AGILE FLIGHT IN DYNAMIC ENVIRONMENTS: BRIDGING REINFORCEMENT AND IMITATION LEARNING

DİNAMİK ORTAMLARDA ÇEVİK UÇUŞ: TAKLİT VE PEKİŞTİRMELİ ÖĞRENME İLE NAVİGASYON

AHMET GAZİ ÇİFCİ

ASST. PROF. ÖZGÜR ERKENT

Supervisor

Submitted to

Graduate School of Science and Engineering of Hacettepe University

as a Partial Fulfillment to the Requirements

for the Award of the Degree of Master of Science

in Computer Engineering

June 2024

ABSTRACT

AGILE FLIGHT IN DYNAMIC ENVIRONMENTS: BRIDGING REINFORCEMENT AND IMITATION LEARNING

Ahmet Gazi ÇİFCİ

Master of Science, Computer Engineering

Supervisor: Asst. Prof. Özgür ERKENT

June 2024, 67 pages

In recent years, the utilization of drones has seen a remarkable increase across various sectors, including surveillance, delivery services, and environmental monitoring. This surge is largely attributed to advancements in drone technology, making them more accessible and versatile. Among the capabilities that distinguish drones, agile flight emerges as a paramount feature, enabling drones to navigate complex environments with precision and efficiency. However, achieving agile flight in dynamic environments presents significant challenges, particularly in terms of rapid trajectory re-planning and computational demands.

This thesis proposes a novel approach to agile drone navigation by integrating Reinforcement Learning (RL) and Imitation Learning (IL). The methodology includes training a state-based teacher policy using the Proximal Policy Optimization (PPO) algorithm, which has access to comprehensive environmental information, including obstacle velocities. Subsequently, a student policy is trained through Behavioral Cloning (BC) to navigate without direct velocity information, relying instead on recurrent neural network architectures to infer this data.

Experimental results demonstrate that the proposed method significantly enhances the agility and efficiency of drones in dynamic environments. The combination of RL and

IL techniques not only reduces the computational burden but also shortens the training time, facilitating quicker adaptation and improved performance. The findings of this study contribute to advancing autonomous drone technology, offering a robust solution for navigating through cluttered and unpredictable environments. The project can be found in this link: https://github.com/Ag05ccc/agile_flight

Keywords: Agile Drone Flight, Reinforcement Learning, Dynamic Obstacle Avoidance, Imitation Learning, High-Speed Navigation

ÖZET

DİNAMİK ORTAMLARDA ÇEVİK UÇUŞ: TAKLİT VE PEKİŞTİRMELİ ÖĞRENME İLE NAVİGASYON

Ahmet Gazi ÇİFCİ

Yüksek Lisans, Bilgisayar Mühendisliği

Danışman: Asst. Prof. Özgür ERKENT

Haziran 2024, 67 sayfa

Son yıllarda, dronların kullanımı çeşitli sektörlerde, özellikle gözetim, teslimat hizmetleri ve çevre izleme alanlarında dikkate değer bir artış göstermiştir. Bu artış, drone teknolojisindeki ilerlemelere, dronların daha erişilebilir ve çok yönlü hale gelmesine büyük ölçüde atfedilmektedir. Dronları ayıran yetenekler arasında, çevik uçuş öne çıkarak dronların karmaşık ortamlarda hassasiyet ve verimlilikle gezinmelerini sağlamaktadır. Ancak, dinamik ortamlarda çevik uçuş elde etmek, özellikle hızlı rota yeniden planlama ve hesaplama gereksinimleri açısından önemli zorluklar sunmaktadır.

Bu tez, Takviyeli Öğrenme (RL) ve Taklit Öğrenme'yi (IL) entegre ederek çevik drone navigasyonu için yeni bir yaklaşım önermektedir. Metodoloji, engel hızları da dahil olmak üzere kapsamlı çevresel bilgilere erişimi olan Proximal Policy Optimization (PPO) algoritması kullanılarak duruma dayalı bir öğretmen politikası eğitmeyi içermektedir. Daha sonra, doğrudan hız bilgisi olmadan gezinecek olan öğrenci politikası, bu veriyi çıkarmak için tekrarlı sinir ağı mimarilerine dayanarak Davranış Klonlama (BC) yoluyla eğitilmektedir.

Deneysel sonuçlar, önerilen metodun dinamik ortamlarda dronların çevikliğini ve verimliliğini önemli ölçüde artırdığını göstermektedir. RL ve IL tekniklerinin kombinasyonu, hesaplama yükünü azaltmanın yanı sıra eğitim süresini kısaltarak daha hızlı uyum ve gelişmiş performans sağlamaktadır. Bu çalışmanın bulguları, dağınık ve öngörülemeyen ortamlarda gezinmek için sağlam bir çözüm sunarak otonom drone teknolojisinin ilerlemesine katkıda bulunmaktadır. Project bu linkte bulunabilir: https://github.com/Ag05ccc/agile_flight

Keywords: Çevik Drone Uçuşu, Takviyeli Öğrenme, Dinamik Engel Kaçınma, Taklit Öğrenme, Yüksek Hızlı Navigasyon

CONTENTS

Page

TABLES

FIGURES

ABBREVIATIONS

RL	:	**R**einforcement **L**earning
IRL	:	**I**nverse **R**einforcement **L**earning
DRL	:	**D**eep **R**einforcement **L**earning
LSTM	:	**L**ong **S**hort **T**erm **M**emory
PPO	:	**P**roximal **P**olicy **O**ptimization
TRPO	:	**T**rust **R**egion **P**olicy **O**ptimization
A2C	:	**A**dvantage **A**ctor **C**ritic
MLP	:	**M**ultilayer **P**erceptron
IMU	:	**I**nertial **M**easurement **U**nit
MDP	:	**M**arkov **D**ecision **P**rocess
IL	:	**I**mitation **L**earning **U**nit
BC	:	**B**ehavioral **C**loning **U**nit
UAV	:	**U**nmanned **A**erial **V**ehicle

1. INTRODUCTION

In recent years, the utilization of drones has seen a remarkable increase across various sectors, including surveillance, delivery services, and environmental monitoring [4]. This surge is largely attributed to the advancements in drone technology, making them more accessible and versatile. Among the capabilities that distinguish drones, agile flight emerges as a paramount feature, enabling drones to navigate complex environments with precision and efficiency. However, the pursuit of agile flight faces significant challenges, especially when confronted with dynamic obstacles. The essence of agility lies in the drone's ability to swiftly update its flight path in response to unexpected obstacles. This rapid trajectory re-planning is inherently computationally intensive, posing a dilemma; while agility demands fast reactions to avoid collisions, the computational burden can hinder the drone's ability to respond swiftly.

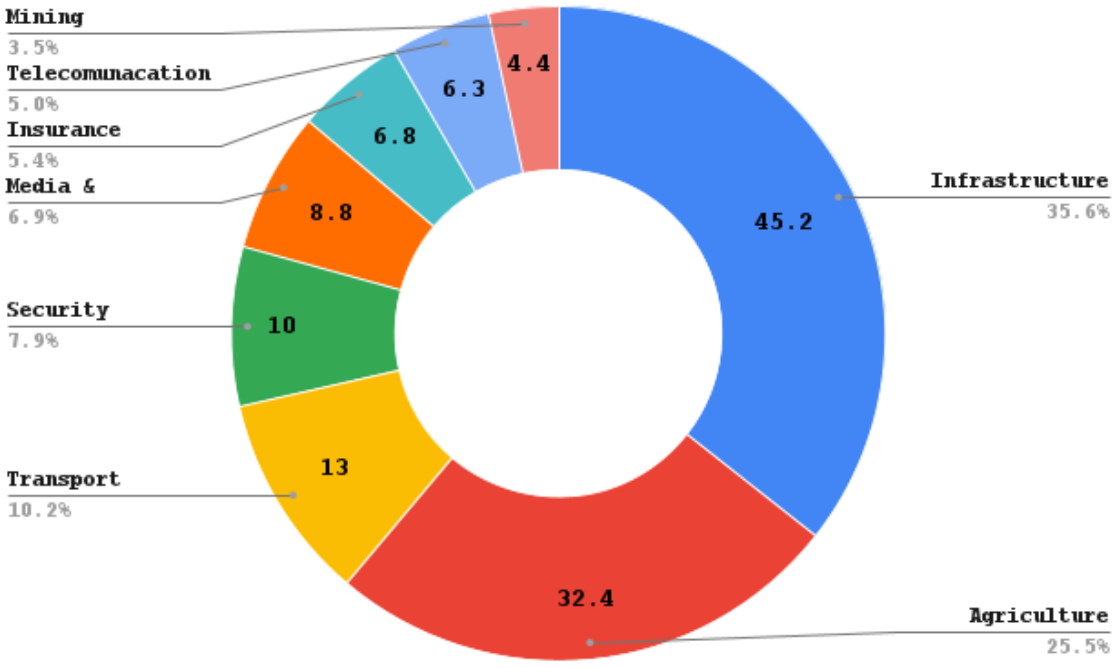

Figure 1.1 Predicted value of UAV solutions in key industries (billion) [4]. Courtesy of Shakhatreh et al.

RL is celebrated for its rapid processing capabilities, adeptness at solving intricate control challenges, and proficiency in optimizing parameters tailored to specific problems [10–12]. This adaptability makes it an ideal candidate for addressing the high-stakes demands of agile drone flight, where precise and real-time decision-making is crucial for successful navigation. However, a notable limitation of RL is its dependency on extensive trial-and-error processes during the learning phase. This characteristic, while fundamental to its learning mechanism, can significantly prolong the training period and increase computational resources [2].

Imitation Learning (IL) has emerged as a promising technique to address these challenges, especially in the context of drones. IL leverages expert demonstrations to train models. Various studies have shown that IL can significantly accelerate the learning process for drones by providing a clear baseline from which the learning algorithm can start [2, 13, 14]. For instance, research has demonstrated the efficacy of IL in training drones to perform complex maneuvers, such as obstacle avoidance by imitating expert [15–17]. These demonstrations offer valuable insights into successful strategies, which the learning agent can then refine and optimize. By integrating IL, researchers have been able to develop drone control systems that learn more efficiently and effectively, navigating through complex environments with reduced computational overhead [18]. These studies generally focus on static environments and topics such as drone racing. In contrast, we focus on achieving agile flight in an environment with dynamic objects and use imitation learning (IL) methods for this problem. Additionally, unlike others [16], we use a reinforcement learning (RL) model as the expert policy instead of an optimization-based expert controller.

To specifically address the challenge of navigating around dynamic obstacles, our approach involves using Position History information within an LSTM-based Recurrent Policy. This model aims to enable the drone to understand the movement direction of obstacles, enhancing its ability to avoid collisions. By incorporating an RNN structure, our model can behave as a Bayes estimator, allowing it to include more states and estimate the next state with superior accuracy compared to models without RNN structure. However, training such a complex model can be excessively time-consuming due to the computational demands of processing

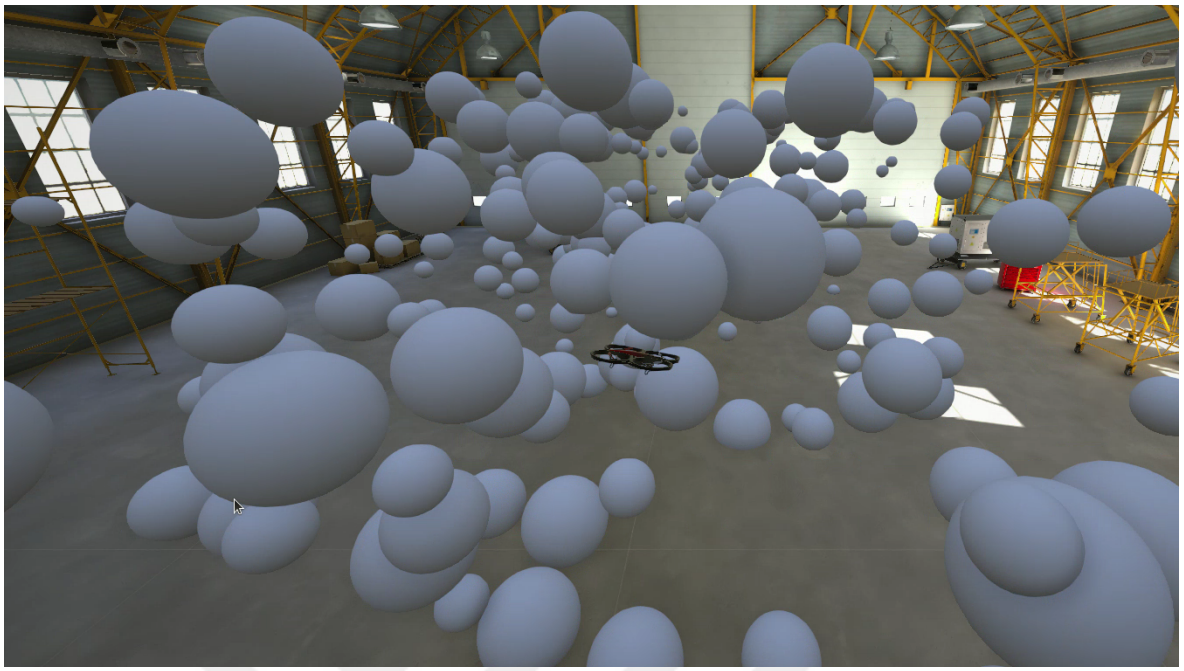

Figure 1.2 Environment With Dynamic Obstacles.

the temporal dynamics of obstacle movements. To mitigate this issue, we employed Imitation Learning (IL), which significantly expedited the learning process by leveraging pre-existing expert knowledge, thereby reducing the extensive trial-and-error phase typically associated with traditional Reinforcement Learning (RL).

To our knowledge, this is the first instance of using LSTM in imitation learning with such small training times. This novel application demonstrates the efficiency of our approach in rapidly training models capable of navigating dynamic environments.

In our study, a student policy learns by observing demonstrations from a teacher policy that's been trained with comprehensive state information, including drone and obstacle dynamics. This process simplifies learning because the student imitates successful strategies without undergoing the tedious process of discovering them independently. A crucial point is that our teacher policy utilizes obstacle velocity information to navigate effectively—a detail not available to the student. This means the student learns to navigate effectively without direct access to obstacle velocity information but by understanding their positioning and

movement patterns, making the learning process both efficient and thereby shortening the model's training time.

In conclusion, this thesis presents a novel approach to drone navigation through cluttered environments, leveraging the strengths of Reinforcement Learning (RL) and Imitation Learning (IL) to overcome the inherent limitations of each. By combining RL's capability for rapid, complex problem-solving with IL's efficiency in learning from expert demonstrations, we develop a robust navigation system that can adapt to dynamic obstacles with unprecedented agility. The key innovation lies in training a student policy to navigate without direct obstacle velocity information, a challenge that pushes the boundaries of current drone technology.

