# Multi-Agent Reinforcement Learning for Autonomous Robotics

by

First Lieutenant Caroline R. Vincent

B.S., United States Military Academy at West Point (2022)

Submitted to the System Design and Management Program
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE IN ENGINEERING AND MANAGEMENT

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2024

© 2024 First Lieutenant Caroline R. Vincent. All rights reserved.

The author hereby grants to MIT and Draper a nonexclusive, worldwide, irrevocable, royalty-free license to exercise any and all rights under copyright, including to reproduce, preserve, distribute and publicly display copies of the thesis, or release the thesis under an open-access license.

| | |
|---|---|
| Authored by: | First Lieutenant Caroline R. Vincent<br>Draper Scholar<br>August 9, 2024 |
| Certified by: | Dr. Michael Ricard<br>Laboratory Fellow, The Charles Stark Draper Laboratory, Thesis Supervisor |
| Certified by: | Professor Sertac Karaman<br>Professor of Aeronautics and Astronautics, Thesis Supervisor |
| Accepted by: | Joan Rubin<br>System Design & Management Program<br>Executive Director |

# Multi-Agent Reinforcement Learning for Autonomous Robotics

by

First Lieutenant Caroline R. Vincent

Submitted to the System Design and Management Program
on August 9, 2024 in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE IN ENGINEERING AND MANAGEMENT

## ABSTRACT

Technological advancements in autonomous robotics, including autonomous vehicles, have created new opportunities for innovative solutions to many everyday challenges. The impact of integrating robotic agents into real-world applications may be significantly enhanced by leveraging advancements in multi-agent autonomous systems. However, the coordination required in multi-agent systems demands complex motion planning to deconflict actions and prevent collisions of vehicles moving at increasingly high speeds. This thesis explores the application of multi-agent reinforcement learning (MARL) to autonomous robotics by teaching a central controller to navigate multiple agents across various environments without collisions. The simulated scenarios range from simple, obstacle-free environments to complex environments with obstacles configured to form narrow passageways or represent other complexities in dense urban environments. The findings demonstrate the potential of MARL to achieve high accuracy in navigating these different environments, highlighting the method's flexibility and adaptability across diverse settings and the resulting implications for applying MARL to real-world scenarios.

Thesis supervisor: Dr. Michael Ricard
Title: Laboratory Fellow, The Charles Stark Draper Laboratory

Thesis supervisor: Professor Sertac Karaman
Title: Professor of Aeronautics and Astronautics

# Acknowledgments

I'm grateful to have had so many brilliant mentors guiding me throughout this process.

First, I want to express my deepest appreciation to my advisors Dr. Michael Ricard, Dr. Begum Cannataro, Dr. Brian Wang, and Professor Sertac Karaman. Their expertise, encouragement, patience, and thoroughness have been invaluable to my research and personal development. It truly has been a privilege to work with and learn from them.

I'd also like to acknowledge Andrea Henshall, who has been an exceptional role model to me. Andrea was always willing to help me think through problems, debug, and offer her valuable insights despite her own busy schedule. Her kindness and mentorship have left a lasting impact on me.

I'm grateful to the MIT System Design and Management Program faculty and professors, especially Joan Rubin, for what has been an enriching academic experience and tremendous opportunity to attend MIT and be a part of such a remarkable cohort.

Thank you to Scott Crino, whose mentorship originally sparked my interest in the field of robotics. His guidance has been a constant source of inspiration throughout my research over the years.

Most importantly, I want to thank General Rich Morales for mentoring me every step along the way. I will forever be grateful for the countless hours he invested to advise, inspire, empower, challenge, and teach me.

Lastly, thank you to my family for their unwavering support and encouragement.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

Technological developments in the field of autonomous robotics continue to unlock new opportunities for novel solutions to ordinary problems. Specifically, multi-agent coordination of autonomous systems greatly scales the potential impact of integrating robotic agents into real-world scenarios. A major barrier to deploying such advancements remains the complex motion planning required for autonomous agents to avoid static obstacles and other simultaneously moving agents while moving at increasingly high speeds. Motion planners for multi-agent systems often rely on some form of predefined rules or fixed algorithms that can limit the system's adaptability and scalability in diverse environments. Such approaches frequently struggle to handle complex real-world scenarios with dynamic uncertainties. This work explores the application of multi-agent reinforcement learning to teach autonomous agents to complete navigation tasks without colliding with one another or other objects. The developed architecture incorporates flexible mechanisms enabling the model's potential application across a wide range of scenarios.

## 1.1 Background and Motivation

Various industries currently rely on automated robotic systems to carry out tasks previously completed by humans. For example, warehouses use robots to sort and package goods, doctors use medical robots to assist in surgery and perform precise movements, and households own robotic vacuum cleaners that move throughout the house [1], [2], [3]. Such autonomous systems limit human exposure to laborious, mundane, or dangerous tasks while increasing efficiency. Many of these examples incorporate easy solutions to avoid collisions with objects or humans: either 1) they are equipped with a simple object detection system that prohibits the robot from moving too close to obstacles, or 2) the robot is placed in or constrained to an area in which it is not at risk of collisions. However, the additional complexity associated with dynamic robots, especially in multi-agent systems, requires a far more sophisticated and robust approach to avoiding collisions.

### 1.1.1 Autonomous Mobility in Dynamic Environments

In the context of this work, a dynamic environment refers to conditions that unpredictably change over time; this could either mean moving obstacles - which could include other agents - or significant variations in terrain between system uses. The mobility of autonomous robotics in dynamic environments challenges autonomous systems, requiring adaptation to continuously unfolding scenarios. Common examples of such robots include autonomous vehicles such as self-driving cars and unmanned aerial vehicles (UAVs, or drones). The high speeds at which such vehicles operate and the dynamic environments they must navigate add complexity to the required motion planning. Compared to simple robots for unchanging automated tasks in largely static environments, these motion planners must rely heavily on more sophisticated decision-making algorithms - typically reliant on artificial intelligence - to complete tasks safely without the need for direct human intervention. Many autonomous vehicles are equipped with standard collision avoidance systems capable of avoiding static obstacles, but coordinating interactions for multiple agents proves to be a much more difficult task.

### 1.1.2 Multi-Agent Systems

Multi-agent systems refer to a group of agents that interact in some form within a given environment. There are two main types of multi-agent robotic systems: competitive and cooperative. Competitive multi-agents work against each other, such as in a two-person video game. Cooperative agents, on the other hand, work collaboratively or in harmony to accomplish their goals, such as a drone swarm.

Path planning for multi-agent autonomous systems is far more complex than planning for a singular agent. Cooperative multi-agent planning requires some knowledge or prediction of where the other agents plan to move next or some coordination between the agents regarding where each agent will move to avoid a collision. Furthermore, suppose the agents are fully autonomous and reacting to some unfolding environment. In that case, these decisions must be made in real-time with severe consequences for mistakes as the vehicles may move at high speeds with only short notice of other fast-approaching agents.

The complexity of motion planning for multi-agent systems exponentially increases when scaling the system for an increased number of agents. For example, if an agent has four possible actions to take at each moment, there are only four possibilities of what action may occur. However, if two agents each have four possible actions, sixteen different combinations of actions could take place at any given moment. Likewise, the number of actions available to an agent at each moment exponentially scales the total action combination possibilities in a multi-agent system, whereas a single-agent system increases linearly. The scalability of multi-agent systems is the primary reason why many traditional methods used for single-agent motion planning are insufficient in multi-agent scenarios. However, alternatives such as machine learning-based methods, specifically reinforcement learning, offer potentially far superior methods for handling the additional complexity present in multi-agent systems.

