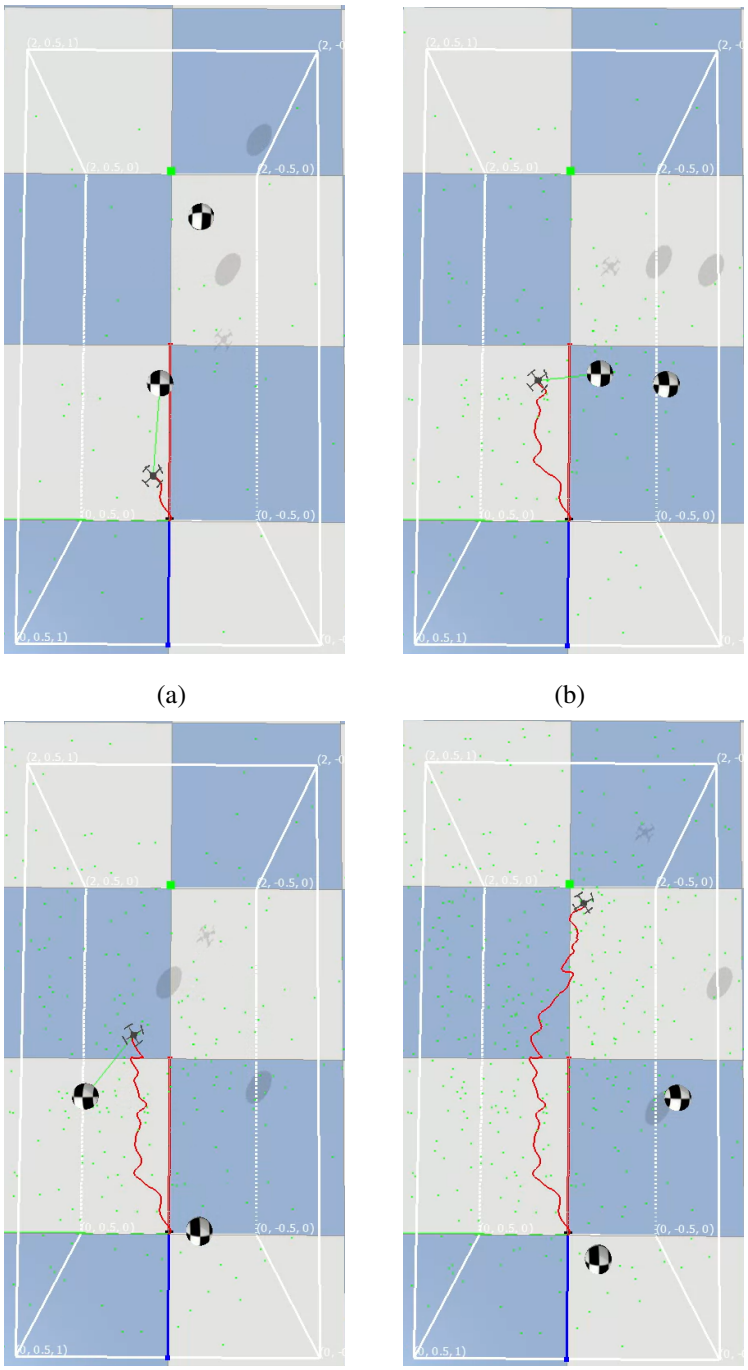
Figure 4.18: **Simulation Trajectory for Environment 2: Model 4 with $\mu = 0.1$ and $\sigma = 0$ and No Denoiser**



(a) (b)

(c) (d)

Figure 4.19: **Simulation Trajectory for Environment 2: Model 4 with $\mu = 0.1$ and $\sigma = 0$ and LPF**



(a) (b)

(c) (d)

Figure 4.20: **Simulation Trajectory for Environment 2: Model 4 with $\mu = 0.1$ and Injection of Unbiased Noise with $\sigma = 0.7$**

Chapter 5

Simulation to Reality Transfer for Static Obstacles

This chapter delves into the intricate process of deploying a trained policy in Pybullet to the real-world environment using a sophisticated Motion Capture (MoCap) system and Crazyflie drone. The chapter talks about MoCap, setup in real world describes about implementation and testing protocol, evaluation of noise impact and summary of real world experiments.

5.1 MoCap

Motion Capture (MoCap) systems are advanced tracking technologies that utilize a network of cameras to monitor markers affixed to objects of interest, such as the Crazyflie drone. These markers are pivotal for the system's ability to ascertain the object's precise location and orientation within a global reference framework. The system's recognition of the marker positions on the object facilitates the calculation of its pose (position and orientation) with high accuracy.

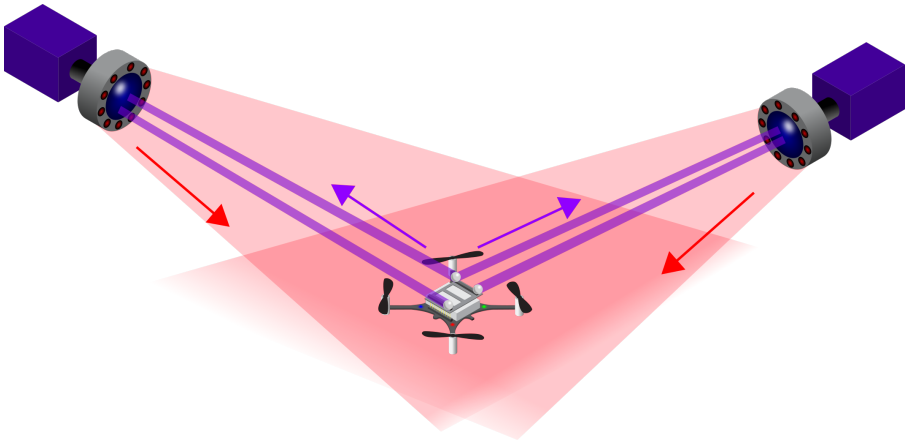


Figure 5.1: Illustration of Motion Capture positioning (Image Reference: Bitcraze Documentation)

The data acquisition process involves multiple cameras—six in our setup—capturing the spatial positions of the markers. This information is then relayed to a central processing unit that performs the

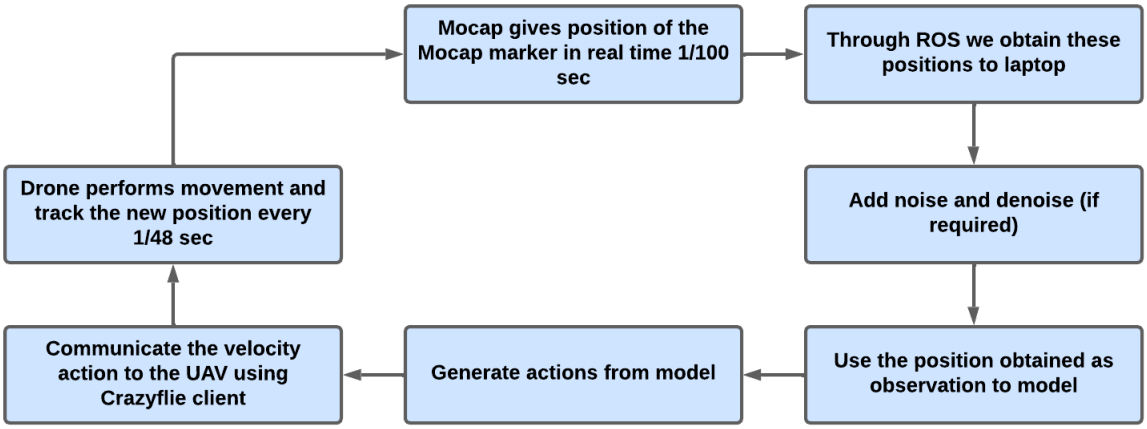


Figure 5.2: The image represents the flowchart of method of deploying the model in simulation to the real world.

necessary computations to determine the markers' locations and, by extension, the object's pose. In the specific scenario involving the Crazyflie drone, its positional data is wirelessly communicated to the drone via a Crazyradio link. This data integration into the drone's control system is crucial for enabling precise maneuvering and stabilization.