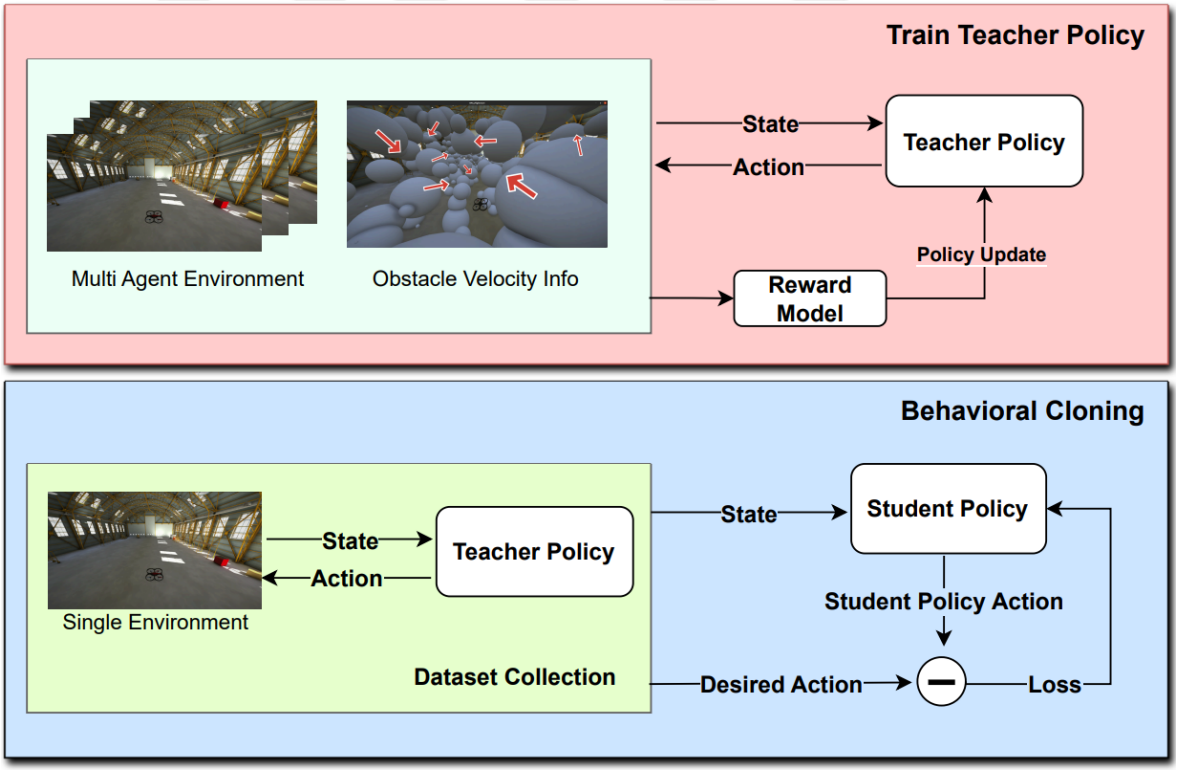

Figure 1.3 Training pipeline for drone policy: The upper section shows the training of the teacher policy in a multi-agent environment with dynamic obstacle velocity information, guided by a reward model. The lower section illustrates the behavioral cloning process where the student policy learns from the dataset collected by the teacher policy in an environment without obstacle velocity information.

1.1. Scope Of The Thesis

The scope of this thesis is strategically concentrated on three pivotal areas that are essential for advancing drone navigation in complex environments. These areas are crucial for enhancing the agility and efficiency of drones, especially in scenarios laden with dynamic obstacles. The focused areas within this thesis include:

- **Development of a Hybrid Learning Framework:** This framework is designed to leverage the rapid problem-solving capabilities of RL and the efficient learning mechanisms of IL, aiming to facilitate a more effective and expedient training process for drone navigation systems. The integration of these methodologies seeks to address the limitations posed by traditional RL techniques, particularly their extensive reliance on trial and error.
- **Dynamic Obstacle Navigation:** This thesis focuses on improving the ability of drones to navigate through environments that contain moving obstacles. This challenge is critical for real-world applications where the environment is not static, and obstacles can change position or appear unexpectedly. The research focuses on developing algorithms that enable drones to anticipate and adapt to these dynamic elements swiftly, ensuring safe passage and maintaining optimal flight paths even in unpredictable settings.
- **Computational Efficiency:** A key objective of this thesis is to improve the computational efficiency of drone navigation systems. This entails innovating strategies to reduce the computational burden associated with real-time trajectory planning and obstacle avoidance. By streamlining these processes, the aim is to enable drones to make quicker decisions without compromising accuracy or safety. This aspect is particularly crucial for agile flight, where the ability to respond rapidly to environmental changes can significantly impact the drone's performance and reliability.

1.2. Contributions

This thesis explores the critical challenges of autonomous drone navigation, particularly focusing on the complexities of dynamic obstacle avoidance and improving learning algorithms for more efficient drone operation. The cornerstone of this work lies in its novel approach to combining Reinforcement Learning (RL) and Imitation Learning (IL) for superior navigation performance. Here, we briefly outline the key contributions of this thesis

- **Advancement in Dynamic Obstacle Navigation for Agile Flights**: Utilizing recurrent neural networks, the drone's ability to respond to moving obstacles in real-time is enhanced, significantly improving navigation accuracy and safety in dynamic environments.
- **Efficient Learning Mechanism With Direct Obstacle Velocity Information**: Incorporating direct velocity information of obstacles into the learning process reduces the computational burden and enhances the efficiency of the learning algorithms, leading to quicker adaptation and more reliable obstacle avoidance.
- **Integration of Reinforcement Learning and Imitation Learning**: A synergistic integration of RL and IL is introduced, where RL refines the navigation policies initially learned through IL, resulting in a robust and adaptive navigation system capable of handling complex and unforeseen scenarios.

Each of these contributions not only pushes the boundaries of current drone navigation capabilities but also opens new avenues for future research and development in autonomous system technologies.

The project code can be found in this link: https://github.com/Ag05ccc/agile_flight

2. BACKGROUND OVERVIEW

The purpose of this section is to provide a comprehensive understanding of the foundational concepts and related work that inform the methodologies employed in this thesis, which focuses on agile flight in dynamic environments using reinforcement and imitation learning. The topics covered in this section are presented in a logical order to build a cohesive background, starting with fundamental theories and progressing towards more specific techniques and applications relevant to our study on drone navigation through dynamic obstacles. This section covers Markov Decision Processes (MDP), which model decision-making in uncertain environments, and Reinforcement Learning (RL), which enables an agent to learn optimal policies through interactions. The Proximal Policy Optimization (PPO) Algorithm is highlighted as a key RL method for its balance between exploration and exploitation. Additionally, Imitation Learning is introduced, focusing on Behavioral Cloning, which trains policies by mimicking expert actions, relevant for transferring knowledge from a teacher policy to a student policy.

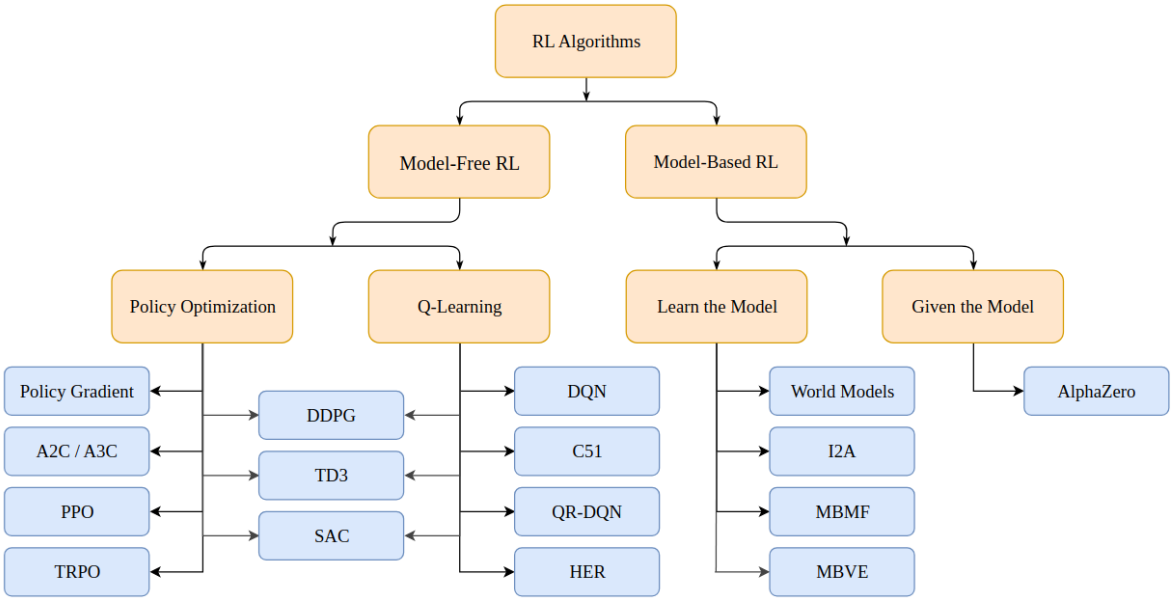

Figure 2.1 A Classification of Reinforcement Learning Algorithms [5].

2.1. Markov Decision Processes (MDP)

In environments where outcomes are influenced by both randomness and the choices of a decision maker, the concept of Markov Decision Processes (MDPs) is applied. These processes help in understanding and structuring the decision-making dynamics in such settings. MDPs are characterized by states, actions, transition probabilities, and rewards. States represent different scenarios or configurations in which the system can exist, while actions are the choices available that can change the state. Transition probabilities quantify the likelihood of moving from one state to another given a specific action, reflecting the inherent uncertainties of the environment. Rewards provide feedback based on the actions taken, guiding the decision maker toward desirable outcomes.

The essence of an MDP is its Markov property, which asserts that the future state is determined solely by the current state and action, without regard to the sequence of preceding events. This property simplifies the decision-making process by focusing only on the present state, making past states irrelevant for future decisions.

The Bellman equation is a crucial recursive formula in both dynamic programming and reinforcement learning, used to find the optimal policy in a Markov Decision Process (MDP). It expresses the value of a state as the expected return of the best action taken from that state, considering both immediate rewards and future rewards.

The Bellman equation for the value function $V(s)$ is given by:

$$V(s) = \max_a \sum_{s'} P(s'|s,a) \left[R(s,a,s') + \gamma V(s') \right] \tag{1}$$

where:

- $V(s)$ is the state value,
- $\max_a$ indicates the maximum value over all possible actions a,

- $P(s'|s, a)$ represents the likelihood of transitioning to state s' from state s given action a,
- $R(s, a, s')$ is the reward obtained after moving from state s to state s' as a result of action a,
- γ is the discount factor, which determines the importance of future rewards.

The objective in an MDP is to determine a policy, which is a strategy that specifies the best action to take in each state to maximize the expected sum of future rewards, often considering a discount factor for future rewards to ensure the sum is finite. Solving an MDP involves finding this optimal policy using algorithms such as dynamic programming. MDPs are widely applicable in fields like robotics, economics, and artificial intelligence, where they help in developing strategies for systems that must operate in complex, uncertain environments.

MDPs have certain shortcomings, especially when applied to drone navigation. The state and action spaces can become extremely large, making the computation of optimal policies challenging. Additionally, the dynamic and unpredictable nature of real-world environments can lead to inaccuracies in the transition probabilities and rewards.

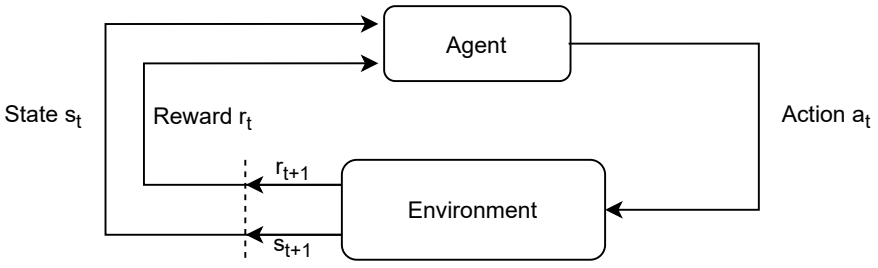

Figure 2.2 Typical framing of the RL scenario. Observing the environment through the state space, the agent uses the reward value as a feedback mechanism. Courtesy of Sutton and Barto [6]

2.2. Reinforcement Learning (RL)

Reinforcement Learning (RL) is a subset of machine learning that emphasizes how agents should make decisions within an environment to maximize their cumulative reward. Unlike

other types of learning, reinforcement learning is particularly concerned with making decisions sequentially and learning from the outcomes of these decisions. This field of study bridges the gap between decision-making and optimal control, using concepts from both artificial intelligence and psychology to develop algorithms that improve through trial and error. [19]

The core of reinforcement learning revolves around the agent, representing the decision-maker, and the environment, representing the world in which the agent operates. The interaction between the agent and the environment occurs at discrete time steps. At each time step, the agent receives the current state of the environment and selects an action in response. The environment, in turn, presents a new state and a reward to the agent. The reward, expressed as a numerical value, indicates the benefit of an action, and the agent's goal is to maximize the total accumulated reward over time. Reinforcement Learning differs from Markov Decision Processes (MDPs) in that MDPs provide a formal framework for modeling decision-making problems, but do not inherently involve learning. RL can be seen as a subset of MDPs, focusing on solving MDPs through learning algorithms that improve from interaction with the environment.

This process involves learning a policy — a mapping from states of the environment to the actions the agent should choose when in those states. The challenge in RL is that the agent must learn the best policy through its own experience, without prior knowledge of how the environment behaves. This is similar to learning to play a new game by playing it repeatedly and observing what actions lead to winning. For example, the AlphaGo system used RL to learn and master the game of Go, achieving superhuman performance [20].

Reinforcement learning is notably different from other types of learning in that it does not require labeled input/output pairs, as is typical in supervised learning, nor does it seek to find patterns in data, as in unsupervised learning. Instead, RL is about finding a balance between exploration (trying unfamiliar actions to discover their effects) and exploitation (taking actions known to yield high rewards, albeit with a risk).

A fundamental aspect of reinforcement learning is the trade-off between immediate and long-term rewards. This is often handled using a discount factor, a parameter less than one that reduces the value of future rewards. This concept is crucial as it helps in formulating a solution that balances the immediate payoff with delayed benefits, which is often essential in dynamic environments where the circumstances may change unpredictably.

Reinforcement learning has a broad range of applications, from robotics, where it can be used to teach machines to perform complex tasks that cannot be explicitly programmed, to game playing and autonomous vehicles [21, 22]. In robotics, RL has been used to train robots for tasks such as manipulation and locomotion. In autonomous vehicles, RL helps in decision-making processes like lane changing and collision avoidance[23]. For instance, RL algorithms have enabled drones to navigate through dynamic environments like autonomous landing on a moving platform [24].