### 1.1.3 Multi-Agent Reinforcement Learning

Machine learning (ML) leverages data to make predictions and is categorized into supervised learning, unsupervised learning, and reinforcement learning (RL) [4]. The difference between unsupervised learning and supervised learning is data labeling; supervised learning has labeled data, and unsupervised learning does not. The focus of each method is typically myopic. RL deviates from such traditional ML methods by focusing instead on far-sighted sequential decision-making. RL's learning process mimics the process of human learning through trial and error, making RL an attractive method for training robots to more closely imitate human decision-making.

Multi-agent RL (MARL) refers to using RL techniques to train multiple agents within an environment to make decisions autonomously while accounting for the actions of other agents. Broadly, there are two types of training and execution: decentralized and centralized. The kind of training determines whether agents train independently or together. Likewise, the type of execution determines whether each agent makes individual autonomous decisions or if a centralized controller assigns actions to each agent. Decentralization offers greater autonomy and scalability, while centralization facilitates easier coordination and implementation.

Successful demonstrations of MARL show potential for multi-agent autonomous vehicle collaboration in achieving complex tasks in a way that is adjustable to various possible scenarios. Rather than being constrained to predefined coordination strategies, MARL allows agents to learn optimized policies for the specific task and environment in which the system operates. The emergent behavior from MARL systems can adapt to dynamic and uncertain environments, making it a robust approach to solving real-world problems where the application's conditions may be unpredictable.

## 1.2 Problem Statement and Objectives

Deploying multi-agent autonomous robotics within uncertain and dynamic environments relies on complex motion planning systems to avoid collisions among agents and obstacles. Traditional approaches that integrate fixed algorithms or model-based strategies limit the system's flexibility and scalability. RL offers a promising approach to multi-agent motion planning that allows for robust coordination strategies optimized for each specific task and environment in which they operate. This work applies an RL approach to multi-agent autonomous robotics by teaching a centralized controller to direct multiple agents across various scenarios to complete tasks without collisions. The experiments are conducted in simulation with the potential for many different real-world applications.

### 1.2.1 Aim of Thesis and Contributions

This thesis will demonstrate the value and limitations of applying RL to multi-agent autonomous robotic scenarios. The primary goal is to teach multiple agents within an environment to successfully navigate without colliding with other moving or static obstacles or agents. This research will highlight the advantages of MARL over more traditional mo-

tion planning techniques by applying a MARL algorithm to diverse environments, thereby showcasing its adaptability and robustness.

The work contributes to the field by demonstrating the ability of MARL to handle numerous complex environments with differing sizes, number of agents, and obstacle configurations. Furthermore, this work offers a unique capability for a central controller to direct agents from every possible starting location within an environment rather than merely solving one set of fixed paths. Lastly, this research showcases the flexibility of the proposed method to retrain and adapt to different scenarios, enhancing the potential applicability across diverse real-world situations.

# Chapter 2

# Literature Review

In robotics, motion planning refers to determining a viable path for a robotic agent free of collisions given the robot's position in an environment and the surrounding obstacles [5]. Multi-agent motion planning extends this concept to coordinate the movement of two or more agents, ensuring collision avoidance between agents and obstacles. Given the complexity of the field, much of the research on multi-agent motion planning focuses on a sub-problem known as multi-agent pathfinding, often accomplished by representing the problem on a graph. Rather than focusing on the dynamics of the robots, such as speed or mechanics, the emphasis shifts towards finding optimal paths for each agent.

The primary objective of pathfinding algorithms is to compute the optimal path between two points or nodes [6]. The performance metrics of such algorithms are typically based on the speed of search, the quality of the results, and the computational resources of pre-processing - metrics that may often work against each other and must be carefully balanced in order to achieve overall optimization.

Traditional planning methods for single-agent systems are well-established, such as Dijkstra or A* [7]. However, efforts to directly translate such approaches to multi-agent systems often face scalability challenges due to the additional degrees of complexity associated with coordinating multiple agents [8]. The development and recent popularity of deep learning and RL are enhancing research in motion planning [9]. Applying these tools as alternatives to conventional planning techniques for multi-agent pathfinding enables a more robust model that is capable of effectively handling dynamic scenarios.

A comprehensive exploration of the existing literature in this field provides insight into the limitations of traditional motion planning methods for multi-agent systems and demonstrates the potential advancements that may be achieved by employing a MARL approach instead. This chapter is organized to first compare traditional pathfinding methods with RL algorithms by providing an overview of each. Subsequently, it shares successful applications of deep RL for multi-agent pathfinding, showcasing the possibilities of such methods. Table 2.1 summarizes each method's pros and cons.

## 2.1 Traditional Pathfinding Methods

Traditional pathfinding methods such as graph search algorithms and sampling-based algorithms efficiently determine optimal paths for single-agent systems [9]. However, they may not be best suited for multi-agent systems due to the additional complexity inherent in multi-agent pathfinding.

### 2.1.1 Graph Search Algorithms

Graph search, or graph traversal, algorithms explore the vertices and edges of graphs to solve problems [10]. They are often used to search for an optimal path or to determine whether a path exists [9].

One of the most common examples of graph search is the pathfinding algorithm, Dijkstra [9]. Dijkstra solves the shortest path problem using edge weights on a graph, exploring every option from the starting point and updating the shortest distances as it goes on [11]. The algorithm maintains a priority queue of unprocessed vertices to explore until all options are examined and the optimal path is determined.

Researchers have demonstrated the success of Dijkstra's algorithm in robot path planning. For example, Wang et al. use Dijkstra to prove that a robot can solve a maze of barriers [12]. Likewise, Dhulkefl et al. leverage the Dijkstra algorithm for UAV path planning where the shortest path includes traversing from start point to end point while navigating static obstacles most efficiently [13].

Although Dijkstra's algorithm is recognized as one of the best pathfinding algorithms, it often requires significant computation time, making it challenging to meet the needs of real-time path planning [14]. Additionally, the classic version of Dijkstra focuses solely on the shortest path, meaning it often requires additional algorithms to consider other problem dimensions, such as dynamic obstacles. Most applications of Dijkstra involve some "improved" variation of the original algorithm, or it is combined with other methods to account for additional complexities. For example, a "relaxed" version of Dijkstra, created by Ammar et al., improves the scalability of Dijkstra to larger grid environments, which shows decreased computation time [15]. Researchers have verified this improved version in successful demonstrations of intelligent robot path planning [16]. However, the majority of applications remain single-agent systems.

Similarly, the A* (A-Star) algorithm derives from Dijkstra and introduces a heuristic function to estimate the cost of nodes that speeds up the search [7]. A* is considered an improvement of Dijkstra because it can find the shortest path more quickly than Dijkstra, providing superior performance in terms of runtime efficiency. However, A* also has limitations, such as an inability to avoid dynamic obstacles, including other agents, and having sensitive parameters, such as the accuracy of the heuristic function [7], [17].

Although pathfinding algorithms such as Dijkstra and A* exhibit strong performance in certain scenarios, they are ultimately not the best-suited solution for a multi-agent dynamic system due to their limited adaptability and capability to handle complex agent interactions relative to other algorithms [9]. While Dijkstra and A* are effective for solving shortest path problems on smaller scales, they are poor candidates for multi-agent dynamic environments

due to their inability to sufficiently account for agent interactions and inherent computational complexity, limiting their ability to scale to larger scenarios.

### 2.1.2 Sample-Based Algorithms

Sample-based approaches offer certain advantages over graph search methods when applied to high dimensional and complex problems [18]. Sample-based algorithms explore a given environment through random sampling to generate paths [9]. While the paths generated are often suboptimal, sample-based algorithms have the advantage of being able to generate motion plans more quickly because they are computationally efficient, making them a superior candidate for robotic motion planning methods [19].