The arrangement of markers on the Crazyflie is a critical factor that influences the MoCap system's ability to accurately determine the drone's pose. A distinctive marker configuration can significantly enhance the system's efficiency in tracking the object amidst other items within the same environment. To optimize tracking accuracy while minimizing additional weight, small passive markers are directly affixed to the Crazyflie's structure. This approach avoids the weight penalty associated with attaching larger tracking modules or decks, ensuring that the drone's agility and performance are not compromised.

5.2 Experimental Setup in the Real World

We employed the Crazyflie 2.1 UAV, augmented with motion capture markers to facilitate real-time, high-precision localization. This setup ensures meticulous control over the observation perturbations, a critical aspect for validating our simulation-to-reality transfer methodologies. The UAV's position data, captured through motion capture technology, is relayed to our computational framework. Herein, it undergoes a sequential process beginning with intentional corruption via a bespoke noise generator, followed by an optional denoising phase, depending on the experimental conditions. Subsequently, the observation is derived and relayed to the policy algorithm.

The policy algorithm, upon receiving the modified observation, computes the requisite action. This action, in turn, is translated into a velocity command, meticulously crafted to guide the UAV's move-

ment. The command is then dispatched to the UAV, marking the completion of a singular timestep in our experiment.

We delineate an episode's success by the UAV's ability to navigate within a predefined proximity to the designated target. Notably, the transition from simulation-based training to real-world application was seamless, with the trained agent's network being directly implemented on the UAV sans any modifications. This seamless sim-to-real transition underscores the robustness and adaptability of our training methodologies, holding promise for future explorations in autonomous UAV navigation.

For the environment with static obstacles, policies underwent training for 5 million steps each, using RTX 3050 GPU and took 5 hours. The evaluation process for a policy involves running it across 1000 episodes, where each episode is capped at 1000 timesteps. This evaluation, conducted on a set of parameters—mean (μ), standard deviation (σ), and a denoiser—takes roughly 10 minutes to complete without GUI Rendering. To assess the effectiveness of the UAV navigation system in static environments, two modes of evaluation are implemented: with and without graphical user interface (GUI) rendering. Conducting evaluations with GUI rendering necessitates approximately 15 seconds to complete a successful episode. Notably, this duration mirrors the time required for analogous real-world experiments, highlighting the simulator's fidelity and its utility in realistic scenario testing.

In our study, We meticulously constructed a physical testing environment ref fig 5.4 that mirrors the simulated environment ref fig 5.6 where our UAV control policies were initially trained. This approach is vital for evaluating the fidelity of sim-to-real transfer techniques.

A critical aspect of our real-world setup involves the transferring of the simulated environment's - floating spherical obstacles were replaced with physical cylindrical pipes, each measuring 10 centimeters in diameter, to maintain consistency with the simulation parameters ref fig 5.5. The starting and target locations are explicitly marked within the operational area. The target zone is surrounded by a 10-centimeter radius circle, signifying the success threshold identical to the simulation environment, where $\epsilon_{success}$= 0.1 meter denotes the margin for error in completion.

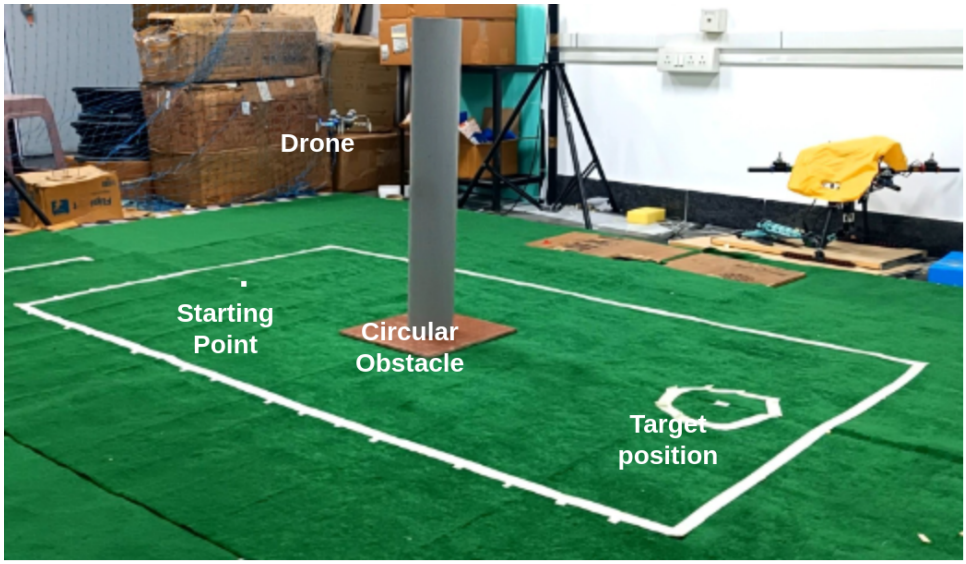


Figure 5.4: **Real World Setup (Side View)**: The setup, depicted in consists of a delineated operational area marked by distinct white boundary lines, within which a Crazyflie 2.1 drone undertakes navigation tasks.

Figure 5.3: **Craziflie 2.1 Drone**: The drone used for experiments having MOCAP passive markers placed on it.

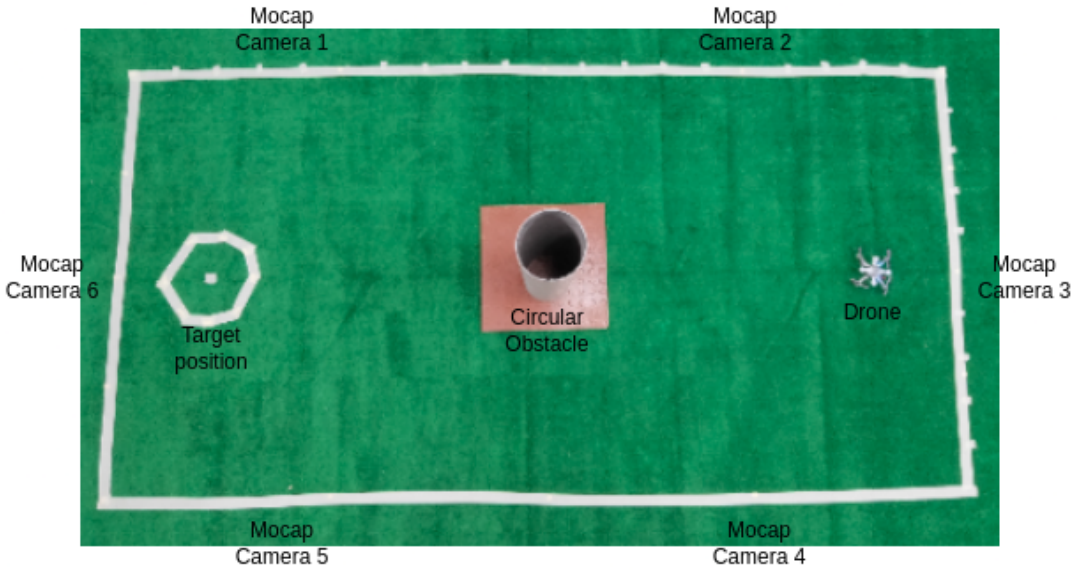


Figure 5.5: **Real World Setup (Top View)**: There are 6 mocap cameras well calibrated and the Craziflie 2.1 (drone) at the right moves to the Target Position by avoiding the Circular Obstacle that was supposed to be on the plane of its trajectory