Overall, reinforcement learning represents a sophisticated area of study in machine learning, focusing on the development of algorithms that improve automatically through experiences. It holds the promise of enabling sophisticated levels of artificial intelligence in a variety of fields.

Reinforcement learning can be broadly categorized into model-based and model-free methods. Model-based methods involve creating a model of the environment to plan actions by predicting future states and rewards. In contrast, model-free methods learn the optimal policy directly from interactions with the environment without explicitly modeling it. Proximal Policy Optimization (PPO), a model-free method, is used in our project due to its efficiency and stability in learning policies directly from experiences.

2.2.1. Proximal Policy Optimization (PPO) Algorithm

Proximal Policy Optimization (PPO) [25] and other model-free reinforcement learning methods are crucial in the field of policy optimization. Some of the fundamental model-free methods include Policy Gradient, Asynchronous Advantage Actor-Critic (A2C/A3C)[26],

Trust Region Policy Optimization (TRPO)[27], and PPO. Policy Gradient methods optimize the policy directly by estimating the gradient of the expected return with respect to the policy parameters, iteratively improving the policy. They are simple and applicable to both discrete and continuous action spaces but can suffer from high variance and instability in learning. A2C/A3C methods improve on basic Policy Gradient by using multiple agents to explore the environment in parallel. This parallelism reduces variance and accelerates learning. However, A3C's reliance on on-policy updates can still lead to instability and slower convergence. TRPO addresses the stability issues of Policy Gradient and A3C by ensuring that updates to the policy do not deviate too much from the previous policy, thus maintaining a trust region. This method is more stable but computationally complex due to the second-order optimization involved. PPO builds on the strengths of TRPO while simplifying the optimization process. It uses a novel objective function to penalize large policy updates, effectively maintaining a trust region without the computational overhead of TRPO. This balance makes PPO both stable and computationally efficient, making it suitable for a wide range of reinforcement learning applications.

Policy gradient methods optimize the policy directly by estimating the gradient of the expected return (a measure of how well the policy performs) with respect to the policy parameters. By adjusting the parameters in the direction of the gradient, the policy is iteratively improved. These methods are particularly appealing because they are applicable to both discrete and continuous action spaces, and they can naturally incorporate stochastic policies.

Traditional policy gradient methods, such as Trust Region Policy Optimization (TRPO) [27], aim to take the largest possible improvement step on a policy without stepping too far and causing performance collapse. TRPO, in particular, uses a complex second-order optimization method to maintain a balance between improvement and stability. However, TRPO's computational complexity and implementation difficulty limit its practicality, especially for large-scale problems.

PPO aims to retain the benefits of TRPO but in a simpler, more efficient manner. It introduces

a novel objective function that penalizes changes to the policy that move it too far from the old policy, effectively maintaining a trust region around the old policy. There are two primary variants of PPO, differentiated by their objective function. The first variant, PPO-Clip, uses a clipped surrogate objective. The idea is to clip the probability ratio between the new policy and the old policy to be within a certain range (typically between $1-\epsilon$ and $1+\epsilon$, where ϵ is a small positive number). This clipping prevents the new policy from deviating too much from the old policy, thus ensuring stability and reliable updates. The second variant, PPO-Penalty, adds a penalty term to the objective function that directly penalizes large deviations from the old policy. The penalty is often based on the KL divergence between the old and new policies, ensuring that the updates remain within a trust region. Both variants aim to strike a balance between exploration and stability, preventing the policy from changing too drastically in a single update, thereby retaining the advantages of trust region methods like TRPO (Trust Region Policy Optimization) but in a more computationally efficient way.

Proximal Policy Optimization represents a significant advancement in the field of reinforcement learning, particularly in policy gradient methods. By addressing the stability and efficiency challenges of its predecessors, PPO has paved the way for more effective and practical reinforcement learning solutions. Its balance between performance and computational simplicity makes it a valuable tool for researchers and practitioners alike in the ongoing exploration and application of reinforcement learning techniques.

For example, it has been successfully used in robotics for tasks such as robotic manipulation and locomotion [21]. In the domain of game playing, PPO has been used in training AI agents for complex strategy games like Dota 2 and StarCraft II [28, 29]. Additionally, PPO is used in autonomous driving systems to optimize decision-making processes for tasks like lane keeping and obstacle avoidance, [22, 30, 31].

PPO uses a clipped objective function to prevent large updates in the policy. This function is defined as:

$$L^{\text{CLIP}}(\theta) = \mathbb{E}_t \left[\min \left(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t \right) \right]$$

where,

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$$

is the probability ratio, $\hat{A}_t$ is the advantage function, and ϵ is the clip range.

The total loss function used in PPO includes the clipped objective function, value function loss, and an entropy term for policy diversity:

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

where,

$$L^{\text{VF}}(\theta) = \left(V(s_t) - V_{\text{target}}(s_t) \right)^2$$

is the value function loss, $S[\pi_\theta](s_t)$ is the policy entropy term, and c_1 and c_2 are hyperparameters.

The advantage function indicates how good an action is in a particular state and is often calculated using Generalized Advantage Estimation (GAE):

$$\hat{A}_t = \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_{t+l}$$

where,

$$\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t)$$

is the temporal difference error, γ is the discount factor, and λ is the GAE parameter.

2.3. Imitation Learning

Imitation Learning, in the context of artificial intelligence and machine learning, is a method by which agents learn to perform tasks by observing and imitating the actions of experts

[32]. This learning paradigm is rooted in the idea that complex behaviors can be acquired more efficiently by leveraging the knowledge and strategies of those already proficient, rather than learning from scratch through trial-and-error [2]. Imitation Learning is particularly useful in scenarios where direct programming of desired behaviors is impractical due to the complexity of the task or the environment.

There are several approaches within Imitation Learning, including Behavior Cloning and Inverse Reinforcement Learning [33, 34]. Behavior Cloning, involves direct mapping from observed states to actions. The agent learns a policy that replicates the expert's actions in similar situations by training on a dataset of state-action pairs collected from expert demonstrations. This method is similar to supervised learning and is straightforward to implement. However, it may suffer from issues such as covariate shift, where the distribution of states encountered by the agent differs from those in the training dataset, leading to poor generalization.

Inverse Reinforcement Learning (IRL) focuses on learning the underlying reward function that the expert is assumed to be optimizing, based on their demonstrated behavior [34]. Once the reward function is inferred, various reinforcement learning techniques can be applied to learn optimal policies. This approach can potentially lead to more robust and generalizable policies compared to Behavior Cloning, as the agent learns the "why" behind the actions rather than merely mimicking them.

Imitation Learning has applications across various fields, including robotics (where robots learn tasks by observing human demonstrations)[35], autonomous vehicles (learning to drive by mimicking human drivers)[36], and game playing (learning strategies by observing expert players) [37].

One of the main advantages of Imitation Learning is its ability to significantly reduce the amount of exploration needed to learn a task, as the agent is guided by expert demonstrations. However, challenges such as ensuring the quality of the expert demonstrations, dealing with environments where expert knowledge is not available or incomplete, and addressing the

discrepancy between the agent's and the expert's state representations (observability) remain active areas of research.

2.3.1. Behavioral Cloning

Behavioral cloning is a type of machine learning that falls under the broader category of imitation learning, where the goal is to mimic human behavior in a specific task [33]. It involves training an algorithm, typically a neural network, to replicate the actions of an expert by observing examples of their behavior. This method is commonly used in domains where defining explicit rules for performing a task is difficult, but where demonstration data is readily available.

The process of behavioral cloning can be broken down into several steps. First, the expert's behavior is captured in the form of data, which typically includes observations of the environment (input) and the corresponding actions taken by the expert (output).

Once the data is collected, it is used to train a model. This training process involves adjusting the parameters of the model so that it predicts the expert's actions as accurately as possible, given the observed environmental states. The model essentially learns a function that maps from the state of the environment to the actions that should be taken in those states.

For example, in the context of autonomous flight for drones, this data might consist of sensor readings (like altitude, speed, and obstacles detected through lidar or cameras) as the input, and the control inputs from a skilled pilot (such as adjustments to throttle, pitch, and yaw) as the output. The trained model then aims to replicate these pilot actions based on similar sensor readings, effectively learning to fly the drone by mimicking the human expert's control maneuvers [7].

Figure 2.3 Example of behavioral cloning drone flight experiment. Courtesy of Rodríguez-Hernandez et al. [7]

Behavioral cloning directly maps observed states to actions, thus bypassing the need to understand the underlying motivations or strategies behind the expert's decisions. This makes the approach relatively simple to implement, as it does not require the model to understand or model the environment's dynamics or the reward structure that the expert might be optimizing.

However, there are several challenges associated with behavioral cloning. One significant issue is the problem of distribution shift, where the trained model may perform well on scenarios similar to those seen during training but fails to generalize to new, unseen situations. This is because the model has only learned to replicate the observed actions and lacks a deeper understanding of how those actions affect the environment.

Another challenge is the need for a large amount of high-quality, diverse training data. Since the model learns only from the data it has seen, its performance is highly dependent on the variety and representativeness of the training examples. If the training data is not diverse or is biased in some way, the model may not perform well in real-world conditions.

Despite these challenges, behavioral cloning is widely used due to its simplicity and effectiveness in many applications. It has been particularly impactful in robotics and autonomous systems, where it has enabled machines to perform complex tasks such as flying drones, driving cars, and performing surgery, tasks that would be difficult to program explicitly. Overall, behavioral cloning represents a practical approach to machine learning when direct teaching through demonstration is feasible, providing a straightforward path to developing capable systems in various domains.

In this thesis, a teacher policy was trained using the RL-based PPO algorithm, which utilized comprehensive environmental information. The teacher policy, with access to detailed data such as obstacle velocities, achieved efficient learning. Subsequently, Behavioral Cloning was employed to train the student policy. This approach allowed the student policy to successfully navigate dynamic obstacles without collision, even in the absence of specific speed information, by mimicking the behaviors demonstrated by the teacher policy.

3. RELATED WORK

Recent advancements in the domain of autonomous drone navigation have highlighted the efficacy of Deep Reinforcement Learning (DRL) methods over traditional optimization-based approaches. Studies such as [38, 39] have demonstrated that neural control policies can significantly outperform conventional model-based planning and control methods. Specifically, these DRL approaches have been successful in navigating quadrotors through cluttered environments swiftly and with greater accuracy, achieving minimal travel time [40].

Agility in drone systems, especially in obstacle-dense environments, hinges critically on the capability to detect obstacles early and react promptly. To address this, several perception-aware methodologies leveraging Reinforcement Learning have been developed [2, 13]. These methods integrate sensory processing with control mechanisms, enhancing the drones' ability to respond to environmental challenges more effectively.

However, a significant challenge with RL-based approaches is their dependency on extensive trial-and-error data sets, which can be impractical in real-world scenarios. To mitigate this, recent research has explored the use of Imitation Learning to bootstrap RL solutions using existing datasets generated from classical approaches [2, 14, 16]. This synergy facilitates the development of robust models by significantly reducing the need for new data collection. Furthermore, some studies have adopted an RL-based model as a 'Teacher policy' to refine and surpass the limitations of classical optimization-based policies, effectively enhancing overall system performance [2, 41].

These studies have generally been conducted in environments with racing drones or static obstacles [2, 12]. Due to their focus on racing drones, they tend to perform well in the environments in which they were trained but show limited generalizability to other environments beyond their original training. In our work, we have applied these successful methods to environments with a high density of dynamic obstacles. By leveraging these innovative approaches, drone systems not only improve in agility but also in operational

safety and efficiency, promising substantial advancements in the field of autonomous aerial navigation.

While the previous studies predominantly focus on static environments and applications such as drone racing, our research shifts the focus to achieving agile flight in environments populated with dynamic objects. To tackle the complexities of these dynamic scenarios, we implement imitation learning (IL) techniques. Unlike the methods that depend on an optimization-based expert controller [16], our approach leverages a reinforcement learning (RL) model as the expert policy. This choice enhances our system's ability to adapt to novel and unforeseen situations, thereby improving agility and robustness. Our goal is not only to achieve high performance within the training environments but also to ensure effective generalization to new and dynamic contexts, thereby expanding the practical applicability and reliability.

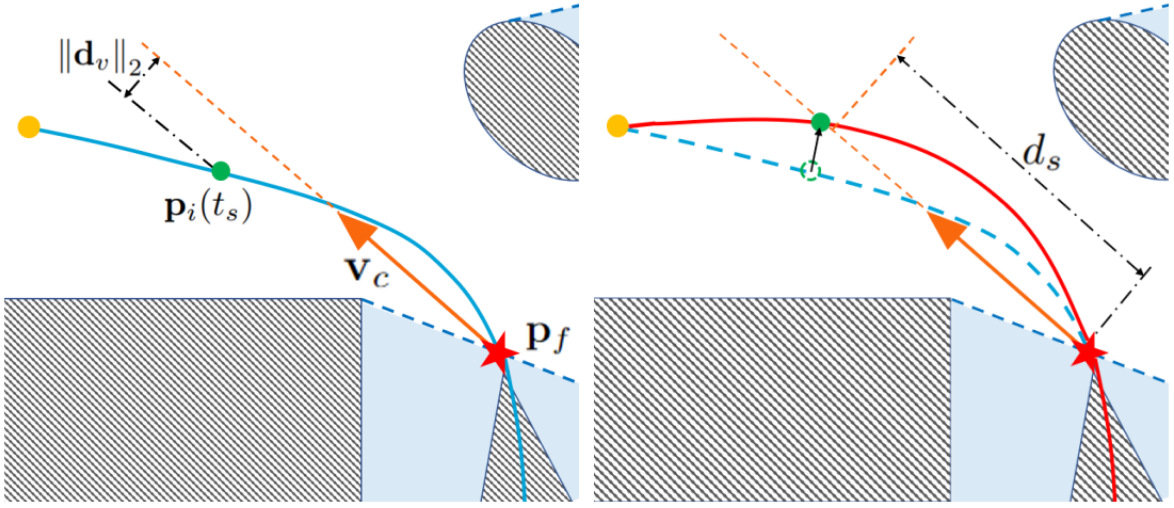

Figure 3.1 RAPTOR: Gradient-based motion planning with perception-aware strategies enhances robustness and safety in unknown environments. The trajectory is updated to maximize the visible area, providing the drone with longer distance and more time to react and prevent collisions [8].

Recent advancements in the domain of autonomous drone navigation have highlighted the efficacy of Deep Reinforcement Learning (DRL) methods over traditional optimization-based approaches. Studies such as [38, 39] have demonstrated that neural control policies can significantly outperform conventional model-based planning and control

methods. Specifically, these DRL approaches have been successful in navigating quadrotors through cluttered environments swiftly and with greater accuracy, achieving minimal travel time [40].