One example of a sampling-based algorithm is rapidly-exploring random tree (RRT), which constructs and optimizes a tree-like structure of nodes using randomly sampled positions and constantly checks to determine whether the path of the new position is obstacle free [9], [19]. While the original version of RRT was limited in its ability to converge to an optimal solution, Karaman et al. proposed an improved approach, RRT*, that nearly guarantees convergence to an optimal solution by leveraging asymptotic optimality [19]. Under this more advanced approach, the algorithm converges as the number of iterations or samples approaches infinity, ensuring the solution quality improves over time.

Successful demonstrations of RRT include simulation path planning for unmanned surface vehicles [20]. Similarly, RRT was used to solve path planning for forming underwater gliders in ocean environments [21].

While sample-based methods such as RRT are computationally efficient for pathfinding, they are designed for single-agent systems. Sampling-based algorithms are susceptible to the curse of dimensionality [22]. They become computationally expensive as the number of agents to account for increases and are slow to react in addressing dynamic obstacles such as other agents [9]. The limitations of traditional pathfinding algorithms, including both graph search and sample-based approaches, highlight the need for comprehensive motion planning algorithms for dynamic environments, especially multi-agent environments.

## 2.2 Reinforcement Learning Approaches

Recent deep learning and RL developments have steered researchers away from conventional methods. This shift is motivated by RL's superior demonstrated performance in handling complexity, making RL a strong option specifically for autonomous vehicles [9]. Deep learning, like machine learning, is a subset of machine learning. Deep learning uses layers of neural networks to process input data and extract information [4]. This section discusses two major types of RL that, when combined with deep learning, show significant promise in robotic motion planning: optimal value RL and policy gradient RL.

RL solves motion planning by associating actions taken by an agent with rewards and penalties. For example, using RL for pathfinding involves assigning values to areas on a map. If an agent reaches an area identified as its goal location, it may receive a numerical reward, but if the agent crashes, the agent's score is penalized. Additionally, the agent may receive a penalty for every second it takes to reach the goal, encouraging the agent to take

the shortest path. The agent learns how to navigate an optimal path through trial and error by finding and taking actions that maximize the attainable rewards.

### 2.2.1 Optimal Value RL

An example of optimal value RL is a deep Q-learning network (DQN). In DQN, the agent constantly evaluates the potential future reward of any available action and then selects the action that maximizes the expected value. Bellman's equation serves as a fundamental component of RL, providing the formula to estimate the rewards of each action [4]. Bellman's equation calculates the value of an action by determining what the immediate reward of that action is plus the expected value of all future rewards in light of that first decision.

One of the most notable examples of the application of DQN is playing Atari games. Mnih et al. present a deep learning model that, without adjustments, plays seven Atari 2600 games at an expert level [23]. Such results show the adaptability of RL to excel in various use cases.

However, DQN can be very sensitive to parameters such as the learning rate, the discount factor of future rewards, the exploration rate, the batch size, and the network update frequency. Therefore, DQN can require careful tuning to achieve optimal performance, which often involves significant time and effort invested into experimenting with different settings in order to achieve the desired results.

### 2.2.2 Policy Gradient RL

Alternatively, policy gradient RL uses neural networks to generate a probability distribution known as a policy to select actions [9]. The policies are optimized to maximize expected return using gradient descent [24].

Proximal policy optimization (PPO) is a powerful type of policy gradient method proposed by researchers at OpenAI in 2017 and is currently considered state-of-the-art in RL [25]. PPO is generally less sensitive to parameters than DQN and other notable algorithms, making it a more robust and user-friendly approach for RL tasks.

As it has rapidly gained traction in recent years, researchers have found various applications in which PPO outperforms other common methods. For example, researchers demonstrated UAV path planning and obstacle avoidance using an enhancement-based PPO [26]. In this work, Huang et al. demonstrated how a PPO-based method could achieve UAV decision-making under various uncertainties, such as searching for targets while avoiding obstacles.

Overall, PPO currently leads the way in terms of performance for multi-agent RL algorithms. PPO is easy to implement, requires relatively little parameter tuning, and is generalizable to many potential applications. However, like most traditional methods, PPO was still designed for single-agent scenarios, so some extension is necessary for multi-agent applications if the system is not entirely centralized [27]. However, with such extension, researchers have demonstrated strong performance of PPO even in multi-agent scenarios.

### 2.2.3 RL for Multi-Agent Pathfinding

There is a substantial amount of recent research exploring applications of deep RL for multi-agent pathfinding problems. For example, in terms of DQN, Xu et al. used deep RL, specifically DQN, to solve multi-agent pathfinding in an intelligent warehouse scenario [28]. This work demonstrated successful generalization and performance even when the problem scale increased. Similarly, Yang et al. used DQN for multi-robot path planning for an unmanned warehouse dispatching system [29].

Additionally, Park et al. demonstrated an implementation of PPO for multi-agent pathfinding while avoiding dynamic obstacles [30]. Likewise, Zhan et al. improved PPO for application in a multiple-UAV application [27]. Finally, Zheng et al. used PPO to build an autonomous collision avoidance system for planning in a complex multi-ship environment and integrated the PPO system with other algorithms [31].

Such examples show the motivation of many recent researchers to apply deep RL to multi-agent pathfinding problems. Whereas conventional pathfinding methods fail to account for agent interaction, many RL algorithms are designed to handle such complexity and are thus better fits for multi-agent scenarios.

## 2.3 Conclusion

Deploying multi-agent autonomous robotics requires multi-agent path planning that deconflicts collisions within a given environment and among multiple agents. Due to the limitations of conventional single-agent pathfinding algorithms to account for agent interactions, especially in dynamic or uncertain environments with multiple agents, alternative methods are necessary for multi-agent systems. Integrating RL with more traditional methods has resulted in significant advancements in pathfinding for autonomous robotics. The emergence of RL techniques offers an adaptable solution to complex and dynamic environments, making it an attractive option for autonomous agents, specifically those with mobility in uncertain environments. In particular, PPO currently leads the field in terms of high-performing RL algorithms that are user-friendly and easily generalized. The current literature demonstrates that PPO is a promising technique for multi-agent pathfinding in various complex environments.

Table 2.1: Comparison of pathfinding approaches

| Algorithm | Type | Pros | Cons |
|---|---|---|---|
| Dijkstra | Traditional: Graph Search | • Solves shortest path.<br>• Demonstrated success in robot path planning. | • High computation time.<br>• Focuses solely on shortest path.<br>• Limited scalability to larger environments. |
| A* | Traditional: Graph Search | • Finds shortest path faster than Dijkstra.<br>• Performs well on runtime efficiency evaluation. | • Does not always handle obstacles well.<br>• Sensitive to parameters. |
| RRT* | Traditional: Model-based | • Nearly guarantees convergence to optimal solution.<br>• Relatively fast motion planning. | • Does not scale well to complex multi-agent problems. |
| DQN | RL: Optimal Value | • Capable of learning complex policies.<br>• Demonstrated success in various applications. | • Sensitivity to parameters.<br>• Requires careful tuning for optimal performance. |
| PPO | RL: Policy Gradient | • Less sensitive to parameters than DQN.<br>• Robust and user-friendly approach. | • Requires improvement/extension for multi-agent application. |

# Chapter 3

# Problem Formulation

This work applies an RL approach to various autonomous multi-agent scenarios within simulated environments to demonstrate the robustness of such an approach to real-world applications. The problem is formulated as follows: a central controller directs two or more agents to navigate a given environment, modeled through a two-dimensional grid space. The controller is tasked with assigning a sequence of actions to each agent, representing directions in which the agents can move, and leading the agents from their start locations to their goal destinations. This must be done while avoiding any collisions with obstacles or between agents. A collision occurs when an agent shares the same grid space at the same time as another agent or an obstacle. The controller learns an optimal policy of successfully directing both agents to their goals without collisions using an RL algorithm. The policy adjusts along the way in training based on feedback given to the controller through trial and error. This process is repeated and evaluated across various scenarios, each with unique grid arrangements and a differing number of agents, to demonstrate the robustness of the RL approach. This chapter discusses the custom environment built to simulate various use cases and describes the designed scenarios in detail.