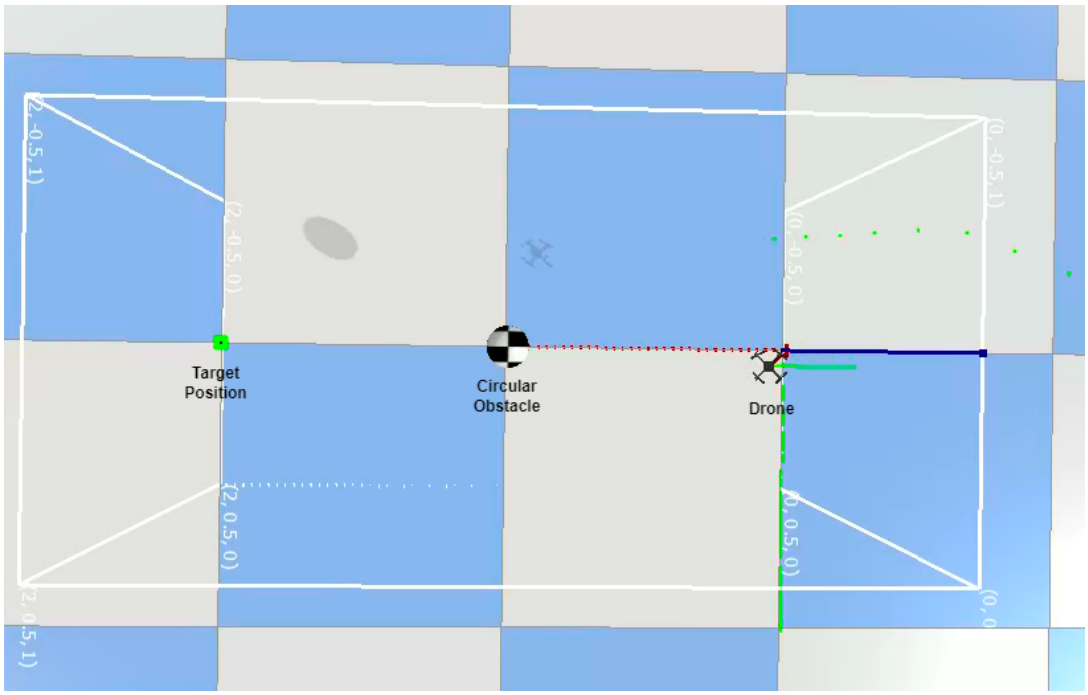


Figure 5.6: **Simulation Setup (Top View)**: Pybullet View for the Fig. 5.5 above

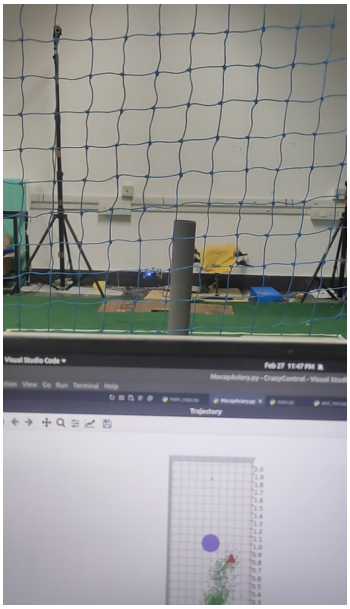


Figure 5.7: **Real World Setup (Top View)**: Snapshot of Experiment being performed, where the UAV is flying to avoid the obstacle and the trajectory plot is being created on the laptop screen.

5.2.1 Implementation and Testing Protocol

our control policy, designed to be robust and adaptable, was directly deployed on the Crazyflie 2.1 drone without necessitating modifications. This direct applicability underscores the high-level abstraction and generalization capabilities of our policy design. To evaluate the efficacy and reliability of our approach, We focused our testing on two distinct policies—Policy 1 (No Noise $\mu = 0$, $\sigma = 0$) and Policy 2 (Low Noise $\mu = 0$, $\sigma = 0.1$) —both of which demonstrated promising results during the simulation phase for static obstacles. Another note is We demonstrated Sim to Real for our first paper.

Given the practical constraints on the number of trials executable in a physical setting, each policy underwent a series of five trials to ensure the statistical significance of the results. The real-world experiments aimed to replicate the conditions of the simulation as closely as possible, including the introduction of controlled unbiased and biased noise to assess the policies' robustness and adaptability.

5.2.2 Evaluation of Noise Impact and Performance Metrics

A noteworthy aspect of our real-world experimentation involves the systematic introduction of noise to the UAV's localization system. Initial tests were conducted under conditions of minimal bias (0.15 meters) without added variance, resulting in significantly lower success rates. This outcome aligns with simulation expectations, as the absence of variance in localization estimates can exacerbate the impact of bias.

Subsequently, We introduced unbiased noise with a standard deviation $\sigma = 0.8$, which remarkably improved the success rates for both policies, achieving target acquisition in all trials. This improvement suggests that introducing a moderate level of variance can enhance the robustness of control policies against localization bias.

However, increasing the standard deviation further to $\sigma = 1.3$ led to a decrease in success rates, illustrating the delicate balance between beneficial and detrimental levels of noise in real-world UAV navigation tasks. These findings were consistent across both policies, with Policy 2 generally outperforming Policy 1, mirroring the trends observed in simulation studies.