Agility in drone systems, especially in obstacle-dense environments, hinges critically on the capability to detect obstacles early and react promptly. To address this, several perception-aware methodologies leveraging Reinforcement Learning have been developed [2, 13]. These methods integrate sensory processing with control mechanisms, enhancing the drones' ability to respond to environmental challenges more effectively.

However, a significant challenge with RL-based approaches is their dependency on extensive trial-and-error data sets, which can be impractical in real-world scenarios. To mitigate this, recent research has explored the use of Imitation Learning to bootstrap RL solutions using existing datasets generated from classical approaches [2, 14, 16]. This synergy facilitates the development of robust models by significantly reducing the need for new data collection. Furthermore, some studies have adopted an RL-based model as a 'Teacher policy' to refine and surpass the limitations of classical optimization-based policies, effectively enhancing overall system performance [2, 41].

These studies have generally been conducted in environments with racing drones or static obstacles [2, 12]. Due to their focus on racing drones, they tend to perform well in the environments in which they were trained but show limited generalizability to other environments beyond their original training. In our work, we have applied these successful methods to environments with a high density of dynamic obstacles. By leveraging these innovative approaches, drone systems not only improve in agility but also in operational safety and efficiency, promising substantial advancements in the field of autonomous aerial navigation.

While the previous studies predominantly focus on static environments and applications such as drone racing, our research shifts the focus to achieving agile flight in environments populated with dynamic objects. To tackle the complexities of these dynamic scenarios, we implement imitation learning (IL) techniques. Unlike the methods that depend on an

optimization-based expert controller [16], our approach leverages a reinforcement learning (RL) model as the expert policy. This choice enhances our system's ability to adapt to novel and unforeseen situations, thereby improving agility and robustness. Our goal is not only to achieve high performance within the training environments but also to ensure effective generalization to new and dynamic contexts, thereby expanding the practical applicability and reliability.

4. PROPOSED METHOD

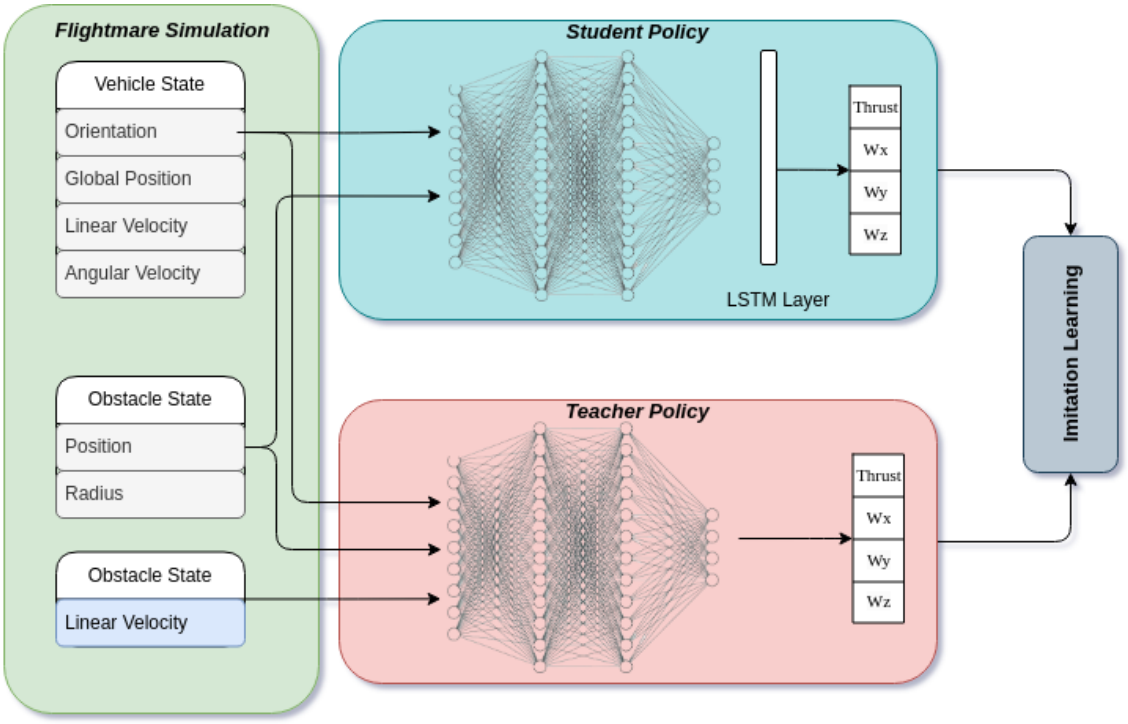

Figure 4.1 System Overview. The study focuses on training a state-based teacher policy that has access to obstacle velocity data. This knowledge is then transferred to a student policy using a Long Short-Term Memory (LSTM) based recurrent neural network framework.

In this section, the methodologies employed as solutions to the problem, which were outlined in the background section, will be explored. This chapter provides a comprehensive overview of the proposed method, including the general architecture, the interrelations of its components, the structure of the input and output values, and the training and testing processes.

The simulation environment used during training and testing will be discussed with details on the obstacles and technical specifications of the drone platform. Additionally, this section includes an explanation of the Teacher/Student models, their technical intricacies, and the configurations of the PPO algorithm. The reward function used during the model training will be elaborated upon, highlighting its components and their purposes. Furthermore, this

chapter explains how these elements contribute to the understanding of the results presented in the experiment section.

4.1. Method Overview

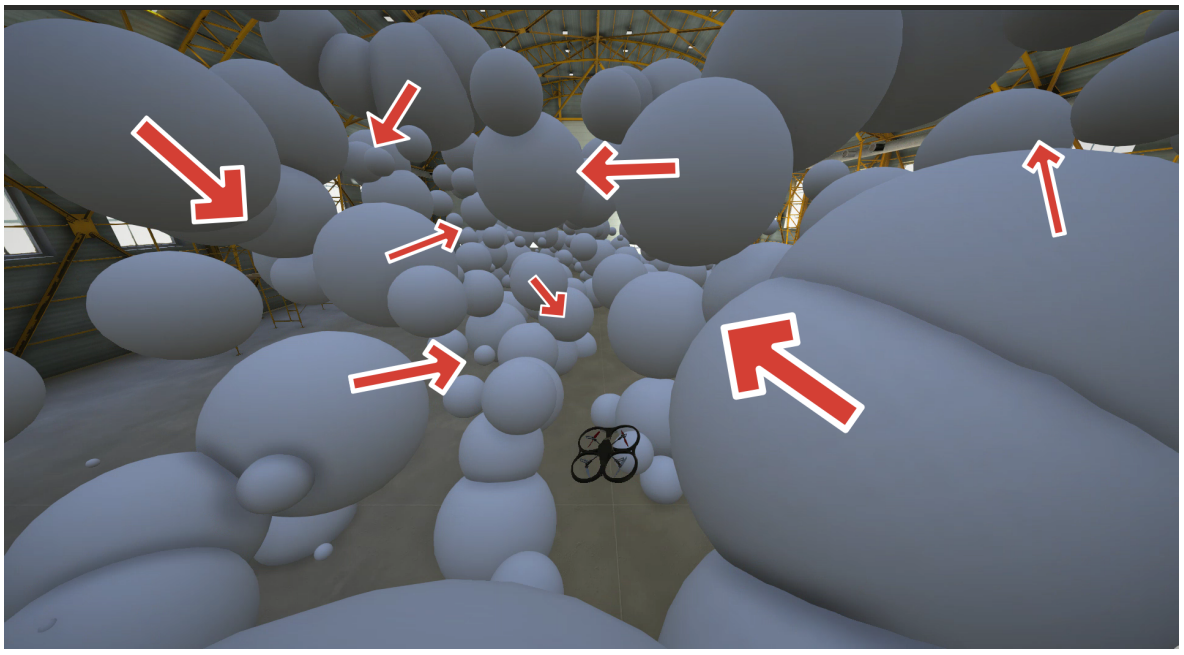

Figure 4.2 Utilizing velocity information of dynamic obstacles to train the model for navigation. The uncertainty of the movement direction of obstacles can cause collisions; however, the velocity information helps the model to understand their movement direction, leading to better navigation results. The red arrows highlight these dynamic movements, emphasizing the importance of velocity data in enhancing obstacle avoidance performance.

In this study, we explore the facilitation of agile flight within dynamic environments for drone systems. We propose a novel approach utilizing a deep reinforcement learning framework supplemented by imitation learning techniques. Our methodology incorporates two distinct policies: a teacher policy and a student policy. The teacher policy is privy to comprehensive environmental data, particularly the velocity of obstacles, which significantly enhances the model's capability to navigate swiftly around dynamic obstacles. Subsequently, the teacher policy functions as a mentor to the student policy during the imitation learning phase. The student policy employs a recurrent neural network architecture with Long Short-Term Memory (LSTM) layers, enabling it to infer obstacle velocity information indirectly rather

than relying on direct input. This strategy accelerates the learning process compared to traditional reinforcement learning methods that build knowledge from scratch.

The input features for the model include several parameters. Orientation is a matrix of size 9 that refers to the rotation matrix of the drone, helping in understanding its current orientation in space. Quad state position, quad state velocity, and quad state angular velocity are vectors of size 3 each, providing information about the drone's current position, linear velocity, and angular velocity, respectively. Obstacle observations is a vector of size 80, detailing the positions of 20 obstacles in the environment, each described by their coordinates (x, y, z) and size (r). The world box input is a vector of size 4 that indicates the distance to the edge of the map, helping the model to prevent the drone from exiting the defined boundaries. Additionally, obstacle velocity is a vector of size 60, providing information about the velocities (Vx, Vy, Vz) of 20 obstacles.

The output of the model consists of four parameters: the total thrust of the drone and the roll, pitch, and yaw rates. These control commands are then passed to the low-level flight controller, enabling the drone to navigate effectively in dynamic environments [42].

4.2. Environment

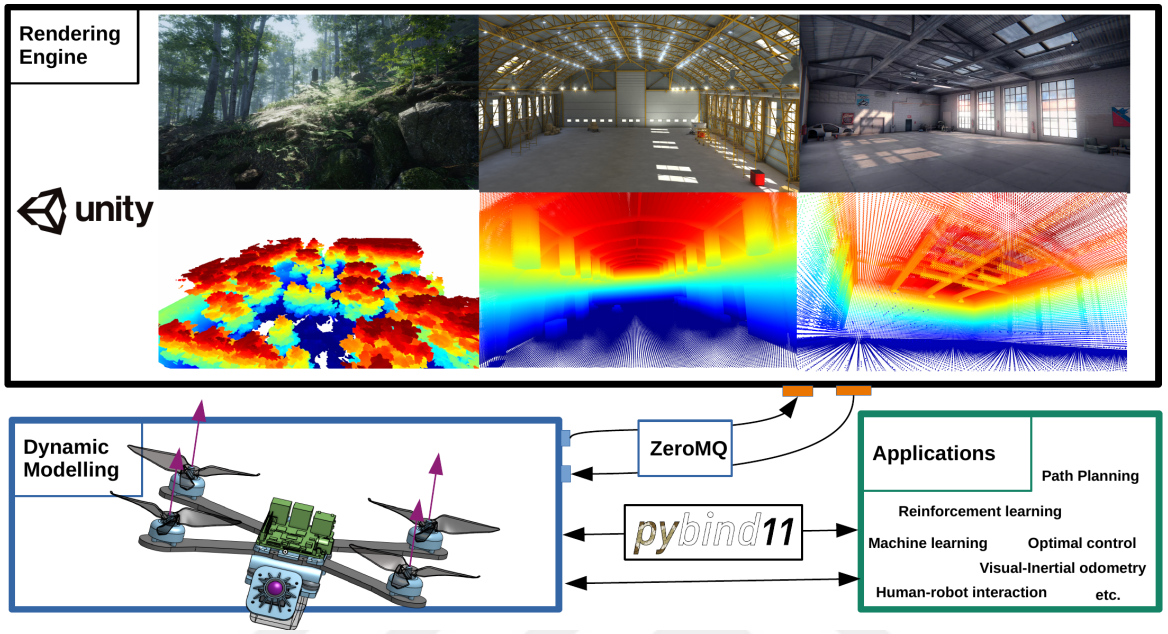

Figure 4.3 Flightmare System Overview [9].

In this research, we utilize the Flightmare simulator [9], an open-source, Unity-based drone simulation tool equipped with a flexible physics engine, serving as both our development and testing environment. The robust physics engine ensures realistic flight dynamics, crucial for accurate simulation of drone behavior.

Flightmare seamlessly integrates with the OpenAI-Gym framework [43], enabling the simultaneous operation of multiple environments through its parallel multi-agent support. This integration facilitates extensive testing and training scenarios. Additionally, the simulator is compatible with ROS and Gazebo, broadening its applicability for various robotic applications.

The simulator boasts a versatile sensor suite, providing RGB images, IMU data, depth information, segmentation maps, 3D point clouds, and optical flow data. These sensors are essential for creating realistic and varied training data, mimicking real-world drone perception systems.

Figure 4.4 Flightmare Parallel Multi Agent Support.

Our primary benchmarks are derived from the ”DodgeDrone Challenge” [3], which features environments populated with both static and dynamic obstacles at varying levels of complexity. The challenge environments are meticulously designed, featuring obstacles that vary in type, speed, acceleration, size, and density, thus providing a comprehensive testbed for evaluating drone performance under diverse conditions. The environments include moving obstacles with varying velocities and accelerations, designed to represent different obstacle densities.

Flightmare offers a variety of maps that mimic different locales, including warehouses, streets, forests, and wastelands, along with available Unity prefabs. These diverse environments are critical for testing the generalization capabilities of the developed algorithms.

Furthermore, we employ the Agilicious autopilot, a sophisticated control system that ensures precise maneuvering and stability, enabling the simulation of agile flight capabilities. The moderately lightweight drone model used within Flightmare provides a solid foundation for testing drone agility in dynamic environments, ensuring that our simulations are both realistic and relevant to real-world scenarios.