## 3.1 Custom Environment

The scenarios for the training and testing simulations are designed using a custom environment built in OpenAI Gym - an open-source platform designed to advance research in RL [32]. OpenAI Gym offers diverse pre-built environments with varying ranges of environment complexity, allowing users to develop and test their preferred RL algorithms in simulated scenarios.

### 3.1.1 Pre-Built Environment Examples

An example of such an environment available in OpenAI Gym is the "Cart Pole" environment, as seen in Figure 3.1. The Cart Pole problem is one example of a typical starting point when learning RL. OpenAI's environment for the Cart Pole problem serves as a platform where researchers can practice RL methods on this problem.

In the Cart Pole environment, the agent is a cart moving along a frictionless track while

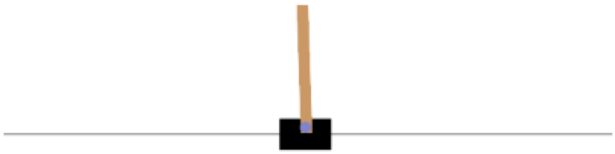


Figure 3.1: Cart Pole Environment. Source: OpenAI Gym [32].

Figure 3.2: Frozen Lake Environment. Source: OpenAI Gym [32].

balancing a pole [32]. The agent's objective is to continuously move left or right at the correct times to maintain the balance of the pole through the force of the cart's movement. If the pole falls, the episode terminates. The agent is rewarded for every second it maintains the balance of the pole, incentivizing the agent to take actions that lead to that desired behavior.

The agent has a discrete action space involving two possible actions: move left or move right. The observation space of this environment is known as a "box," which refers to an array of values denoting different aspects of the observation within relative upper and lower bounds [32]. In the example of the Cart Pole, the cart's position, the cart's velocity, the pole's angle, and the pole's angular velocity have a defined left and right limit. At any given moment, the observation provides an array of values representing each feature's current status.

Alternatively, an observation space can be discrete. In a discrete observation space, the agent can be located at n possible locations such as 0, 1, ..., n-1 [32]. An example of an OpenAI Gym environment with a discrete observation space is the "Frozen Lake" environment as seen in Figure 3.2.

In this example, the agent is an elf whose objective is to find a toy on a frozen lake while avoiding falling into any holes. This environment takes place on a 4x4 two-dimensional grid. There is a discrete observation space of sixteen unique locations the elf may reside at any

moment, representing each of the sixteen grid squares. The action space is discrete and contains four actions the elf can take: up, down, left, or right.

The elf is rewarded for reaching the toy, which occurs when it takes the correct sequence of actions to navigate from its starting grid square to the toy's grid square. The elf is penalized for falling into any holes along the way, which occurs if the elf moves into the same grid square as a hole. Therefore, the rewards and penalties incentivize the elf to navigate to the toy without falling into a hole.

### 3.1.2 Custom Environment Design

In addition to the pre-made environments available within OpenAI Gym, users can make their own custom environments to suit their specific research needs [32]. This may be done either by modifying existing environments or designing one from scratch using the available Gym functions and object classes.

Much like the Frozen Lake example, the custom environment built for this research consists of a two-dimensional grid. There is one "goal" location associated with each agent, similar to the toy in the Frozen Lake example. These goal locations remain fixed while the agents move.

Travel time is divided into discrete time intervals, or timesteps [33]. At each timestep, agents can take one of two types of actions: stay or move. "Stay" means the agent remains in the exact location for one timestep. "Move" means the agent moves from its current location (a grid space) to an adjacent grid space.

An episode consists of a series of timesteps in which agents take actions and receive corresponding rewards or penalties while navigating to their respective goals. A reward system is designed for each scenario, incentivizing or penalizing actions within the environment based on predefined criteria, such as penalties for collisions or rewards for reaching a goal.

The termination conditions are 1) the agents each find and remain in the grid space of their respective goals until all agents in the system reach their goal, at which point the episode ends, or 2) the episode times out after a predetermined number of timesteps that the agents are given to search for their goals. Collisions that occur do not cause the episode to terminate; if a collision occurs, the agents involved in the crash receive penalties but can continue their search for their goal.

During an episode, the actions, rewards, termination status, and an updated observation of each agent's location are collected after each timestep. The environment is reset if the specified termination conditions are met and a new episode begins. When an episode resets, the obtained rewards for each agent are set to zero, and the (x,y) coordinates of each agent are reset to their starting locations. In both training and testing, these starting locations are randomly selected.

Once a pre-specified number of training timesteps are run, the performance of the learned policy is evaluated over a series of predefined testing episodes. This is done by averaging the rewards earned over several episodes. The intended behavior in each scenario is for the agents to navigate to their respective goals without colliding with obstacles or each other. Thus, an optimal policy would result in rewards corresponding to this desired behavior.

### 3.1.3 Custom Environment Action Space

In each timestep, every agent has five available actions to select: move up, down, left, right, or stay. Agents cannot move diagonally across the grid. Therefore, the total number of possible action combinations a controller may select from to assign discrete actions to agents becomes $5^n$, where $n$ represents the number of agents.

Therefore, the custom environment incorporates a multi-discrete action space; each agent has a discrete action space - one of five actions - and the total actions assigned by the controller at any time will involve multiple of these discrete actions because there is one for each agent. For example, a sample of the action space could give [2 3], where the first agent is assigned action 2 (interpreted as moving one grid space down) and the second agent is assigned action 3 (interpreted as moving one grid space left). The interpretation of the action numbers is defined before training begins within the custom environment and remains the same through training and testing.

Additionally, the custom architecture of this environment specifies that any action that would cause an agent to move off the grid is treated as if the agent chose to stay. For example, if the agent starts in the top left corner and decides to move up, the agent remains in the exact grid location for the next timestep. This keeps all agents within the defined bounds of an environment space, similar to a drone reaching the perimeter of a forbidden airspace. If programmed safely, a drone will remain hovered in place if the pilot attempts to move the drone into a restricted airspace. To model such behavior, a decision to move off the grid is replaced with the action "stay." Given that there is a penalty for each timestep it takes to reach the goal, the controller learns that the action to stay by attempting to move off the grid is not part of the optimal solution.

### 3.1.4 Custom Environment Observation Space

An observation provides a snapshot of each agent's location within the environment. While the observation space of the custom environment is also multi-discrete given the distinct grid spaces that agents can occupy, the observation space is of "box" type. This is so that the lower and upper bounds can be defined based on the grid size, which can change depending on the scenario, unlike the number of actions available, which always remains 5 per agent.

The observation consists of each agent's x and y coordinates, represented as the agent's position relative to its goal. For instance, an observation sample of a scenario with two agents could give [-3 1 2 4], where the numbers correspond to [x1 y2 x2 y2]. Therefore, the first agent is three grid squares to the left of its goal and one grid square above it. Assuming no obstacles, the agent could reach its goal by taking three steps right and one step down. Likewise, the second agent is two steps to the right of its goal and four steps above it. The second agent could reach its goal by choosing to move left twice and down four times.

### 3.1.5 Custom Environment Reward System

The reward system values are determined through iterative testing during the system design phase. The reward system allocates +100 to each agent that reaches its goal. If the agent

departs from the goal before the episode terminates (when all agents reach their goal), the agent loses that 100 points. The agent can regain the points when returning to the goal.

Furthermore, each crash penalizes each agent by -35. Therefore, if two agents crash into each other, the overall system sees -70, but if the agent crashes into an obstacle, it is only -35.