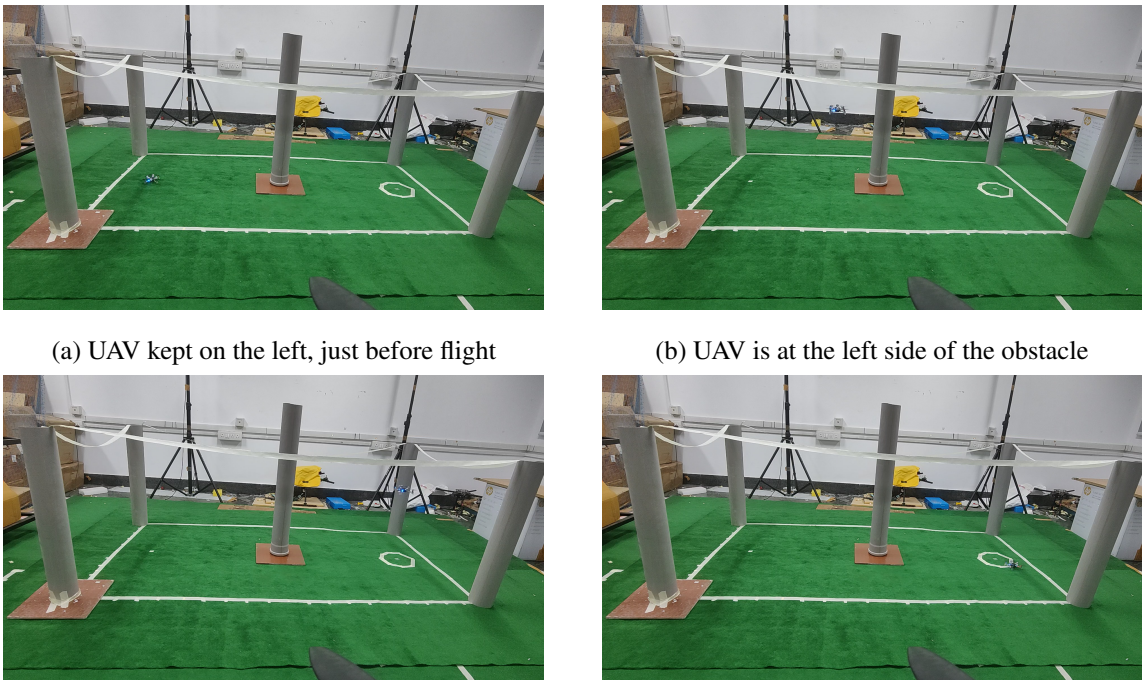
5.3 Summary of Real-World Experimentation Results

The summarized outcomes of our real-world tests are presented in Table 3.1, which details the success and failure rates of each policy under various noise conditions. These results corroborate our simulation findings, underscoring the efficacy of our sim-to-real transfer methodology. Notably, the data reveals that a calibrated injection of unbiased noise into the system can mitigate the adverse effects of bias, enhancing overall performance.

Through comprehensive analysis and rigorous testing, our research demonstrates the feasibility and effectiveness of applying simulation-trained policies to real-world UAV navigation tasks. This sim-to-real transition not only validates our control strategies but also provides valuable insights into the dynamics of UAV operation in complex, noise-influenced environments.

Policy Experimental Result in Real Environment				
μ	σ	Filter	Policy 1 (S/F)	Policy 2 (S/F)
0	0.1	None	2/3	5/0
0	0.1	LPF	5/0	5/0
0	0.1	KF	5/0	5/0
0.15	0	None	0/5	0/5
0.15	0.8	None	5/0	5/0
0.15	1.3	None	0/5	1/4

Table 5.1: **Real-World Evaluation Results**: The table presents the outcomes of conducting each experiment five times, indicating whether it was successful (S) or failure (F). Policy 2 demonstrates superior performance compared to Policy 1, which aligns with the findings from the simulation. Furthermore, the injection of a small amount of unbiased noise to bias enhances performance.



(a) UAV kept on the left, just before flight

(b) UAV is at the left side of the obstacle

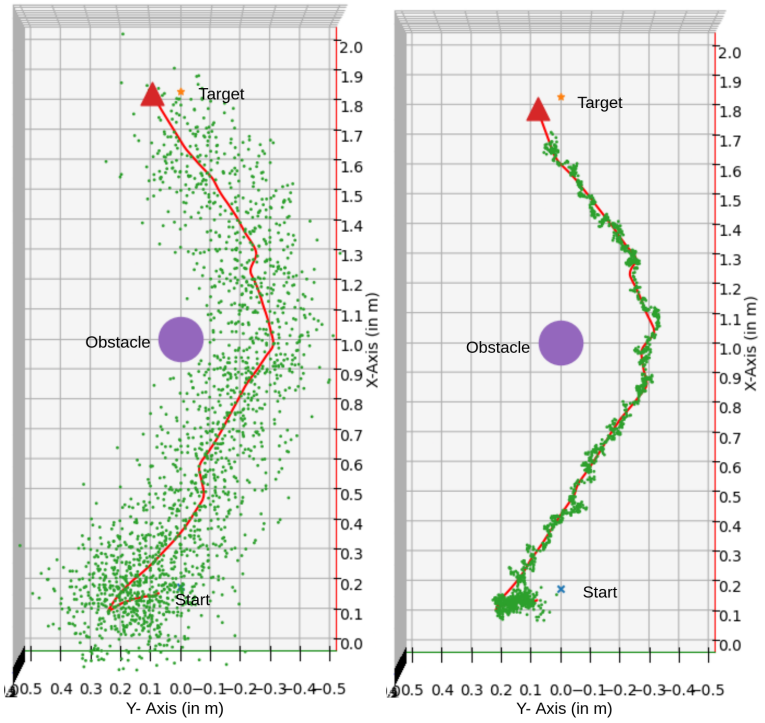
(c) UAV Successfully avoided the Obstacle

(d) UAV has reached the target location

Figure 5.8: **Real World Experiment Setup:** A sample zoom out version of the environment with instances of flight

The red line depicts the actual trajectory of the UAV, obtained via motion capture. Meanwhile, the green dots indicate the position estimate after undergoing perturbation and, if necessary, denoising. The initial deviation is attributed to the UAV's takeoff. Once the UAV ascends to the default altitude of 0.5 meters and stabilizes, control transitions to the trained policy.

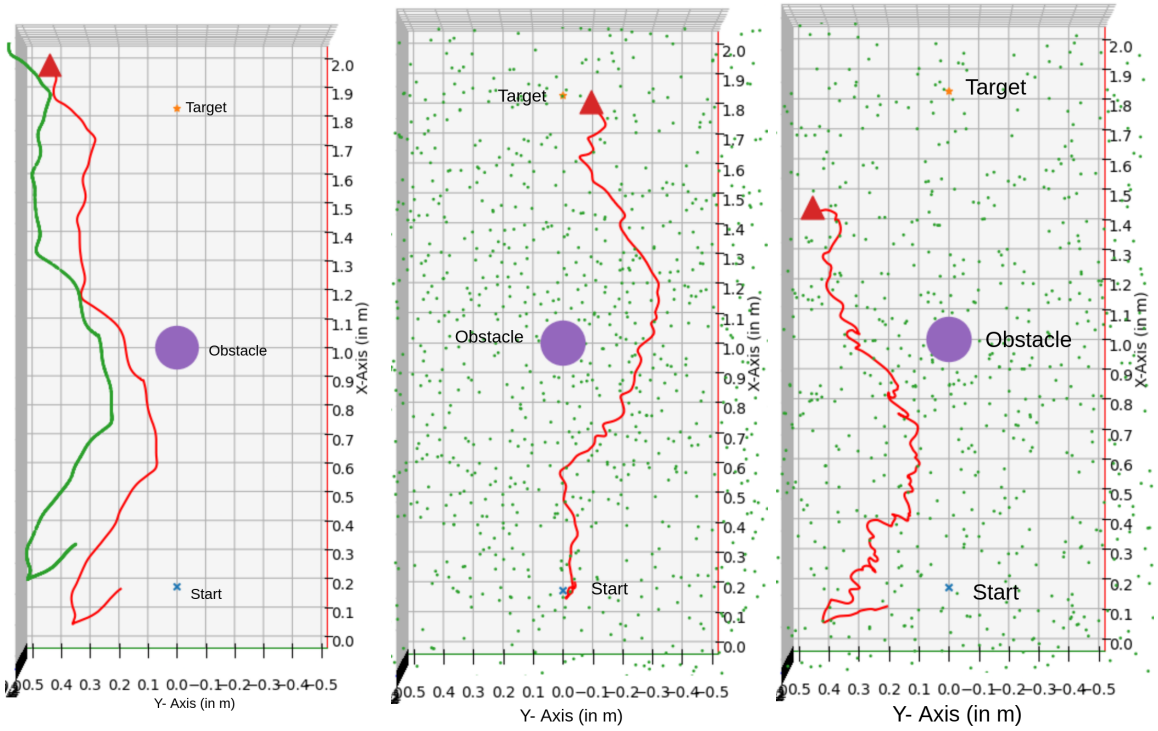
[0]The experimental video for sim to real could be found at https://youtu.be/ALTblQmQtHM