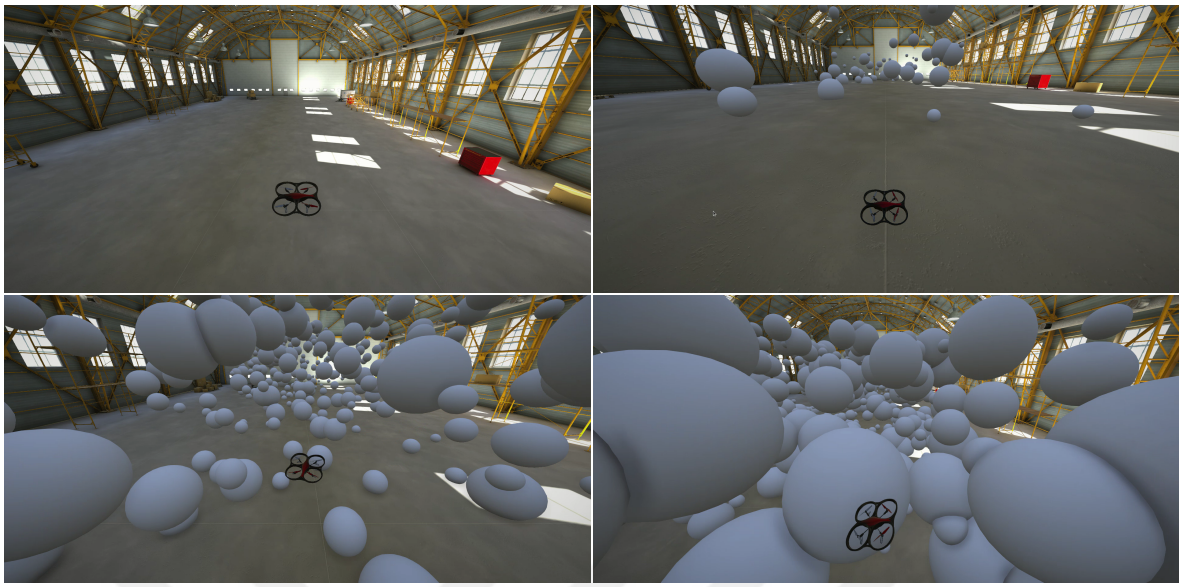

Figure 4.5 Flightmare Simulation Empty - Easy - Medium - Hard Level Environment

By leveraging these advanced simulation capabilities, we aim to develop and test reinforcement learning algorithms capable of navigating complex environments. The ”DodgeDrone Challenge” serves as a rigorous benchmark for our methods, ensuring that our solutions are robust and effective in real-world applications.

Parameter	**Value**
Mass (kg)	0.752
Thrust-to-Body Mappings (m)	Front Left: [0.075, 0.10, 0.0] Front Right: [0.075, -0.10, 0.0] Back Left: [-0.075, 0.10, 0.0] Back Right: [-0.075, -0.10, 0.0]
Max Angular Velocities (rad/s)	[6.0, 6.0, 2.0]
Inertia (kg·m^2)	[0.0025, 0.0021, 0.0043]
Torque Coefficient (N·m)	0.022
Motor Speed Range (rpm)	150 - 2800
Motor Response Time (s)	0.033
Thrust Map (N)	[1.562522e-6, 0.0, 0.0], Max Thrust = 8.50 N
Body Drag Coefficients	[0.00, 0.00, 0.00] (1st and 3rd Order), Horizontal: 0.00

Table 4.1 Quadrotor Dynamics Parameters

The quadrotor dynamics specified in our study represent the specifications of a drone designed to simulate agile flight capabilities in various environments. This drone is moderately lightweight, equipped with a sophisticated control system that allows for

precise maneuvering and stability even when confronted with obstacles. This simulation setup provides an excellent foundation for testing and enhancing drone agility in dynamic environments.

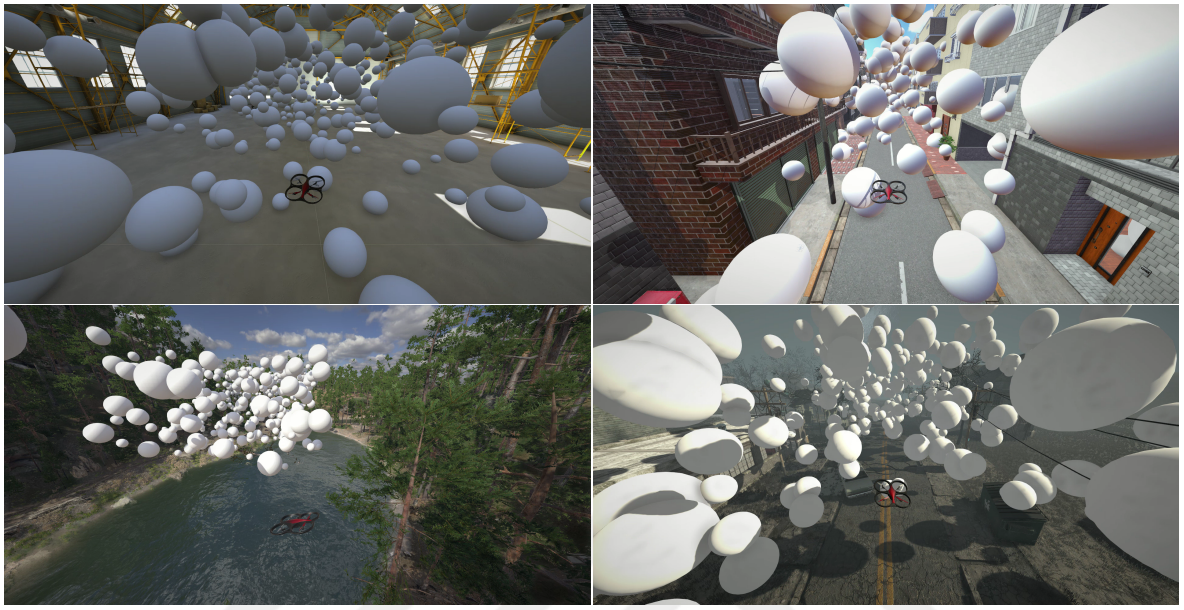

Figure 4.6 Environments: Warehouse - Street - Forest - Wasteland

4.3. Teacher Policy

The teacher policy serves as an expert model that provides guidance or supervision. It typically has access to comprehensive environmental data and uses this information to make optimal decisions in complex scenarios. The role of the teacher policy is to generate behavior and decisions that the student policy can learn from, effectively setting a performance benchmark and providing a source of high-quality, successful actions for the student to imitate. This helps ensure that the student learns effective strategies for navigating and responding within its operational environment.

The inputs of the models is quadrotor state information, obstacle position and velocity information. The outputs of the model is control actions of collective total thrust and bodyrates. The model architecture is a two layer of multilayer perceptron (MLP). We use StableBaseline3 Proximal Policy Optimization (PPO) algorithm with default settings. The

StableBaseline Proximal Policy Optimization (PPO) algorithm integrates concepts from both A2C, which employs multiple workers, and TRPO, which utilizes a trust region to enhance the actor. The core principle is to ensure that after each update, the new policy remains close to the previous one. To achieve this, PPO employs clipping to prevent excessively large updates.

The training is done with parallel 100 environment and the teacher policy can learn to fly in dynamic environment with up to 25m/s speed depends on the obstacle density.

4.4. Proximal Policy Optimization (PPO) Configurations

Parameter	**Value**
policy	MultiInputPolicy
learning_rate	7×10^{-4}
gae_lambda	0.95
γ	0.99
n_steps*	150
ent_coef	0.0
vf_coef*	0.3
max_grad_norm	0.5
batch_size	75000
clip_range	0.2
use_sde	False

Table 4.2 Parameters configuration

We utilized a specific configuration of parameters for the reinforcement learning model, tailored to optimize performance within the given environment. The choice of the policy model is crucial as it dictates the learning capabilities of the agent. The MultiInputPolicy was

selected to utilize multiple inputs, which was specifically adjusted from the default policy to meet the requirements of our model architecture.

In our setup, we adhered to the proven parameter configurations recommended by the Stable-Baseline3 [44] default values. These settings are selected based on extensive testing to ensure optimal performance across a variety of environments.

The learning rate, a crucial parameter for controlling the update magnitude in the model's weights during training, is set to a linear schedule starting at 7×10^{-4}. This setting allows the learning rate to decrease as the model progresses, optimizing convergence rates.

The discount factor (γ), set at 0.99, dictates the importance of future rewards over immediate ones, enabling the agent to prioritize long-term gains effectively.

The n_steps parameter, determining the number of steps to run for each environment per update, is crucial for defining the size of the rollout buffer. The product of n_steps and the number of environment copies (n_envs) must exceed one due to the requirements of advantage normalization.

The clipping parameter ($clip\_range$), set as a function of the current progress remaining from 1 to 0, helps in moderating policy updates and avoiding large deviations from the current policy, which could lead to destabilization.

Entropy coefficient (ent_coef) and value function coefficient (vf_coef), set at 0.0 and 0.5 respectively, are used to balance exploration by encouraging more diverse actions and accurately estimating value functions.

The maximum gradient norm (max_grad_norm), capped at 0.5, prevents the gradients from growing too large and potentially causing unstable training dynamics.

These parameters form the backbone of our model configuration, ensuring that the agent learns effectively and robustly in diverse settings.

4.5. Student Policy

The student policy is a model that learns to replicate the behavior of a teacher policy, which typically has access to more complete information about the environment. The student policy is trained to approximate the decisions of the teacher by observing and mimicking its actions, even when certain environmental data, like obstacle velocities, are not directly available to it. This approach enables the student model to perform tasks autonomously by leveraging the distilled knowledge from the teacher, improving its ability to operate effectively in complex scenarios.

The primary distinction between the teacher and student policies lies in their access to obstacle velocity information and the incorporation of a Recurrent architecture. The student policy aims to learn the movement trajectories of dynamic obstacles based on historical data, enabling rapid navigation within dynamic environments. Unlike the teacher policy, the student model does not directly incorporate obstacle velocity information into its inputs. The outputs from the student policy model dictate the control actions, specifically the collective total thrust and body rates. The architectural framework of the model consists of two layers of a multilayer perceptron (MLP) and a single 256-unit LSTM layer.

4.6. Reward Function

The reward function, represented in Equation (1), is a composite measure that includes components for linear velocity, survival, collision avoidance, and angular velocity penalties:

$$\begin{aligned} r(t) = r_{position} + r_{lin\_vel} + r_{acceleration} + r_{jerk} \\ + k_{\omega}\,||\omega|| + k_{coll}\,r_{coll} + r_{survive} \end{aligned} \tag{2}$$

The linear velocity reward is calculated using a masking operation that allows the drone flexibility in moving in various directions without incurring penalties. This adaptability is crucial for avoiding collisions by enabling movements to the left/right and up/down. To

ensure that these movements are not penalized, a component-wise product with a linear velocity mask is employed, which acts as a directional weight.

The survival reward is assigned a static value of 0.3, encouraging the drone to avoid ”death” when navigating through multiple obstacles and seeking escape routes. The collision penalty is computed as the sum of the exponential distances to the nearest N obstacles, enhancing the model's collision avoidance capabilities. Additionally, the angular velocity penalty aims to minimize the drone's angular movements, contributing to a more stable and controlled flight.

Due to the advantages of imitation learning, the training process for the student policy is significantly more efficient than that for the teacher policy, facilitating quicker adaptation and learning within dynamic environments.

$$r_{position} = k_{position} \cdot p_x \tag{3}$$

The position reward is proportional to the drone's position on the x-axis. It ensures that the drone maintains a fundamental forward motion. Despite the collision penalties from obstacles, the position reward encourages the drone to move forward.

$$\vec{v}_{rew_1} = \vec{v} - \vec{v}_{goal} \tag{4}$$

$$\vec{v}_{rew_2} = \vec{v} \tag{5}$$

We tried two different reward function as the velocity reward and obtained satisfactory results with both. In the first reward function, we use the difference between specific goal velocity, and the drone's velocity as a penalty. This ensures that the drone fly at the desired speed. As an alternative and currently used second reward function, directly use the drone's instantaneous velocity as a reward. This way, the faster the drone goes, the more reward it receives. This method allows the drone to reach much higher speeds and decide for itself when to slow down. Instead of forcing the model to maintain a specific speed, we let it make its own decisions.

$$v_{rew} = \log(1.0 + \min(10.0, |\vec{v}|)) \tag{6}$$

To prevent the reward value from becoming overly dominant as the speed increases, we use a log function to ensure that further acceleration beyond certain speeds does not provide additional advantage. This helps the model avoid focusing solely on speeding up at the expense of optimizing other reward values.

$$r_{lin\_vel} = -k_{vel} \cdot \sqrt{\sum_{i=1}^{n} (v_{rew,i} \cdot v_{mask,i})^2} \tag{7}$$

Where:

$$v_{mask} = \begin{bmatrix} 0.95 & 0.35 & 0.35 \end{bmatrix} \tag{8}$$

The linear velocity reward is proportional to the norm of the velocity error vector, masked by a linear velocity mask and scaled by a negative velocity coefficient.

$$\vec{a}_{error} = \frac{\vec{v} - \vec{v}_{old}}{\Delta t} \tag{9}$$

$$r_{acceleration} = k_{acceleration} \cdot \log(1 + ||\vec{a}_{error}||) \tag{10}$$

In dynamic environments, the most critical feature needed for agile flight is the ability to respond quickly. To highlight this, we utilized acceleration values as an advantage when encountering obstacles. By doing so, the drone was able to make sudden stops and rapid jumps, enhancing its maneuverability. This approach ensures that the drone can swiftly adjust its speed to avoid collisions and navigate through complex scenarios effectively. However, it is crucial to manage these acceleration values carefully, as excessively high values can lead to instability and unpredictable movements.

$$\vec{j}_{error} = \frac{\vec{a}_{error} - \vec{a}_{old}}{\Delta t} \tag{11}$$

$$r_{jerk} = k_{jerk} \cdot \log(1 + ||\vec{j}_{error}||) \tag{12}$$

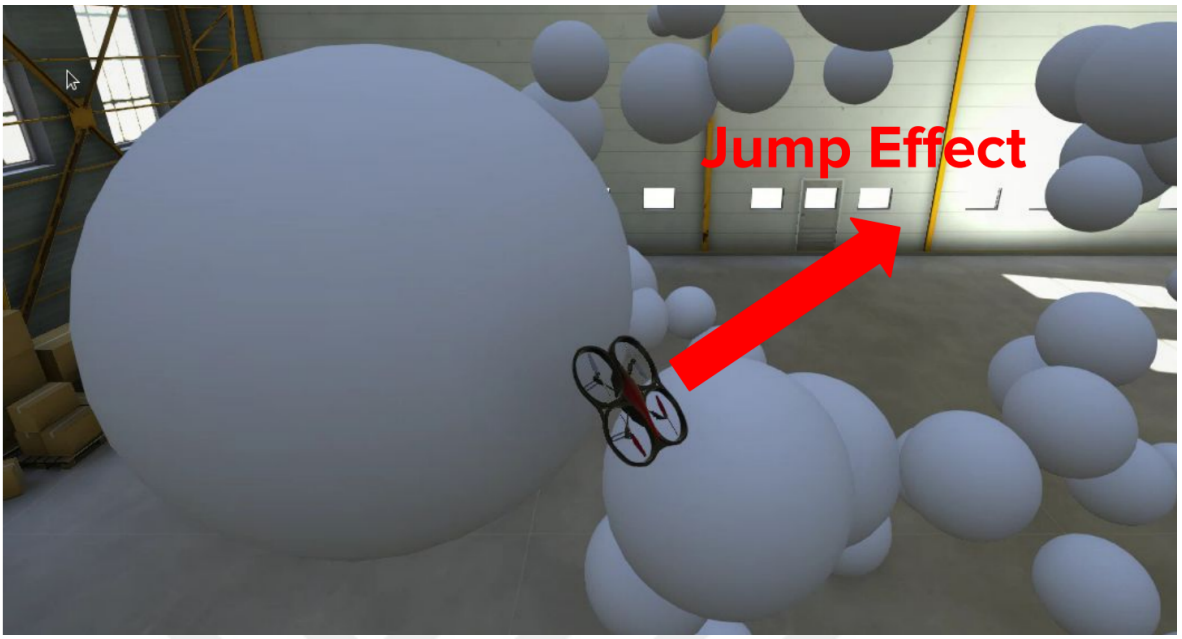

Figure 4.7 Impact of Jerk Reward Function on Obstacle Avoidance Performance.