Additionally, each agent's action, including the action to stay, incurs a -1 penalty to encourage time efficiency. The only way to stop incurring this time penalty is for the agent to reach and remain at its goal location. This encourages the agents to travel to their goal as quickly as possible and remain at their goal while waiting for the other agents in the system to reach their goals.

At the end of each timestep, each agent's rewards or penalties are summed to acquire an overall system reward for the central controller. That total reward is returned as feedback to use in the RL calculations to improve the policy.

### 3.1.6 Flexibility in Design

The custom environment's architectural design includes flexible mechanisms that allow the user to define any grid size at the beginning of training, which also determines the size of the observation space's upper and lower bounds. The user may also specify the number of agents and goals (one per agent). Lastly, the number and locations of obstacles may be altered for each individual scenario. After receiving these various inputs, the environment populates a two-dimensional grid world with the defined numbers and arrangement of agents and obstacles. The environment can be modified for different scenarios to rearrange the numbers or locations of obstacles and agents to test for various situations.

## 3.2 Scenarios

Using the custom environment, there are four scenarios to which this model is applied: an obstacle-free airspace, a narrow passageway, dense urban terrain, and a scaled grid and agents. Table 3.1 summarizes the key differences between each scenario. In training for each scenario, the agents' various starting locations vary so that the controller may explore all possible routes within that scenario and ensure the agents can traverse to their goal from any initial position. The goal locations remain the same throughout training and testing.

Table 3.1: Summary of scenarios

| Scenario | Grid Size | Agents | Obstacles | Maximum Rewards | Minimum Rewards |
|---|---|---|---|---|---|
| 1 | 7x7 | 2 | 0 | 176-198 | -144,000 |
| 2 | 10x10 | 2 | 54 | 169-192 | -144,000 |
| 3 | 10x10 | 2 | 34 | 167-197 | -144,000 |
| 4 | 15x15 | 3-5 | 0 | 250-497 | (-215,965) - (-360,000) |

### 3.2.1 Scenario 1

The first scenario simulates two UAVs traversing an obstacle-free airspace to return to their base after completing missions at differing locations, necessitating in some instances that their paths cross. The custom environment is formulated as two agents with randomized starting locations within a 7x7 grid without obstacles, as shown in Figure 3.3.

In application, the user could define the real-world size of each grid cell as the desired distance each UAV should maintain from one another while flying. For example, if the UAVs must remain at least 5 feet apart, then such a grid could be overlaid onto a map where the scale of each grid space represents a 5x5 foot area in the real world. Assuming that the agents travel to the center of a grid space when choosing a location to move to, this formulation assures that the agents will learn to remain at least 5 feet apart.

The destinations of each agent remain the same regardless of starting position. The intended behavior is for each agent to travel a path within an optimal time from any given starting point without colliding with the other agent. While the routes of the agents may eventually cross, they do not collide so long as they do not share the same grid square at the same timestep.

The total rewards per episode awarded to the overall system for learning the optimal behavior - each agent taking the most efficient path to its goal without collision - varies based on each agent's starting position since the agents are penalized for every step taken to reach their goal. The range of possible rewards for each scenario guides the analysis of how well the policy performs during simulation testing.

In this scenario, an agent's highest possible reward is 99. An agent receives 99 by receiving +100 for reaching its goal and losing -1 for taking one step to achieve it. Thus, the maximum reward per episode for two agents is 198.

The lower range the system could achieve while still behaving optimally is 176 overall - 88 per agent. This would transpire if each agent started six steps away both vertically and horizontally, as first visualized in Figure 3.3.

Alternatively, the lowest possible reward the system could receive for the worst learned policy is -144,000 per episode. This would occur if the agents started one step away from each other, chose to crash, and remained crashed for all 2,000 timesteps before the episode timed out and terminated. In this instance, each agent would receive a -35 penalty for crashing and a -1 timestep penalty for each of the 2,000 timesteps, for a total of -144,000.

### 3.2.2 Scenario 2

The second scenario simulates a situation where two autonomous vehicles come upon a narrow tunnel or passageway wide enough for only one agent to pass through at a time. The environment consists of two agents on a 10x10 grid with obstacles arranged to form an open trail or tunnel only one grid square wide, as shown in Figure 3.4. The agents can start anywhere within the narrow passage or on the side opposite of their goals.

The desired behavior is for the controller to decide and execute a way for the agents to traverse through the narrow passageway without colliding with one another or the surrounding walls while minimizing travel time. Depending on each agent's starting and ending positions, this may include choosing to have one agent "stay" for one timestep to allow the

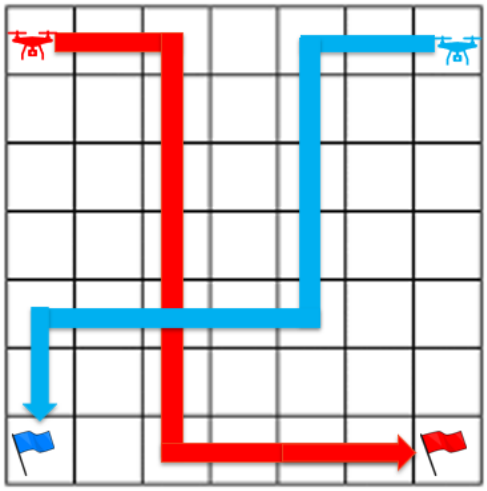


Figure 3.3: Scenario 1 - Two UAVs returning to their base

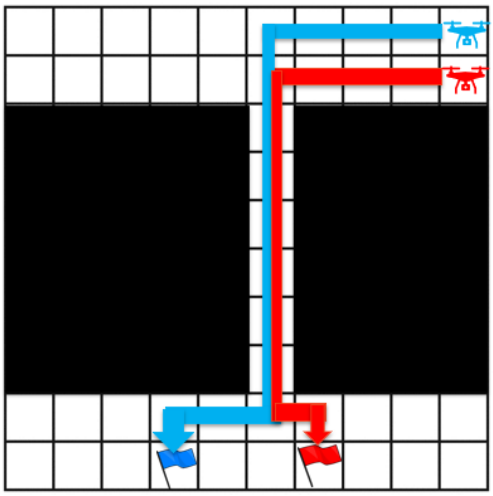


Figure 3.4: Scenario 2 - Two agents navigating a narrow passage

other to go first. The range of maximum rewards for optimal behavior in this scenario is 169-192.

### 3.2.3 Scenario 3

The third scenario replicates an environment in which two delivery robots must navigate a dense urban environment to reach their pre-specified locations where they are tasked to deliver their packages. The environment consists of two agents on a 10x10 grid with 34 obstacles, as shown in Figure 3.5. Similar to solving a maze, the agents must explore the environment in training for the controller to find the optimal routes for their tasks while avoiding collisions with one another or the terrain. The range of maximum rewards for optimal behavior in this scenario is 167-197.