(a) $\mu = 0$
$\sigma = 0.1$, None

(b) $\mu = 0$
$\sigma = 0.1$, KF

Figure 5.9: **Real World Trajectories for Policy 2 (part 1):** Red line represents the true trajectory of the UAV collected through motion capture. The green dots represent the position estimate after being perturbed and passed through the denoiser if needed. The initial drift is due to the takeoff of the UAV. After the UAV reaches the default altitude (0.5 meters) and stabilizes, control is passed over to the trained policy. Fig (a) shows the trajectory in the presence of unbiased noise with $\sigma = 0.1$, and no denoiser. In Fig (b), the Kalman Filter is used to denoise the position estimate, as a result of which the green dots are much closer to the true trajectory.

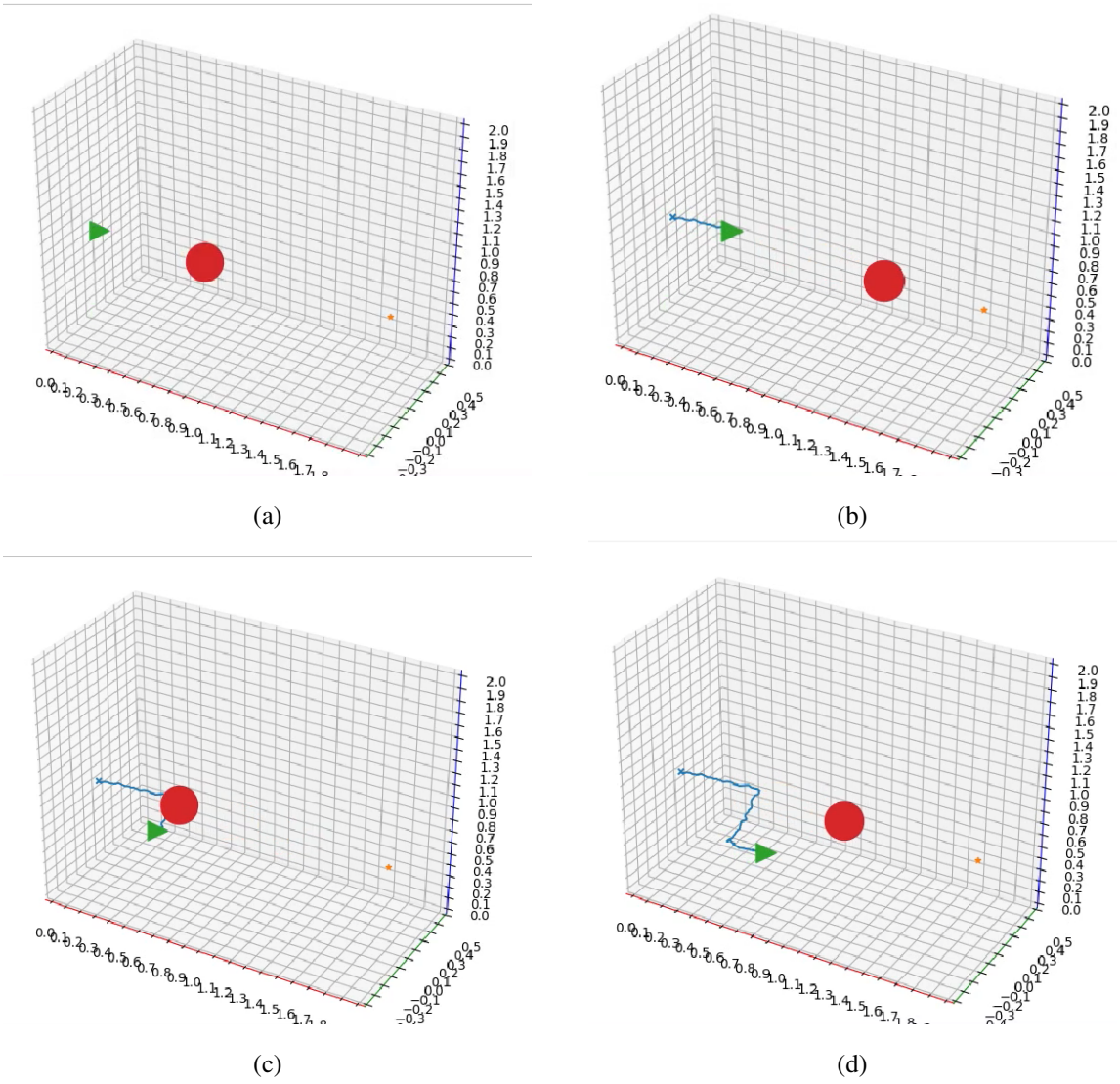


(a) $\mu = 0.15$
$\sigma = 0$, None

(b) $\mu = 0.15$
$\sigma = 0.8$, None

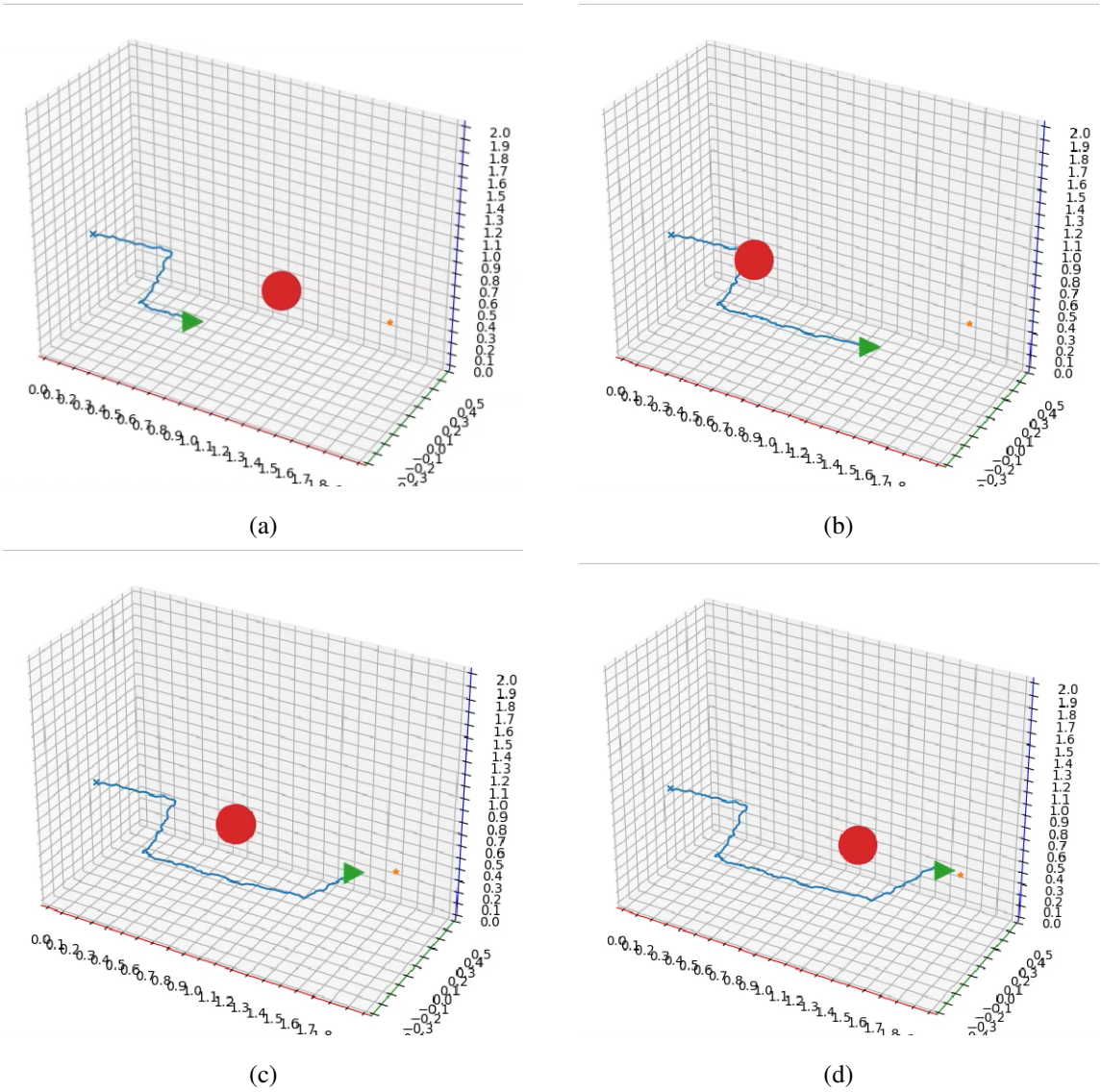
(c) $\mu = 0.15$
$\sigma = 1.3$, None

Figure 5.10: **Real World Trajectories for Policy 2 (part 2):** Figures (a), (b) and (c) correspond to evaluation with standard deviation at 0, 0.8 and 1.3 respectively, all with $\mu = 0.15$. Consistent with simulated results, bias-only (a) causes the policy to fail. A carefully selected value of σ causes improves performance (b) but choosing a value of σ that is too high causes failure.