Furthermore, we emphasized the importance of jerk values in agile flight. Jerk, being the rate of change of acceleration, plays a significant role in the drone's ability to make swift changes in direction. By incorporating jerk into our reward function, we enabled the drone to perform agile maneuvers such as quick directional shifts and rapid evasive actions. This capability is particularly beneficial in avoiding sudden obstacles and maintaining a fluid flight path. Nevertheless, it is important to note that high jerk values, if not properly controlled, can cause the drone to continuously spin or perform spiral maneuvers, potentially compromising its stability and control.

$$r_{ang\_vel} = k_\omega \cdot ||\vec{\omega}|| \tag{13}$$

The angular velocity penalty is designed to minimize the drone's angular movements, thereby promoting a more stable and controlled flight. By discouraging excessive rotations and oscillations, this penalty helps the drone maintain a steady orientation, which is crucial for precise navigation and maneuverability, especially in complex and dynamic environments. Consequently, the drone can better maintain its intended flight path, respond

more predictably to control inputs, and avoid erratic behavior that could compromise its mission or safety.

$$r_{coll} = k_{coll} \cdot \sum_{i=1}^{N} \exp\left(-1.0 \cdot (\text{relative\_dist}_i - r_{obs_i} - 0.1)\right) \tag{14}$$

The collision penalty is calculated as the sum of exponential functions of the relative distance to each obstacle, adjusted by the obstacle radius and a margin.

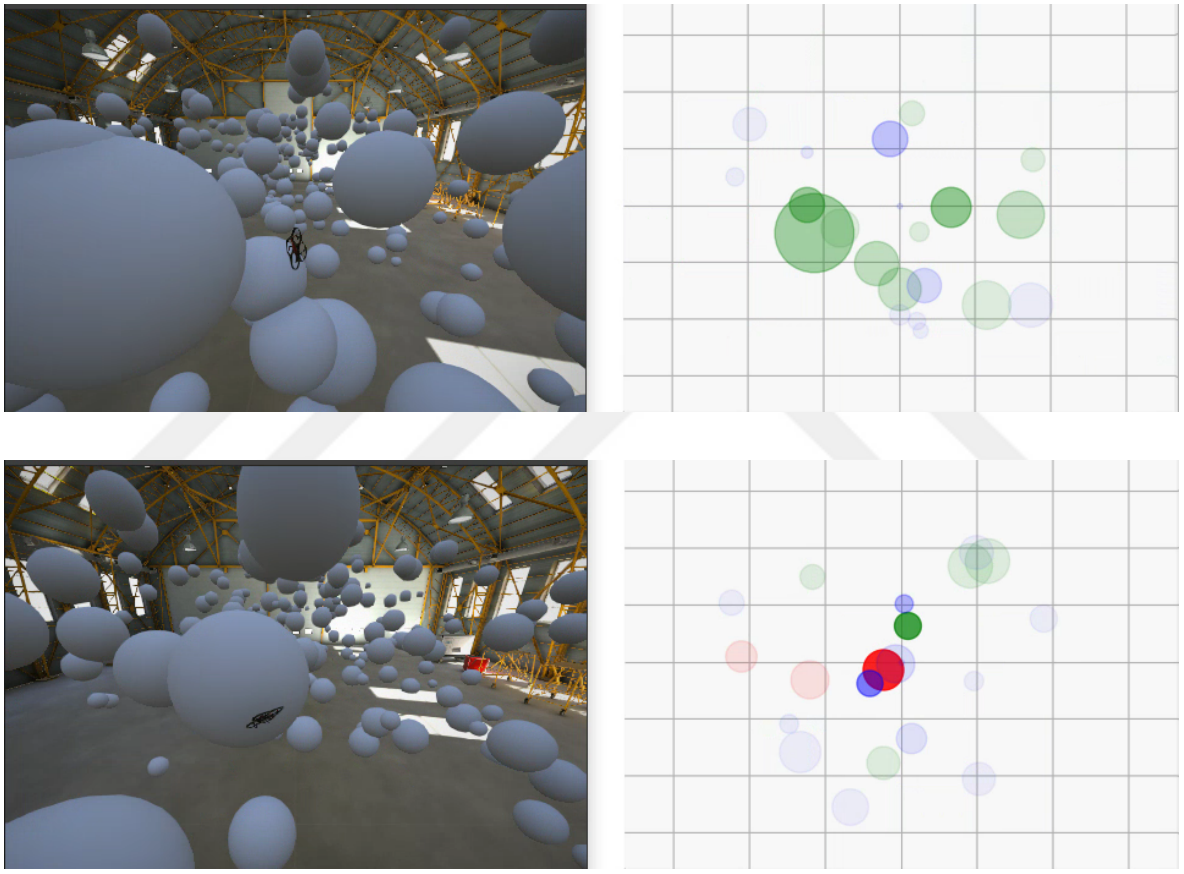

Figure 4.8 The top image shows a scenario where the collision penalty is not applied to obstacles in the movement direction, while the bottom image shows a scenario where the collision penalty is applied.

In Figure 4.8, the blue circles denote obstacles located behind the drone, while the green circles indicate obstacles situated in front of the drone that do not result in a collision penalty on the reward. The red circles represent obstacles that incur a collision penalty. The mechanism operates such that a collision penalty is applied if the velocity directions of

the obstacle and the drone intersect. Conversely, if the obstacles are in close proximity to the drone but do not intersect with its velocity direction, no collision penalty is imposed.

To calculate the collision penalty, the relative position vector ($\Delta\mathbf{p}$) between the drone and the obstacle is first determined:

$$\Delta\mathbf{p} = \mathbf{p}_{\text{obstacle}} - \mathbf{p}_{\text{drone}}$$

The distance is then normalized with respect to the velocity vector of the drone. The normalized distance vector ($\mathbf{d}_{\text{normalized}}$) is computed as:

$$\mathbf{d}_{\text{normalized}} = \left(\frac{\Delta p_1}{v_1}, \frac{\Delta p_2}{v_2}, \frac{\Delta p_3}{v_3} \right)$$

Finally, the norm of this normalized distance vector ($d_{\text{normalized}}$) is calculated:

$$d_{\text{normalized}} = \|\mathbf{d}_{\text{normalized}}\|$$

This normalized distance helps in assessing the collision risk, ensuring that if the velocity direction of the obstacle and the drone intersect, a collision penalty is applied. Otherwise, even if the obstacles are close to the drone, no collision penalty is imposed. This approach allows for a more nuanced evaluation of potential collisions based on both proximity and relative motion.

$$r_{survive} = k_{survive}, \tag{15}$$

The survival reward is a constant reward given for the drone's survival.

5. EXPERIMENTAL RESULTS

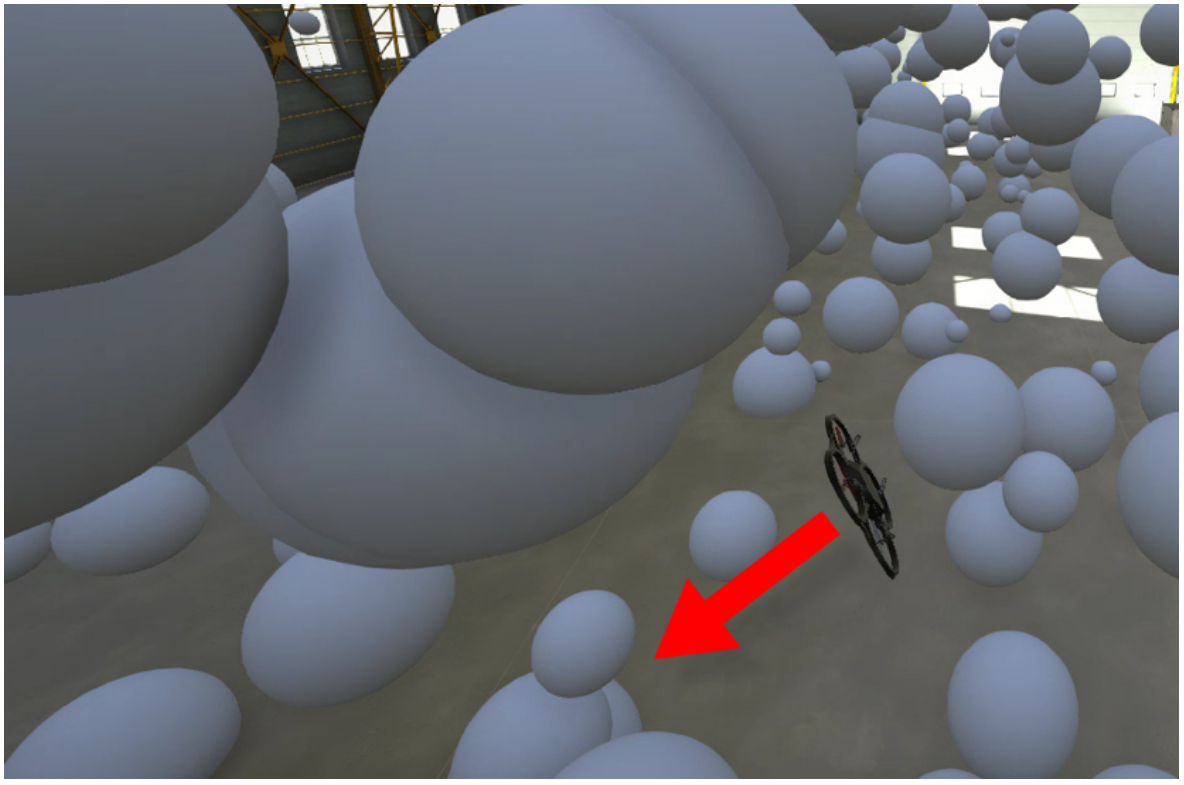

Figure 5.1 An example of an unsuccessful flight in a dynamic environment resulted in a collision.

Prior to the development of the main experimental framework, a preliminary evaluation was conducted to determine the suitability of the Proximal Policy Optimization (PPO) algorithm for drone navigation in environments laden with static obstacles. This initial phase was critical as it utilized both drone state information and obstacle location data to train the model, setting a foundational benchmark for subsequent experiments. The results from this phase were encouraging, demonstrating that the PPO algorithm could effectively control drone movements in a predictably static setting.

Following this initial success, the research focus shifted towards more complex, dynamic environments where obstacles not only existed but moved with variable speeds and directions. The retraining of the RL model under these new conditions aimed to test the adaptability of the same architecture used previously in static environments. However, the model's performance during these trials was suboptimal; it struggled to maintain its

efficacy in obstacle avoidance when confronted with unpredictably moving objects. Analysis indicated that the drone's response time was inadequate, likely due to the insufficient processing of rapidly changing environmental data.

To enhance the model's capability in dynamic settings, a significant modification was introduced. The input data for the model training was expanded to include not only the positions but also the velocities of obstacles. This additional layer of data was hypothesized to provide the drone with a more comprehensive understanding of its environment, thereby improving its predictive and reactive navigational responses. Upon retraining with this augmented data set, the model demonstrated a marked improvement in managing scenarios with moving obstacles, showcasing its robustness in more realistically chaotic conditions.

Policy	**Static Obstacle**	**Dynamic Obstacles**	**Successful**	**Training Time**
PPO	✓	✗	✓	Low
PPO	✓	✓	✗	Low
Recurrent PPO	✓	✗	✓	High
Recurrent PPO	✓	✓	✓	High
PPO + Velocity	✓	✗	✓	Low
PPO + Velocity	✓	✓	✓	**Low**

Table 5.1 Performance comparison of PPO and Recurrent PPO policies under different settings. The PPO algorithm with velocity information stands out as the solution that successfully navigates both static and dynamic obstacles while maintaining relatively fast training times.

However, the practical application of this improved model in real-world scenarios introduced another challenge the difficulty of obtaining real-time velocity data of surrounding obstacles. To address this, a novel approach was proposed: leveraging the historical positional data to infer the velocity of obstacles. This led to the adoption of a Long Short-Term Memory (LSTM) based recurrent neural network architecture, effectively mimicking the processing of velocity data.

An extensive training session was then conducted with the LSTM-based model in a controlled environment populated with dynamic obstacles. The outcomes were promising; the model not only met but in some respects, exceeded the performance benchmarks set by its predecessors. It managed agile navigation without direct access to velocity data, relying solely on the positional changes of obstacles, thus simplifying the data acquisition process

in actual deployment scenarios. However, this sophisticated model required approximately three times longer training periods compared to previous iterations, posing a significant drawback in terms of efficiency.

Policy Name	Reward Value	Number of Steps	Time
PPO	6.08	101.3M	21m 8s
Recurrent PPO	6.43	99.75M	1h 10m 36s

Table 5.2 Performance Comparison of PPO and Recurrent PPO Policies

The Table 5.2 shows that the training times of the Proximal Policy Optimization (PPO) and its recurrent variant (Recurrent PPO). Both algorithms were trained over 100 million iterations. The results indicate a significant difference in training durations; the PPO algorithm completed its training in 21 minutes, whereas the Recurrent PPO required 1 hour and 10 minutes to complete the same number of iterations. This constitutes a more than threefold increase in training time for the Recurrent PPO compared to the standard PPO.

To overcome the extended training duration, an innovative solution was tested. The LSTM model was subjected to an imitation learning process, using the velocity-trained PPO model as a 'teacher'. This approach aimed to compress the training time by directing the LSTM model through pre-established successful navigation patterns derived from the PPO model. The experiment yielded positive results, indicating that while there was a slight reduction in overall model performance, the rate of training improved substantially, enabling quicker deployment of the model in dynamic environments.