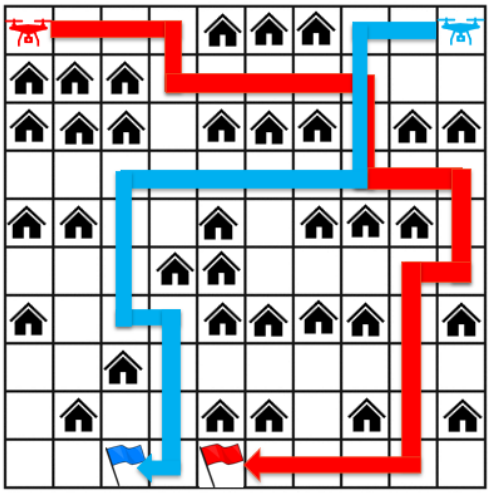


Figure 3.5: Scenario 3 - Two delivery robots navigating a dense urban environment

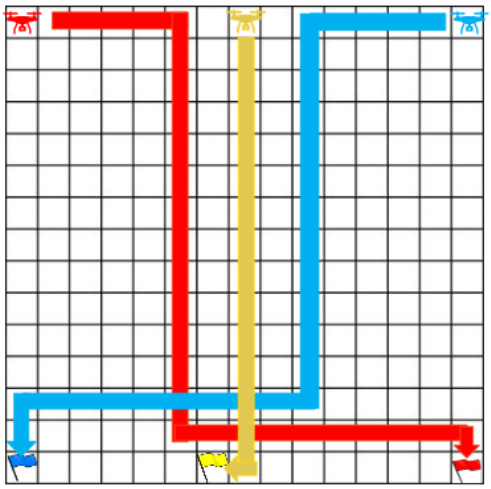


Figure 3.6: Scenario 4 - Three agents navigating to their base

### 3.2.4 Scenario 4

The final scenario involves a 15x15 grid without any obstacles. In each trial, every agent must travel to its own goal without colliding with other agents. The number of agents is varied between 3 and 5 to explore the scalability of the approach. Figure 3.6 provides an example including just three agents. The range of maximum rewards for optimal behavior in this scenario is 250-497 because it is dependent on the number of agents incorporated.

# Chapter 4

# Methodology

This chapter outlines the methodology for this research, which revolves around a centralized approach to control using the RL algorithm proximal policy optimization (PPO). An overview of RL and the PPO algorithm and a discussion of centralization are provided. The system architecture is discussed along with the specific RL techniques and details regarding the training and testing of the RL models. Table 4.1 provides a reference for certain terminology used throughout this chapter.

## 4.1 Reinforcement Learning (RL) Approach

According to [34], RL is defined as the "fundamental science of optimal decision-making." RL differs from other types of machine learning because it does not interact with datasets but with environments that depict real-world scenarios.

The basis of RL involves an agent interacting with an environment and receiving continuous feedback through rewards or penalties for its actions [34]. This cycle is visualized in Figure 4.1. The agent learns a behavior, known in RL as a *policy*, $\pi$, through trial and error by taking actions that maximize its rewards.

The agent's position within the environment is known as the *state*. At time $t$, the agent in state $s_t$ takes action $a_t$ to receive reward $R_{t+1}$ in state $s_{t+1}$ [34].

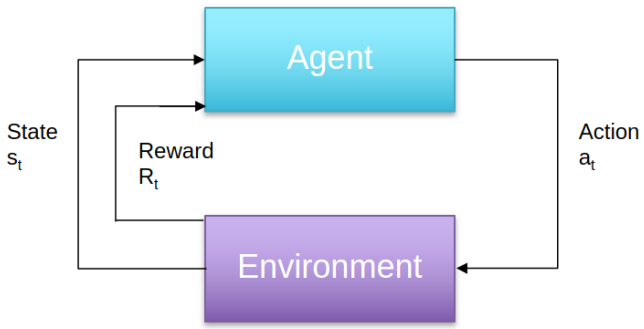



Figure 4.1: Reinforcement learning cycle

Table 4.1: Reinforcement learning terminology

| Terminology | Symbol | Description |
|---|---|---|
| Observation | - | Information perceived by the agent about the environment |
| State | $s$ | The current position of the agent relative to the environment |
| Action | $a$ | The decision or move that the agent makes at a particular state |
| Reward | $R$ | Feedback from the environment indicating the desirability of an action |
| Discount Factor | $\gamma$ | A value between 0 and 1 representing the scaled value loss of obtaining rewards in the future as opposed to the present |
| Policy | $\pi$ | A strategy that the agent employs to determine actions based on states |
| Callback Method | - | A function called at specific points during the training process to evaluate the policy |
| Warm Start | - | Starting the training process with pre-existing knowledge |

Gamma, $\gamma$, is the *discount factor* that determines the value of future rewards [34]. When gamma is closer to 0, the reward is discounted more, so reaching that reward later on decreases its value. If gamma is closer to 1, rewards maintain a relatively higher value even if they are not achieved until later. Therefore, the discounted reward received at time $t$ from taking action $a_t$ is represented as

$$\gamma^t R_{a_t}(s_t, s_{t+1}) \tag{4.1}$$

The accumulated rewards over all timesteps are given by

$$\sum_{t=0}^{\infty} \gamma^t R_{a_t}(s_t, s_{t+1}) \tag{4.2}$$

where $\gamma$ is a real number between 0 and 1 [35]. The optimal policy, $\pi$, is the strategy that selects the actions that maximize these cumulative rewards in Equation 4.2. Finding such a policy requires some form of policy search, such as policy gradient methods.

### 4.1.1 Policy Gradient Methods

A policy search in RL aims to find a policy $\pi$ that maximizes Equation 4.2 [36]. Policy gradient methods are one type of policy search algorithm that estimates the gradient of the cumulative rewards to iteratively improve the policy [37]. Table 4.2 summarizes the terms used within the following two subsections surrounding the topic of policy gradient methods.

Common gradient estimators take the form

$$\hat{g} = \mathbb{E}_t \left[ \nabla_\theta \log \pi_\theta(a_t|s_t) \hat{A}_t \right] \tag{4.3}$$

Table 4.2: Policy gradient terminology

| Terminology | Symbol | Description |
|---|---|---|
| Policy gradient methods | - | A subset of RL policy search algorithms that estimate the gradient of the expected cumulative rewards to improve the policy iteratively |
| Gradient Estimators | $\hat{g}$ | A mathematical technique used to approximate the rate of change of a function, such as the expected cumulative rewards in policy gradient methods |
| Advantage Function | $\hat{A}_t$ | A measure of how advantageous it is to select a specific action at a given state |
| Trust Region Policy Optimization (TRPO) | - | A policy optimization method that constrains the divergence between the old and new policies within a trust region |
| Trust Region | - | A defined area within which the policy can be updated without causing large changes |
| Probability Ratio | $r_t(\theta)$ | A ratio of the probability of selecting action $a_t$ given state $s_t$ under the new policy as compared to the old policy |
| KL Divergence | KL | A measure of the difference between the new and old policy after an update used in TRPO |
| Proximal Policy Optimization (PPO) | - | A type of RL policy gradient method that uses a clipping function to restrict policy updates |
| Clipping Function | $\mathcal{F}$ | A function used in PPO to directly cap the size of policy updates |
| Actor-Critic Architecture | - | An RL architecture with two networks: an actor network for action selection and a critic network for value estimation |

where $\hat{g}$ is the estimated gradient, $\mathbb{E}_t$ represents the average of expected rewards, $\hat{A}_t$ is an estimator of the advantage function which indicates the quality of an action given the current state, $\theta$ denotes the policy parameters such as the neural network weights and biases, and $\nabla_\theta \log \pi_\theta(a_t|s_t)$ is the gradient of the logarithm of the current policy $\pi_\theta(a_t|s_t)$ [25], [36].

Before the trust region policy optimization (TRPO) method - an RL technique preceding PPO - the primary challenge for policy gradient methods was finding the right step size to update the policy. TRPO addressed this by constraining the divergence between the old and new policies into a trust region. A trust region constrains the magnitude of policy updates within a defined region of the current policy to ensure a stable learning process.