(a) (b) (c) (d)

Figure 5.11: **Real World Experiment Setup for Dynamic Obstacles:** Four of eight images representing different stages of dynamic obstacle avoidance by drone.



(a) (b) (c) (d)

Figure 5.12: **Real World Experiment Setup for Dynamic Obstacles:** Next four of eight images representing different stages of dynamic obstacle avoidance by drone.

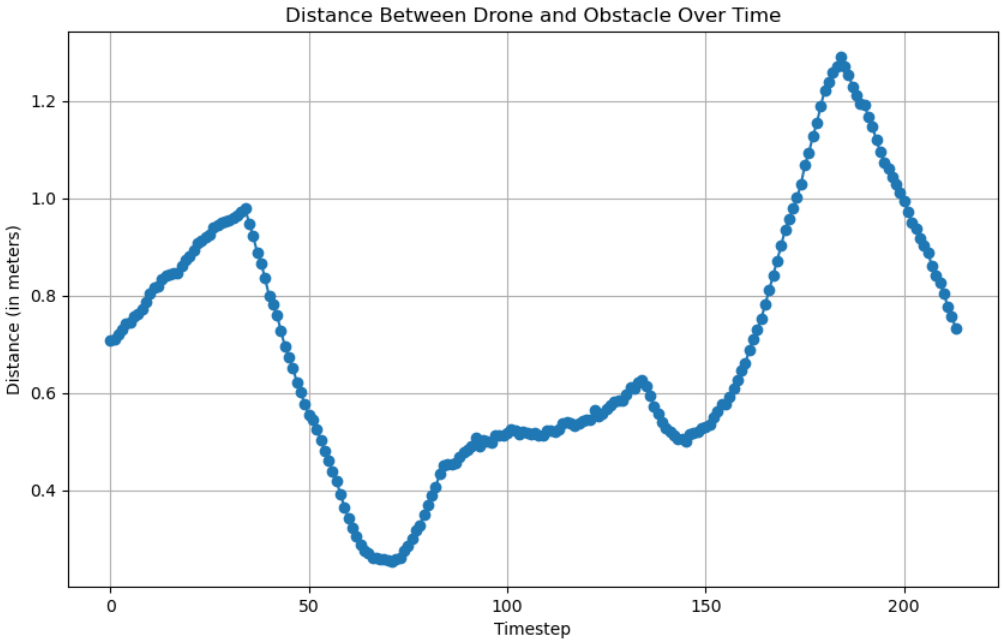


Figure 5.13: **Distance Plot** : The 2D plot provides a summary of the distance between the drone and the dynamic obstacle over time Fig. 5.11 and 5.12 above. The drone starts at a certain distance from the obstacle. It moves closer, showing some complex maneuvers to avoid the obstacle. After getting closer, the drone avoids the obstacle and moves away, which is reflected by the increasing distance in the final plot.

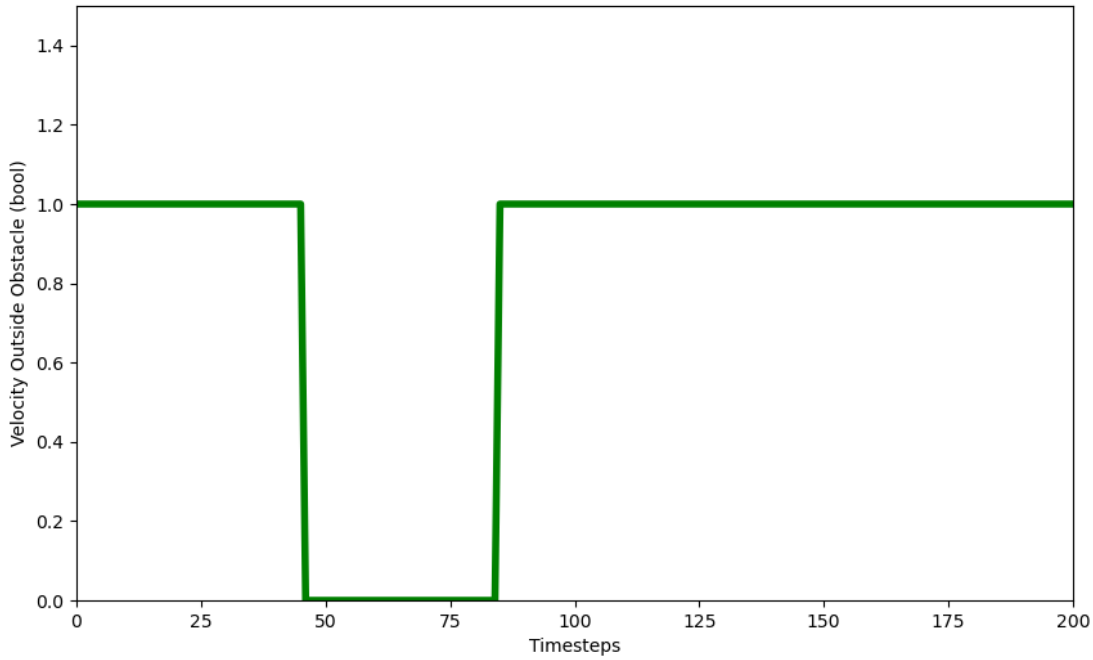


Figure 5.14: **VO Plot** : This is the VO plot for the above fig 5.13 The figure demonstrates when the UAV is avoiding obstacles by choosing a velocity outside the velocity obstacles.