5.1. Performance Evaluation

In evaluating the performance of models designed for agile drone flight, our primary metrics included the drone's speed and the time it took to reach a target location. Higher speeds were especially valued because the main goal was to achieve agile, efficient movement. Reaction time also played a crucial role; this is the time it takes for the drone to respond to sensor inputs and execute commands, impacting how quickly the drone can adjust to changes in its environment.

Metric	Description
Speed	Flight velocity measured in meters per second (m/s)
Time	Process completion time in milliseconds (ms)
Success Rate	Percentage of trials successfully completed
Latency	Measure of time delay in milliseconds (ms)

Table 5.3 Summary of Key Performance Metrics

To measure how well the drone navigated spaces with moving obstacles, we used a ”success rate” metric which was determined by the frequency of collisions during 10 experimental trials. This rate is important as it helps gauge how reliably the drone can operate in varied, unpredictable conditions.

It is important to note that performance metrics in real-world applications may vary due to physical constraints of the drone, such as onboard hardware capabilities and battery limitations. The number and closeness of obstacles also significantly affected the drone's performance. In simpler environments with fewer obstacles, the drone moved faster and more efficiently. But as obstacle density increased, the drone slowed down, and it took longer to reach its destination. The arrangement and movement of obstacles—like their size and speed—made navigating through them increasingly difficult

Method	Type	Latency	Hardware
Liu et al. [45]	Control	160ms	3.4 GHz dual-core i7 Intel NUC
Burri et al. [46]	Control	$>$ 40ms	AscTec Firefly 2.4 GHz Controller
Chen et al. [47]	Control	$>$ 34ms	3.20 GHz Intel Core i5-4570 CPU
Lopez et al. [48]	Control	$\leq$ 5.06ms	2.70GHz Intel Core i7-2620M
CL-RRT* [49]	Control	$\geq$ 650ms	Intel Xeon 2.4GHz
FastPlanner [50]	Control	65.2ms	3.00 GHz i7-5500U
Reactive [51]	Control	19.1ms	Intel i7 NUC
Agile autonomy [52]	RL	10.3ms	NVIDIA Jetson TX2
Y. Song et al. [2]	RL	1.41ms	N/A
Our method avg.	**RL**	**0.75ms**	**GeForce RTX 4070 Ti**

Table 5.4 Comparisons with other similar trajectory replanning methodologies. Courtesy of Hasanzade et al.[1] and Y. Song et al. [2].

Table 5.4 illustrates the latency and hardware specifications of various trajectory replanning methodologies. The data clearly highlights the limitations of classical methods in meeting

the rapid response times required by high-speed/agile drone systems. These methods exhibit significant latency, which is primarily attributed to their sequential architecture involving perception, planning, and control stages, and their reliance on CPU-based processors. This architectural approach and processing methodology result in considerable latency differences.

In contrast, reinforcement learning based approaches leverage GPU-based graphics processors to produce control outputs with substantially lower latency. Our proposed solution, operating on significantly more capable hardware, demonstrates a marked performance improvement. However, even when compared to classical methods, RL-based approaches implemented on onboard Jetson hardware exhibit notable performance gains. This underscores the superiority of RL-based methods in terms of latency, enabling more efficient and responsive control for agile drone systems.

Policy Type	**Max Speed [m/s]**	**Avg. Speed [m/s]**
Y. Song et al. [2]	15.1	-
Agile autonomy [52]	13.0	5-10
Reactive [51]	12.0	-
FASTER [53]	7.8	-
Liu et al. [45]	4.0	-
RAPTOR [8]	3.41	2.14
Our Teacher (PPO)	**27 $\pm$3.0**	**12.0$\pm$2.5**
Our Student (RecurrentPPO)	**23 $\pm$3.0**	**9.70$\pm$1.5**

Table 5.5 Comparison of different policy and methods performance.

Table 5.5 compares the performance of different policies based on maximum speed and average speed metrics. Our model, particularly the Teacher (PPO) and Student (Recurrent PPO), demonstrates a significant improvement in both metrics. Specifically, our Teacher policy achieves a maximum speed of 27 m/s with an average speed of 12.0 m/s, while the Student policy achieves a maximum speed of 23 m/s with an average speed of 9.7 m/s. These results highlight that, in comparison to other trajectory re-planning methods, our model consistently operates at substantially higher speeds. Although other studies like Y. Song et al.[2] report similar velocities, they typically pertain to static environments. This

distinction highlights the superiority of our approach in dynamic settings, where proficiency in handling high-speed flights is imperative.

An important consideration when making these comparisons is the inherent differences in the conditions under which the methods operate. These differences include the hardware platforms used, whether the processing is performed on-board or off-board, and the environments in which tests were conducted (indoor or outdoor). Additionally, the performance capabilities of the drone systems, including the technical specifications of the drone platforms, introduce further variability. These factors collectively hinder the ability to make a fair and direct comparison of the results.

Policy Type	**Max Speed [m/s]**	**Avg. Speed [m/s]**	**Success Rate [%]**
Safe Policy	6 ±3.0	5.0±2.5	100%
Medium Policy	10 ±3.0	6.5±2.5	80%± 10
Fast Policy	**27 ±3.0**	**12.0±2.5**	65%± 10

Table 5.6 Comparison of different policy settings.

Table 5.6 presents the performance of various configurations of the policy, focusing on three different types: Safe Policy, Medium Policy, and Fast Policy. As indicated in the table, the speed of the drone significantly influences the success rate. The Safe Policy, operating at a maximum speed of 6 m/s and an average speed of 5 m/s, achieves a 100% success rate. This suggests that lower speeds enhance the drone's ability to navigate through obstacles effectively.

The Medium Policy, with a maximum speed of 10 m/s and an average speed of 6.5 m/s, shows a slightly reduced success rate of 80% ± 10%. This demonstrates that while the drone can still perform well at moderate speeds, there is a notable decrease in success compared to the Safe Policy.

The Fast Policy, which allows the drone to reach a maximum speed of 27 m/s and an average speed of 12 m/s, results in a significantly lower success rate of 65% ± 10%. This indicates that higher speeds compromise the drone's navigational accuracy and increase the likelihood of failure.

This experiment was conducted in an environment containing 350 dynamic obstacles and 10 static obstacles. The number, size, and density of obstacles also impact the results. If the obstacle density is sparse enough to allow the drone to maneuver freely, the drone tends to produce more successful outcomes. Conversely, in environments with a high density of obstacles, even if the total number is relatively small, the drone may struggle to achieve the same success rate. Due to the lack of information about alternative methods, a fair comparison is not feasible.

Policy Type	**Max Speed [m/s]**	**Type**	**Success Rate [%]**	**Time [s]**
Delft Dodgers [3]	-	Vision-based	100%	20.56
Human [3]	-	-	-	19.7
JSK Robotics [3]	-	State-based	30%± 10	9.3 (best)
Our Teacher (PPO)	**27 ±3.0**	State-based	65%± 10	**6.0 ± 0.5**
Our Student (RecurrentPPO)	**23 ±3.0**	State-based	55%± 10	**6.2 ± 1.5**

Table 5.7 Comparison of different methods performance in the DodgeDrone Challenge [3].

Table 5.7 focuses on the performance of different methods within the specific context of the DodgeDrone Challenge. Here, the success rate and completion time are also considered alongside speed metrics. Our Teacher (PPO) policy again demonstrates superior performance with a maximum speed of 27 m/s, an approximate 70% success rate, and an average completion time of 6.0 seconds. The Student (Recurrent PPO) policy follows closely with a maximum speed of 23 m/s, an approximate 60% success rate, and an average completion time of 6.2 seconds.

In comparison, other participants like JSK Robotics and Delft Dodgers, despite their high success rates, do not achieve the same speed efficiency. Notably, even if we assume that the human participant has a high success rate, they exhibit a significantly inferior completion time of 19.7 seconds. These findings illustrate the competitive edge of our models in a challenging environment, demonstrating the practical feasibility and effectiveness of advanced RL strategies for agile drone navigation.

When examining the instances of failure, it is typically observed that errors occur as the drone accelerates to high speeds in areas without obstacles and then encounters obstacles while cruising at these high speeds. If the model can account for the frequency and risk levels of obstacles in the environment, it can navigate at more appropriate speeds. As shown in Table 5.6, the drone achieves a 100% success rate at cruising speeds of around 6 m/s, but the success rate decreases as speed increases. While the objective is to achieve agile flight, preventing inappropriate acceleration will enhance overall success.

5.2. Flight Dynamics and Rewards

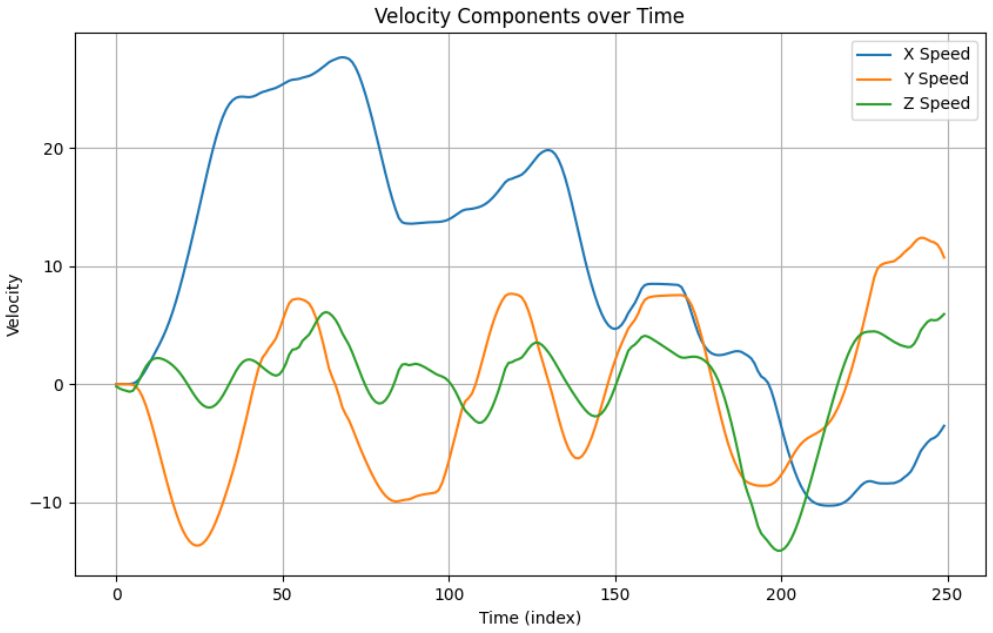

Figure 5.2 Velocity values of the drone in an environment with dynamic obstacles.

The Figure 5.2 provides insight into the drone's maneuvers to avoid collisions. Initially, there is a noticeable acceleration in the forward direction, which can be attributed to the influence of the position reward mechanism, highlighting the drone's initial speed increase. This acceleration occurs because the drone receives higher rewards for forward positions, motivating it to spend as much time as possible in forward locations. As the drone encounters obstacles, it makes several adjustments in both the Y and Z axes. The variations in the Y-axis

speed, in particular, show significant lateral movements to avoid collisions, while the changes in the Z-axis speed reflect vertical adjustments.

Furthermore, the graph indicates moments where the drone slows down, likely to better navigate through clusters of obstacles. As the drone approaches the end of the mapped area, a deceleration is observed, suggesting a tendency to stop, possibly indicating the completion of the navigation task or a deliberate pause to ensure safe stopping. Instead of ending the iteration with a penalty of -1 for exiting the predefined area, the drone attempts to maximize rewards by maneuvering within the forward position as much as possible.

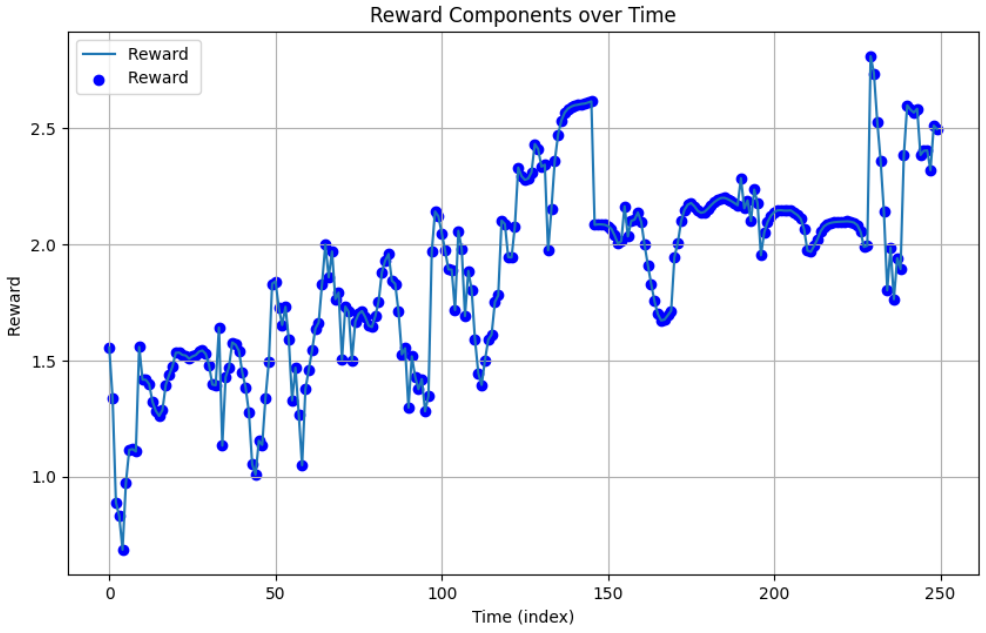

Figure 5.3 Reward values of the drone during the drone's forward movement.

The Figure 5.3 illustrates the reward values obtained by the drone throughout the iteration. Generally, as time progresses (i.e., as the drone moves forward), an increasing trend in the reward values can be observed. This indicates that the drone is effectively navigating through the environment.

However, there are numerous fluctuations in the reward values. These variations are primarily due to the ”collision penalty” incurred when the drone approaches obstacles within a defined

distance margin. Each downward spike in the reward value reflects the penalty applied, indicating that the drone has approached an obstacle too closely. Following these penalties, the reward values tend to rise again, indicating the drone's corrective actions to avoid further collisions and continue its forward movement.

This pattern highlights the dynamic nature of the drone's navigation, where it constantly adapts its path to maximize rewards while avoiding obstacles. The overall increasing trend in rewards, despite occasional penalties, demonstrates the effectiveness of the drone's policy in learning and optimizing its trajectory in a challenging environment.

6. CONCLUSION AND FUTURE WORK

This thesis explored the integration of Reinforcement Learning (RL) and Imitation Learning (IL) to enable agile flight in dynamic environments. By leveraging the Proximal Policy Optimization (PPO) algorithm, a teacher policy was developed that could navigate through environments with both static and dynamic obstacles. The teacher policy utilized comprehensive state information, including obstacle velocities, which significantly enhanced its navigation capabilities.