The probability ratio, $r_t(\theta)$, is defined as

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)} \tag{4.4}$$

where $\pi_\theta$ is the current policy and $\pi_{\theta_{\text{old}}}$ is policy before the update [25], [36]. This gives the probability of selecting action $a_t$ given state $s_t$ under the new policy compared to the old one. TRPO uses the ratio to maximize the objective function of expected cumulative rewards obtained by a policy subject to a policy update size constraint, giving

$$L(\theta) = \hat{\mathbb{E}}_t \left[ \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)} \hat{A}_t \right] = \hat{\mathbb{E}}_t \left[ r_t(\theta) \hat{A}_t \right] \quad \text{subject to} \quad \hat{\mathbb{E}}_t \left[ \text{KL}[\pi_{\theta_{\text{old}}}(\cdot|s_t), \pi_\theta(\cdot|s_t)] \right] \leq \delta \tag{4.5}$$

where KL measures the divergence between the old and new policy, "$\cdot$" means any action, and $\delta$ represents the trust region size [25], [36]. Therefore, when $\delta$ is small, the constrained difference between the old and new policy is insubstantial [36]. Although TRPO can ensure consistent and stable policy improvement, the constraint imposed can be computationally inefficient.

### 4.1.2 Proximal Policy Optimization (PPO)

The specific type of RL algorithm used in this research is PPO. PPO offers an alternative to TRPO that is more computationally efficient through its use of a clipping function [36]. "Clipping" refers to the technique used in PPO to restrict policy updates. PPO learns a policy by iteratively optimizing a clipped surrogate objective function

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t \left[ \min \left( r_t(\theta) \hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon) \right) \hat{A}_t \right] \tag{4.6}$$

where the lower and upper clipping ranges are denoted by 1 - $\epsilon$ and 1 + $\epsilon$ [25], [38]. The hyper-parameter $\epsilon \in (0, 1)$ controls the range within which policy updates are clipped.

Much like the trust region in TRPO, the purpose of the clipping function in PPO is to constrain the magnitude of policy updates by clipping the probability ratio to prevent excessively large changes. The difference is that TRPO constrains the objective function through the KL divergence measure in Equation 4.5, whereas PPO uses "clipping" to directly impose a cap on the objective function, as seen in Equation 4.6.

PPO is more computationally efficient than TRPO because it uses only first-order optimization [25]. Additionally, PPO is more data efficient than TRPO, performing multiple

epochs of mini-batch optimization per data collection, as opposed to TRPO's single policy update per data cycle [25], [38].

To stabilize training, PPO uses an actor-critic architecture [36], which involves two networks: an actor and a critic, both typically represented as neural networks. In this architecture, a policy network (the actor) is responsible for selecting actions according to the current policy. A value network (the critic) estimates the value of the selected actions. The critic's evaluations refine the actor network iteratively and, thus, the policy. In PPO, the critic's estimation is used to calculate the advantage function, $A$, which is then used to inform the policy update, which is constrained to update within the bounds of the clipping range.

## 4.2 Centralized Learning

Using the PPO algorithm, this research adopts a centralized RL approach to control. Whereas some forms of traditional Multi-Agent Reinforcement Learning (MARL) may refer to agents that learn independently and communicate with one another to coordinate actions, this research introduces a "meta agent," which refers to a central controller that oversees agents' actions within the environment. Although the algorithm solves multi-agent scenarios, the central controller learns to coordinate the actions of each agent, ensuring conflict-free interactions and task completion. This approach transforms the problem into a complex combinatorial problem solved by RL rather than classic versions of MARL that may use decentralized learning or execution.

At each timestep, the controller receives information about the location of each agent and then, based on that information, assigns the following action for each agent. Each agent then takes the assigned action and receives the corresponding reward. The actions' outcomes are centrally processed by summing all rewards received in each timestep by all agents. Therefore, performance is measured relative to the system as a whole by evaluating the controller's policy based on how all agents are collectively performing. Leveraging a centralized process is advantageous because it ensures shared information among all agents in the system, minimizing the need for communication between agents and improving decision-making efficiency.

## 4.3 Simulation

The simulation for both training and testing is carried out using Stable Baselines3, a software library containing RL algorithms, including PPO, compatible with OpenAI Gym [39]. This work uses Stable Baselines to create and train models. After the model is trained, the best policy is identified, saved, and evaluated.

Table 4.3 presents two PPO model parameters built in Stable Baselines, which are explored for model tuning throughout this work. For a comprehensive list of all parameters provided by Stable Baselines, refer to Table A.1 in the appendix.

Table 4.3: Stable Baselines PPO model parameters

| Parameter | Description | Default Value |
|---|---|---|
| gamma | Discount factor for future rewards | 0.99 |
| total_timesteps | The total number of timesteps the model trains for | NA |

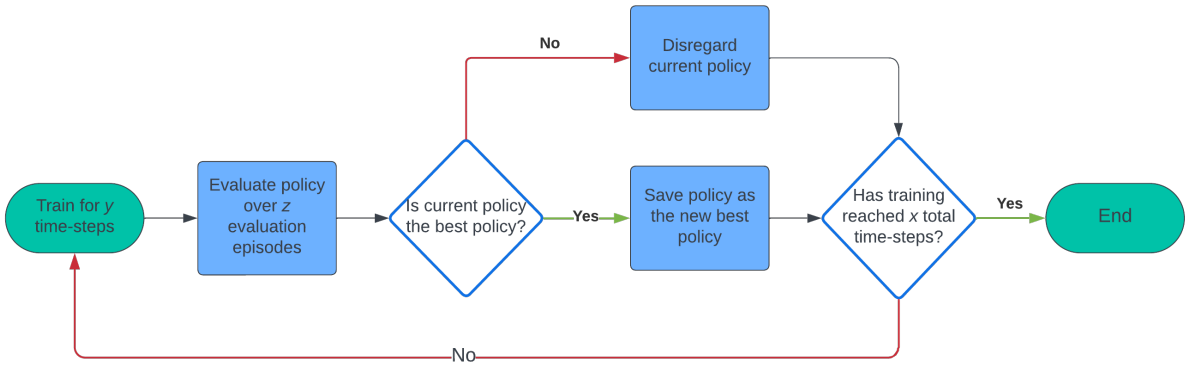



Figure 4.2: Callback function process

### 4.3.1 Training Process Enhancement: Callback Method

This work leverages two main techniques to enhance the training process: a callback method and warm starts. First, Stable Baselines' evaluation callback method is used to customize the policy evaluation during training [39]. The importance of a callback method comes from PPO's tendency to converge and diverge repeatedly. Therefore, this evaluation callback method monitors the controller's performance during training to ensure that the best policy discovered throughout training is saved and selected for subsequent training or testing.

The callback function works as follows: after a certain amount of timesteps, training is temporarily paused to evaluate the current policy the model has learned thus far. The performance is evaluated by averaging the overall system's obtained rewards over a specified number of episodes. If the performance of the policy is higher than any previous recorded performance, the current policy is saved as the new best policy. If the performance is not higher than the best performance thus far in training, that policy is ignored, and the previous benchmark remains.

A flowchart illustrating this process is shown in Figure 4.2, where $x$ represents the total timesteps the model trains for, $y$ is the number of timesteps the model trains for in between each time the callback function is called, and $z$ refers to the number of episodes over which the current policy is evaluated during the callback function. The model performance is tested at the end of this process using the best policy found during the callback evaluations.

Table 4.4 presents the parameters used for the evaluation callback. The evaluation frequency represents how often the callback method is called during training. For example, a parameter of 250 means the policy is evaluated after every 250 training timesteps. The number of evaluation episodes refers to the number of episodes over which the policy is

evaluated.