Chapter 6

Conclusions

This research investigates the correlation between diverse Gaussian noise and a Proximal Policy Optimization agent. The agent is responsible for directing an unmanned aerial vehicle (UAV) to avoid obstacles in environments that are both static and dynamic, and have continuous state and action spaces. We have conducted experiments involving training policies with varying levels of unbiased noise in a dynamic environment and time-varying noise for the static environment. Furthermore, we assessed policy performance across different noise types—unbiased, bias-only, time-varying bias, and biased noise. To evaluate the effectiveness of denoising methods, we conducted experiments where we tested the trained policy in the presence of unbiased noise. We noticed improved performance when utilizing denoising techniques like the Low Pass Filter or the Kalman Filter, especially in situations with unbiased noise and time-varying biased noise. It's noteworthy that these filters did not yield any discernible benefits in cases of bias-only noise. The key results from our work are two-fold — first, training the PPO agent with a small amount of state space noise leads to it learning a very stable policy, outperforming a policy trained without noise across the board when evaluated in noisy environments. Second, and the more surprising result, is that we can leverage the inherent robustness of the trained policy to unbiased noise to improve its performance in environments with high bias low variance noise. This can be done by artificially injecting unbiased noise into the sensor measurements, yielding perturbed observations, which are then fed into the policy, greatly improving the success rate.

We tested addition of artificial noise for the static obstacles case in our work of Chapter 3, and have drawn the basis for dynamic obstacle environment in Chapter 4. And in Chapter 5, we showed the sim-to-real transfer with static obstacles in real-world setup. In our future work we look forward to testing on a real UAV in a dynamic environment, as well as extend the work to multi agent systems.

Related Publications

1. **Sim-to-Real Deep Reinforcement Learning based Obstacle Avoidance for UAVs Under Measurement Uncertainty**

   Bhaskar Joshi*, Dhruv Kapur*, Harikumar Kandath

   Accepted to IEEE International Conference on Automation, Robotics, and Applications (ICARA) 2024

2. **Deep RL based Obstacle Avoidance for UAVs under Measurement Uncertainty in Dynamic Environments**

   Aditya Kurande*, **Bhaskar Joshi***, Arjun Puthli and Harikumar Kandath

   In process of submission

[1]

[1] * These authors contributed equally to the work

Bibliography

[1] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[2] Hongming Zhang and Tianyang Yu. Taxonomy of reinforcement learning algorithms. *Deep Reinforcement Learning: Fundamentals, Research and Applications*, pages 125–133, 2020.

[3] Almira Budiyanto, Adha Cahyadi, Teguh Bharata Adji, and Oyas Wahyunggoro. Uav obstacle avoidance using potential field under dynamic environment. In *2015 International Conference on Control, Electronics, Renewable Energy and Communications (ICCEREC)*, pages 187–192. IEEE, 2015.

[4] Li-Yu Lo, Chi Hao Yiu, Yu Tang, An-Shik Yang, Boyang Li, and Chih-Yung Wen. Dynamic object tracking on autonomous uav system for surveillance applications. *Sensors*, 21(23):7888, 2021.

[5] Mehdi Maboudi, MohammadReza Homaei, Soohwan Song, Shirin Malihi, Mohammad Saadatseresht, and Markus Gerke. A review on viewpoints and path planning for uav-based 3d reconstruction. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 2023.

[6] Xiao Zhang and Lingjie Duan. Fast deployment of uav networks for optimal wireless coverage. *IEEE Transactions on Mobile Computing*, 18(3):588–601, 2018.

[7] Huang Yao, Rongjun Qin, and Xiaoyu Chen. Unmanned aerial vehicle for remote sensing applications—a review. *Remote Sensing*, 11(12):1443, 2019.

[8] Julie Linchant, Jonathan Lisein, Jean Semeki, Philippe Lejeune, and Cédric Vermeulen. Are unmanned aircraft systems (uas s) the future of wildlife monitoring? a review of accomplishments and challenges. *Mammal review*, 45(4):239–252, 2015.

[9] Mingyang Lyu, Yibo Zhao, Chao Huang, and Hailong Huang. Unmanned aerial vehicles for search and rescue: A survey. *Remote Sensing*, 15(13):3266, 2023.

[10] Young Kwan Ko, Ju Hyeong Park, and Young Dae Ko. A development of optimal algorithm for integrated operation of ugvs and uavs for goods delivery at tourist destinations. *Applied Sciences*, 12(20):10396, 2022.

[11] Ugochukwu O Matthew, Jazuli S Kazaure, Amaonwu Onyebuchi, Ogobuchi Okey Daniel, Ibrahim Hassan Muhammed, and Nwamaka U Okafor. Artificial intelligence autonomous unmanned aerial vehicle (uav) system for remote sensing in security surveillance. In *2020 IEEE 2nd International Conference on Cyberspac (CYBER NIGERIA)*, pages 1–10. IEEE, 2021.

[12] Dimosthenis C Tsouros, Stamatia Bibi, and Panagiotis G Sarigiannidis. A review on uav-based applications for precision agriculture. *Information*, 10(11):349, 2019.

[13] William W Greenwood, Jerome P Lynch, and Dimitrios Zekkos. Applications of uavs in civil infrastructure. *Journal of infrastructure systems*, 25(2):04019002, 2019.

[14] Chee Yong Tan, Sunan Huang, Kok Kiong Tan, and Rodney Swee Huat Teo. Three dimensional collision avoidance for multi unmanned aerial vehicles using velocity obstacle. *Journal of Intelligent & Robotic Systems*, 97(1):227–248, 2020.

[15] Jiayi Sun, Jun Tang, and Songyang Lao. Collision avoidance for cooperative uavs with optimized artificial potential field algorithm. *IEEE Access*, 5:18382–18390, 2017.

[16] MaÁ¯ ssa Boujelben, Chokri Rekik, and Nabil Derbel. A reactive approach for mobile robot navigation in static and dynamic environment using fuzzy logic control. *International Journal of Modelling, Identification and Control*, 27(4):293–302, 2017.

[17] Ferhat Uçan and D Turgay Altilar. Using genetic algorithms for navigation planning in dynamic environments. *Applied Computational Intelligence and Soft Computing*, 2012:18–18, 2012.

[18] Anne Steenbeek and Francesco Nex. Cnn-based dense monocular visual slam for real-time uav exploration in emergency conditions. *Drones*, 6(3):79, 2022.

[19] Ahmad Taher Azar, Anis Koubaa, Nada Ali Mohamed, Habiba A Ibrahim, Zahra Fathy Ibrahim, Muhammad Kazim, Adel Ammar, Bilel Benjdira, Alaa M Khamis, Ibrahim A Hameed, et al. Drone deep reinforcement learning: A review. *Electronics*, 10(9):999, 2021.