To transfer this knowledge, Behavioral Cloning (BC) was employed to train a student policy. The student policy, implemented using a Long Short-Term Memory (LSTM) based recurrent neural network, was able to infer obstacle velocities indirectly from positional data. This approach reduced the computational burden and allowed for effective navigation without direct access to velocity information. The results demonstrated that the student policy could achieve a high success rate in dynamic environments, albeit with some performance trade-offs compared to the teacher policy.

Overall, the hybrid RL-IL framework proved effective in addressing the computational challenges of agile flight and demonstrated the potential for real-time application in complex, dynamic settings. The significant reduction in training time and improved robustness of the student policy highlight the practicality of this approach for real-world drone navigation.

While this thesis demonstrated the efficacy of a state-based approach to drone navigation using Reinforcement Learning (RL) and Imitation Learning (IL), several future directions can further enhance the system's capabilities and generalizability.

- **Vision-Based and Perception-Aware Models**: The current system relies on state-based inputs for navigation. Future research should explore the integration of vision-based inputs and develop perception-aware models that enable drones to interpret and navigate environments using visual data and other sensory inputs. Enhancing the perception capabilities of the drone is crucial for better obstacle

detection and avoidance. By leveraging advanced computer vision techniques and integrating sensory data from cameras, LiDAR, and other sensors, the system's adaptability to complex, unstructured environments can be significantly improved. These models will enhance the drone's ability to perceive and react to its surroundings, leading to more robust and efficient navigation.

- **Generalizability of Vision-Based Models**: One of the primary challenges with vision-based navigation is the generalization problem. Future work should focus on developing robust algorithms that can generalize across different environments and conditions. This involves training models on diverse datasets and employing techniques such as domain adaptation and transfer learning to ensure the system performs reliably in new and unseen scenarios.

- **Combining State and Vision-Based Inputs**: A hybrid approach that combines state-based and vision-based inputs could leverage the strengths of both methods. Such a system would utilize state information for precise control and navigation, while visual data would provide contextual awareness and obstacle detection. This combination can enhance overall performance and robustness.

- **Advanced Reward Functions**: Future research can explore more sophisticated reward functions that incorporate visual and state-based information. These reward functions can be designed to encourage smoother trajectories, reduce oscillatory movements, and improve obstacle avoidance, leading to more efficient and agile flight.

- **Real-World Testing and Deployment**: Implementing and testing the developed models on actual drones in real-world environments will be critical. This involves addressing hardware constraints, ensuring reliable sensor integration, and validating the system's performance under varying conditions. Real-world testing will provide valuable insights into the practical challenges and limitations of the models.

By addressing these areas, future research can significantly advance the field of autonomous drone navigation, making drones more versatile, efficient, and capable of operating in diverse and challenging environments.

REFERENCES

[1] Mehmet Hasanzade and Emre Koyuncu. A dynamically feasible fast replanning strategy with deep reinforcement learning. Journal of Intelligent & Robotic Systems, 101(1):13, **2021**.

[2] Yunlong Song, Kexin Shi, Robert Penicka, and Davide Scaramuzza. Learning perception-aware agile flight in cluttered environments. In 2023 IEEE International Conference on Robotics and Automation (ICRA), pages 1989–1995. IEEE, **2023**.

[3] Robotics and Perception Group, University of Zurich. Dodge drone: Autonomous agile flight with obstacles at icra 2022, **2022**. Accessed: 2024-05-21.

[4] Hazim Shakhatreh, Ahmad H. Sawalmeh, Ala Al-Fuqaha, Zuochao Dou, Eyad Almaita, Issa Khalil, Noor Shamsiah Othman, Abdallah Khreishah, and Mohsen Guizani. Unmanned aerial vehicles (uavs): A survey on civil applications and key research challenges. IEEE Access, 7:48572–48634, **2019**. doi:10.1109/ACCESS.2019.2909530.

[5] OpenAI. Introduction to reinforcement learning, **2024**. Accessed: 2024-06-02.

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

[7] Erick Rodríguez-Hernandez, Juan Irving Vasquez-Gomez, and Juan Carlos Herrera-Lozada. Flying through gates using a behavioral cloning approach. In 2019 International Conference on Unmanned Aircraft Systems (ICUAS), pages 1353–1358. **2019**. doi:10.1109/ICUAS.2019.8798172.

[8] Boyu Zhou, Jie Pan, Fei Gao, and Shaojie Shen. Raptor: Robust and perception-aware trajectory replanning for quadrotor fast flight. IEEE Transactions on Robotics, 37(6):1992–2009, **2021**.

[9] Yunlong Song, Selim Naji, Elia Kaufmann, Antonio Loquercio, and Davide Scaramuzza. Flightmare: A flexible quadrotor simulator. In Proceedings of the 2020 Conference on Robot Learning, pages 1147–1157. **2021**.

[10] Tianhao Zhang, Gregory Kahn, Sergey Levine, and Pieter Abbeel. Learning deep control policies for autonomous aerial vehicles with mpc-guided policy search. In 2016 IEEE international conference on robotics and automation (ICRA), pages 528–535. IEEE, **2016**.

[11] Yunlong Song, Angel Romero, Matthias Müller, Vladlen Koltun, and Davide Scaramuzza. Reaching the limit in autonomous racing: Optimal control versus reinforcement learning. Science Robotics, 8(82), **2023**. ISSN 2470-9476. doi:10.1126/scirobotics.adg1462.

[12] Elia Kaufmann, Lukas Bauersfeld, Antonio Loquercio, et al. Champion-level drone racing using deep reinforcement learning. Nature, 620:982–987, **2023**. doi:10.1038/s41586-023-06419-4.

[13] Jiawei Fu, Yunlong Song, Yan Wu, Fisher Yu, and Davide Scaramuzza. Learning deep sensorimotor policies for vision-based autonomous drone racing. In 2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pages 5243–5250. IEEE, **2023**.

[14] Junjie Lu, Bailing Tian, Hongming Shen, Xuewei Zhang, and Yulin Hui. Lpnet: A reaction-based local planner for autonomous collision avoidance using imitation learning. IEEE Robotics and Automation Letters, 8(11):7058–7065, **2023**. doi:10.1109/LRA.2023.3314350.

[15] Antonio Loquercio, Elia Kaufmann, René Ranftl, Matthias Müller, Vladlen Koltun, and Davide Scaramuzza. Learning high-speed flight in the wild. Science Robotics, 6(59):eabg5810, **2021**.

[16] Jesus Tordesillas and Jonathan P. How. Deep-panther: Learning-based perception-aware trajectory planner in dynamic environments, **2023**.

[17] Elia Kaufmann, Antonio Loquercio, René Ranftl, Matthias Müller, Vladlen Koltun, and Davide Scaramuzza. Deep drone acrobatics, **2020**.

[18] Jiarong Lin, Luqi Wang, Fei Gao, Shaojie Shen, and Fu Zhang. Flying through a narrow gap using neural network: an end-to-end planning and control approach. In 2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pages 3526–3533. IEEE, **2019**.

[19] R.S. Sutton and A.G. Barto. Reinforcement learning: An introduction. IEEE Transactions on Neural Networks, 9(5):1054–1054, **1998**. doi:10.1109/TNN.1998.712192.

[20] 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**. doi:10.1038/nature16961.

[21] Petar Kormushev, Sylvain Calinon, and Darwin G. Caldwell. Reinforcement learning in robotics: Applications and real-world challenges. Robotics, 2(3):122–148, **2013**. ISSN 2218-6581. doi:10.3390/robotics2030122.

[22] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D. Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, Xin Zhang, Jake Zhao, and Karol Zieba. End to end learning for self-driving cars, **2016**.

[23] Haochong Chen, Xincheng Cao, Levent Guvenc, and Bilin Aksun-Guvenc. Deep-reinforcement-learning-based collision avoidance of autonomous driving system for vulnerable road user safety. Electronics, 13(10), **2024**. ISSN 2079-9292. doi:10.3390/electronics13101952.

[24] Jingyi Xie, Xiaodong Peng, Haijiao Wang, Wenlong Niu, and Xiao Zheng. Uav autonomous tracking and landing based on deep reinforcement learning strategy. Sensors, 20(19), **2020**. ISSN 1424-8220. doi:10.3390/s20195630.

[25] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, **2017**.

[26] 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**.

[27] John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. Trust region policy optimization, **2017**.

[28] OpenAI, :, Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemysław Debiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, Rafal Józefowicz, Scott Gray, Catherine Olsson, Jakub Pachocki, Michael Petrov, Henrique P. d. O. Pinto, Jonathan Raiman, Tim Salimans, Jeremy Schlatter, Jonas Schneider, Szymon Sidor, Ilya Sutskever, Jie Tang, Filip Wolski, and Susan Zhang. Dota 2 with large scale deep reinforcement learning, **2019**.

[29] Oriol Vinyals, Igor Babuschkin, Wojciech M. Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David Choi, Richard Powell, Timo Ewalds, Petko Georgiev, Junhyuk Oh, Dan Horgan, Manuel Kroiss, Ivo Danihelka, Aja Huang, L. Sifre, Trevor Cai, John P. Agapiou, Max Jaderberg, Alexander Sasha Vezhnevets, Rémi Leblond, Tobias Pohlen, Valentin Dalibard, David Budden, Yury Sulsky, James Molloy, Tom Le Paine, Caglar Gulcehre, Ziyun Wang, Tobias Pfaff, Yuhuai Wu, Roman Ring, Dani Yogatama, Dario Wünsch, Katrina McKinney, Oliver Smith, Tom Schaul, Timothy P. Lillicrap, Koray Kavukcuoglu, Demis Hassabis, Chris Apps, and David Silver. Grandmaster level in starcraft ii using multi-agent reinforcement learning. Nature, 575:350 – 354, **2019**.

[30] Jianyu Chen, Bodi Yuan, and Masayoshi Tomizuka. Model-free deep reinforcement learning for urban autonomous driving, **2019**.

[31] Chao Wang, Jian Wang, Yuan Shen, and Xudong Zhang. Autonomous navigation of uavs in large-scale complex environments: A deep reinforcement learning approach. IEEE Transactions on Vehicular Technology, 68(3):2124–2136, **2019**.

[32] Ahmed Hussein, Mohamed Medhat Gaber, Eyad Elyan, and Chrisina Jayne. Imitation learning: A survey of learning methods. ACM Computing Surveys (CSUR), 50(2):1–35, **2017**.

[33] Faraz Torabi, Garrett Warnell, and Peter Stone. Behavioral cloning from observation. arXiv preprint arXiv:1805.01954, **2018**.

[34] Andrew Y Ng, Stuart Russell, et al. Algorithms for inverse reinforcement learning. In Icml, volume 1, page 2. **2000**.

[35] Brenna D Argall, Sonia Chernova, Manuela Veloso, and Brett Browning. A survey of robot learning from demonstration. Robotics and autonomous systems, 57(5):469–483, **2009**.

[36] Ahmed Hussein, Mohamed Medhat Gaber, and Eyad Elyan. Deep active learning for autonomous navigation. In Engineering Applications of Neural Networks: 17th International Conference, EANN 2016, Aberdeen, UK, September 2-5, 2016, Proceedings 17, pages 3–17. Springer, **2016**.

[37] Jack Harmer, Linus Gisslén, Jorge del Val, Henrik Holst, Joakim Bergdahl, Tom Olsson, Kristoffer Sjöö, and Magnus Nordin. Imitation learning with concurrent actions in 3d games, **2018**.

[38] Yunlong Song, Mats Steinweg, Elia Kaufmann, and Davide Scaramuzza. Autonomous drone racing with deep reinforcement learning. In 2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pages 1205–1212. IEEE, **2021**.

[39] Binyu Wang, Zhe Liu, Qingbiao Li, and Amanda Prorok. Mobile robot path planning in dynamic environments through globally guided reinforcement learning, **2020**.

[40] Yuhan Xie, Minghao Lu, Rui Peng, and Peng Lu. Learning agile flights through narrow gaps with varying angles using onboard sensing. IEEE Robotics and Automation Letters, **2023**.

[41] Jiaxu Xing, Leonard Bauersfeld, Yunlong Song, Chunwei Xing, and Davide Scaramuzza. Contrastive learning for enhancing robust scene transfer in vision-based agile flight. arXiv preprint arXiv:2309.09865, **2023**.

[42] Philipp Foehn, Elia Kaufmann, Angel Romero, Robert Penicka, Sihao Sun, Leonard Bauersfeld, Thomas Laengle, Giovanni Cioffi, Yunlong Song, Antonio Loquercio, and Davide Scaramuzza. Agilicious: Open-source and open-hardware agile quadrotor for vision-based flight. Science Robotics, 7(67), **2022**. ISSN 2470-9476. doi:10.1126/scirobotics.abl6259.

[43] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. arXiv preprint arXiv:1606.01540, **2016**.

[44] 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**.

[45] Sikang Liu, Michael Watterson, Sarah Tang, and Vijay Kumar. High speed navigation for quadrotors with limited onboard sensing. In 2016 IEEE International Conference on Robotics and Automation (ICRA), pages 1484–1491. **2016**. doi:10.1109/ICRA.2016.7487284.

[46] Michael Burri, Helen Oleynikova, Markus W. Achtelik, and Roland Siegwart. Real-time visual-inertial mapping, re-localization and planning onboard mavs

in unknown environments. In 2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pages 1872–1878. **2015**. doi:10.1109/IROS.2015.7353622.

[47] Jing Chen, Tianbo Liu, and Shaojie Shen. Online generation of collision-free trajectories for quadrotor flight in unknown cluttered environments. In 2016 IEEE International Conference on Robotics and Automation (ICRA), pages 1476–1483. **2016**. doi:10.1109/ICRA.2016.7487283.

[48] Brett T. Lopez and Jonathan P. How. Aggressive 3-d collision avoidance for high-speed navigation. In 2017 IEEE International Conference on Robotics and Automation (ICRA), pages 5759–5765. **2017**. doi:10.1109/ICRA.2017.7989677.

[49] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning, **2011**.

[50] Boyu Zhou, Fei Gao, Luqi Wang, Chuhao Liu, and Shaojie Shen. Robust and efficient quadrotor trajectory generation for fast autonomous flight, **2019**.

[51] Peter R. Florence, John Carter, and Russ Tedrake. Integrated perception and control at high speed: Evaluating collision avoidance maneuvers without maps. In Workshop on the Algorithmic Foundations of Robotics. **2016**.

[52] Antonio Loquercio, Elia Kaufmann, René Ranftl, Matthias Müller, Vladlen Koltun, and Davide Scaramuzza. Learning high-speed flight in the wild. Science Robotics, 6(59), **2021**. ISSN 2470-9476. doi:10.1126/scirobotics.abg5810.

[53] Jesus Tordesillas and Jonathan P How. FASTER: Fast and safe trajectory planner for navigation in unknown environments. IEEE Transactions on Robotics, **2021**.