[20] Jens Kober, J Andrew Bagnell, and Jan Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32(11):1238–1274, 2013.

[21] Fadi AlMahamid and Katarina Grolinger. Autonomous unmanned aerial vehicle navigation using reinforcement learning: A systematic review. *Engineering Applications of Artificial Intelligence*, 115:105321, 2022.

[22] Huy X Pham, Hung M La, David Feil-Seifer, and Luan V Nguyen. Autonomous uav navigation using reinforcement learning. *arXiv preprint arXiv:1801.05086*, 2018.

[23] Sihem Ouahouah, Miloud Bagaa, Jonathan Prados-Garzon, and Tarik Taleb. Deep-reinforcement-learning-based collision avoidance in uav environment. *IEEE Internet of Things Journal*, 9(6):4015–4030, 2021.

[24] GUO Tong, Nan Jiang, LI Biyue, ZHU Xi, WANG Ya, and DU Wenbo. Uav navigation in high dynamic environments: A deep reinforcement learning approach. *Chinese Journal of Aeronautics*, 34(2):479–489, 2021.

[25] Chao Yan, Xiaojia Xiang, and Chang Wang. Towards real-time path planning through deep reinforcement learning for a uav in dynamic environments. *Journal of Intelligent & Robotic Systems*, 98:297–309, 2020.

[26] Yupeng Yang, Kai Zhang, Dahai Liu, and Houbing Song. Autonomous uav navigation in dynamic environments with double deep q-networks. In *2020 AIAA/IEEE 39th Digital Avionics Systems Conference (DASC)*, pages 1–7. IEEE, 2020.

[27] Alonica Villanueva and Arnel Fajardo. Deep reinforcement learning with noise injection for uav path planning. In *2019 IEEE 6th International Conference on Engineering Technologies and Applied Sciences (ICETAS)*, pages 1–6. IEEE, 2019.

[28] Chenyang Zhao, Olivier Sigaud, Freek Stulp, and Timothy M Hospedales. Investigating generalisation in continuous deep reinforcement learning. *arXiv preprint arXiv:1902.07015*, 2019.

[29] TQ Pham. Investigation of avoidance assistance system for the driver of a personal transporter using personal space: A simulation based study. *Int. J. Mech. Eng. Robot. Res*, 8(02):254–59, 2019.

[30] Warin Poomarin, Ratchatin Chancharoen, and Viboon Sangveraphunsiri. Automatic docking with obstacle avoidance of a differential wheel mobile robot. *International Journal of Mechanical Engineering and Robotics Research*, 5(1):11, 2016.

[31] Elder M Hemerly. Mems imu stochastic error modelling. *Systems Science & Control Engineering*, 5(1):1–8, 2017.

[32] Ganesan Balamurugan, J Valarmathi, and VPS Naidu. Survey on uav navigation in gps denied environments. In *2016 International conference on signal processing, communication, power and embedded system (SCOPES)*, pages 198–204. IEEE, 2016.

[33] Salah Sukkarieh, Eduardo Mario Nebot, and Hugh F Durrant-Whyte. A high integrity imu/gps navigation loop for autonomous land vehicle applications. *IEEE transactions on robotics and automation*, 15(3):572–578, 1999.

[34] Georg Halmetschlager-Funek, Markus Suchi, Martin Kampel, and Markus Vincze. An empirical evaluation of ten depth cameras: Bias, precision, lateral noise, different lighting conditions and materials, and multiple sensor setups in indoor environments. *IEEE Robotics & Automation Magazine*, 26(1):67–77, 2018.

[35] Beiya Yang, Erfu Yang, Leijian Yu, and Andrew Loeliger. High-precision uwb-based localisation for uav in extremely confined environments. *IEEE Sensors Journal*, 22(1):1020–1029, 2021.

[36] Sinan Gezici, Zhi Tian, Georgios B Giannakis, Hisashi Kobayashi, Andreas F Molisch, H Vincent Poor, and Zafer Sahinoglu. Localization via ultra-wideband radios: a look at positioning aspects for future sensor networks. *IEEE signal processing magazine*, 22(4):70–84, 2005.

[37] Beiya Yang and Erfu Yang. A survey on radio frequency based precise localisation technology for uav in gps-denied environment. *Journal of Intelligent & Robotic Systems*, 103(3):38, 2021.

[38] Jueming Hu, Xuxi Yang, Weichang Wang, Peng Wei, Lei Ying, and Yongming Liu. Obstacle avoidance for uas in continuous action space using deep reinforcement learning. *IEEE Access*, 10:90623–90634, 2022.

[39] Kaifang Wan, Xiaoguang Gao, Zijian Hu, and Gaofeng Wu. Robust motion control for uav in dynamic uncertain environments using deep reinforcement learning. *Remote sensing*, 12(4):640, 2020.

[40] Amy Zhang, Nicolas Ballas, and Joelle Pineau. A dissection of overfitting and generalization in continuous reinforcement learning. *arXiv preprint arXiv:1806.07937*, 2018.

[41] Chiyuan Zhang, Oriol Vinyals, Remi Munos, and Samy Bengio. A study on overfitting in deep reinforcement learning. *arXiv preprint arXiv:1804.06893*, 2018.

[42] Karl Cobbe, Oleg Klimov, Chris Hesse, Taehoon Kim, and John Schulman. Quantifying generalization in reinforcement learning. In *International conference on machine learning*, pages 1282–1289. PMLR, 2019.

[43] Mundla Narasimhappa, Arun D Mahindrakar, Vitor Campagnolo Guizilini, Marco Henrique Terra, and Samrat L Sabat. Mems-based imu drift minimization: Sage husa adaptive robust kalman filtering. *IEEE Sensors Journal*, 20(1):250–260, 2019.

[44] Dimitrios Tsiakmakis, Nikolaos Passalis, and Anastasios Tefas. Improving inertial-based uav localization using data-efficient deep reinforcement learning. In *2023 31st European Signal Processing Conference (EUSIPCO)*, pages 1355–1359. IEEE, 2023.

[45] Csaba Szepesvári. *Algorithms for reinforcement learning*. Springer Nature, 2022.

[46] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

[47] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*, 2017.

[48] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937. PMLR, 2016.

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

[50] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[51] Marc G Bellemare, Will Dabney, and Rémi Munos. A distributional perspective on reinforcement learning. In *International conference on machine learning*, pages 449–458. PMLR, 2017.

[52] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[53] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pages 1861–1870. PMLR, 2018.

[54] Boris T Polyak and Anatoli B Juditsky. Acceleration of stochastic approximation by averaging. *SIAM journal on control and optimization*, 30(4):838–855, 1992.

[55] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015.

[56] Frank Bowman. *Introduction to Bessel functions*. Courier Corporation, 2012.

[57] Rudolph Emil Kalman. A new approach to linear filtering and prediction problems. 1960.

[58] Greg Welch, Gary Bishop, et al. An introduction to the kalman filter. 1995.

[59] Paolo Fiorini and Zvi Shiller. Motion planning in dynamic environments using velocity obstacles. *The international journal of robotics research*, 17(7):760–772, 1998.

[60] Erwin Coumans and Yunfei Bai. Pybullet quickstart guide, 2021.

[61] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021.