Table 4.4: Stable Baselines evaluation callback parameters

| **Parameter** | **Description** |
|---|---|
| eval_freq | How many timesteps the model trains for in between policy evaluations |
| n_eval_episodes | How many episodes over which the policy is evaluated |

The number of evaluation episodes is dependent on many factors. First, it depends on whether a full or partial evaluation will be run. Since there is a trade-off between training time and performance, this work tests both full and partial evaluations. Second, the number of possible unique episodes differs per scenario. For instance, in a 10x10 grid with two agents without obstacles, there are 9,506 unique combinations of starting locations. This number is derived from the following equations:

$$(\#\text{ of vertical grid spaces}) \times (\#\text{ of horizontal grid spaces}) = \text{total grid spaces} \tag{4.7}$$

$$(\text{total grid spaces}) - (\#\text{ of goal locations}) - (\#\text{ of obstacles}) = \text{total potential locations} \tag{4.8}$$

$$(\text{total potential locations}) \times (\text{total potential locations} - 1) = \#\text{ of unique combinations} \tag{4.9}$$

where an agent cannot start on the same grid cell as another agent or a grid cell containing an obstacle or goal. Therefore, when incorporating the values from the 10x10 grid with two agents (each with one goal) and without obstacles into Equations 4.7 and 4.8, Equation 4.9 gives 9,506 total unique combinations.

Using Equations 4.7, 4.8, and 4.9, Table 4.5 presents the number of unique starting combinations of each of the scenarios described in Section 3.2 in the Full Evaluation column. An important exception to note is the number of combinations for a full evaluation of Scenario 2; for this scenario, the starting locations of the agents must be within or above the narrow passage on the grid, limiting the number of unique combinations. This is because Scenario 1 already covers learning to navigate an environment without obstacles, so there is no value in showcasing how the agents can find their goals when starting below the narrow passage. Likewise, in Scenario 3, the agents can only start in rows 2-10 since row 1 contains only goal locations and zero obstacles.

Additionally, the last column gives the value of one-fourth of the total number of unique combinations per scenario. These values will be used when conducting a partial evaluation during the callback function rather than using every possible combination. The exception for this is Scenario 4, which uses 100 fixed evaluation episodes, given the exponential increase in complexity that each additional agent brings.

In a full evaluation, the parameter n_eval_episodes from Table 4.4 will equal the value in the Unique Combinations column. This means that the current policy will be evaluated over every possible combination of starting locations when the callback function is called. In a partial evaluation, n_eval_episodes will equal the value in the Partial Evaluation column, meaning that the policy will be evaluated over the number of partial evaluation combinations.

Table 4.5: Unique combinations per scenario

| Scenario | Grid Size | Agents | Obstacles | Full Eval | Partial Eval (25%) |
|---|---|---|---|---|---|
| 1 | 7x7 | 2 | 0 | 2,162 | 540 |
| 2 | 10x10 | 2 | 54 | 650 | 163 |
| 3 | 10x10 | 2 | 34 | 3,080 | 770 |
| 4 | 15x15 | 3-5 | 0 | 48,180 - 49,062 | *100 |

The evaluation callback occurs within an environment separate from the training. This is important because, in the training environment, the starting locations of the agents are randomized at the reset of each episode using a discrete uniform distribution. As a result, the controller learns to direct the agents from any potential starting points. However, randomized starting points should not be used when evaluating policy performance because a time-based reward penalizes each agent for each step it takes to reach the goal. For example, if one of the agents starts one step away from the goal in one episode but takes three steps to travel there, it will lose three points. It will lose five points if it starts five steps away in the next episode and travels directly to the goal. Even though the agent behaves optimally in the second scenario, the model will evaluate the first scenario as the better policy because it achieved higher rewards than in the second scenario. Therefore, the evaluation episodes must be standardized to compare identical episode resets and thus require a specialized evaluation environment.

Standardizing the evaluation episodes used in the callback function is most important in a partial evaluation because it ensures that the same set of evaluation episodes are used each time the policy is evaluated rather than selecting the episodes at random each time. Generating the episodes used in a partial evaluation involves a "while" loop that, while the list does not exceed 25% of the total unique combinations for that scenario, randomly selects starting locations using a discrete uniform distribution, checks to ensure the coordinates do not conflict which each other or obstacles and, assuming there is no conflict, appends that combination of starting locations to the list.

Standardizing fixed evaluation episodes also ensures that all unique combinations are evaluated during a comprehensive evaluation rather than selecting a number of random episodes to fit the number of total combinations, which would leave to chance that some combinations remain unused while others are used multiple times. Generating an exhaustive list for all unique combinations of starting locations involves iterating over each free x and y location for each agent.

### 4.3.2 Training Process Enhancement: Warm Starts

In addition to a callback method, iterations of "warm starts" are used to enhance training. For example, the agents are first trained in an obstacle-free environment before training multiple agents to navigate an obstacle-filled environment. The agents learn how to navigate to their goals without colliding with one another. Once the controller masters this task, obstacles are slowly introduced in stages. After mastering each stage, the policy is saved and loaded into the new, more complex environment. Breaking the complexity of training

into such modular chunks enhances the overall system performance by allowing the agents to learn one task at a time. These warm starts improve training efficiency by decreasing the total training time needed to learn comprehensive scenarios.

# Chapter 5

# Results

The results showcase that PPO performs well across a variety of scenarios. The algorithm successfully adapted with minimal parameter tuning to solve all given environments containing up to three agents with generally high accuracy levels. The best policies across each of these scenarios range from 93-100% accuracy.

Integrating several RL techniques, such as callback methods and warm starts, dramatically enhances the training process, leading to better performance in less time. Results with and without such enhancements are provided.

Limitations to this approach are also found, such as an inability for the method to converge to 100% accuracy in Scenarios 2, 3, and 4. However, a policy can achieve perfect accuracy after moving Scenario 2 onto a smaller environment, referred to as Scenario 2B. Additionally, the environments within Scenario 4 containing more than three agents cannot reach successful policies.

## 5.1 Scenario 1: Two UAVs in an Obstacle-Free Airspace

Scenario 1 represents the environment of two agents navigating an obstacle-free 7x7 grid, as first illustrated in Figure 3.3. Seven distinct policies are trained with unique parameter settings to illustrate the impact of such features on overall performance. A baseline policy, Policy A, demonstrates the ability of such an approach - PPO with additional training enhancements - to solve the environment with high accuracy quickly. Policies B through G showcase the impacts on performance by adjusting aspects of the baseline method. Table 5.1 summarizes the parameter changes of Policies A through G and the resulting performance.

### 5.1.1 Baseline Policy - Policy A

Policy A uses a callback method with a partial evaluation containing 540 evaluation episodes - 25% of the total 2,162 unique combinations - as calculated in Table 4.5. The parameter eval_freq, as described in Table 4.4, is set to 2,000, indicating that 2,000 training steps take place between each evaluation callback.

Policy A is trained for 2,000,000 total timesteps, which took 2.28 hours. An optimal policy satisfying all 540 specified evaluation scenarios was found without any alterations to

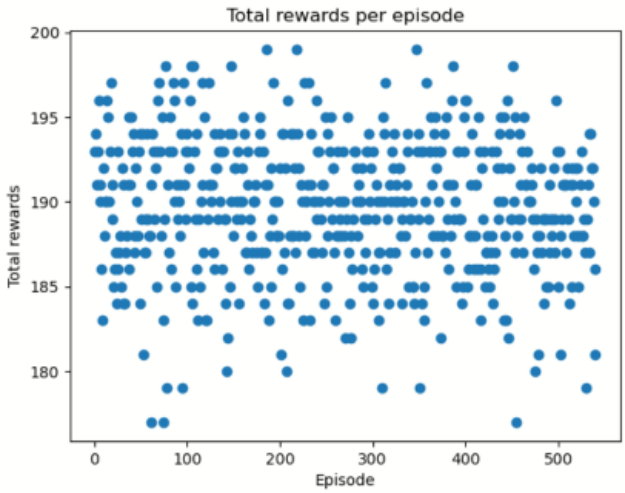



Figure 5.1: Scenario 1 Policy A performance results over 540 fixed evaluation episodes

the default parameters of the Stable Baselines PPO model. Policy A generalizes beyond the 540 specified evaluation episodes with 99.8% accuracy.

Figure 5.1 shows the results of the 540 evaluation episodes included in the partial evaluation of the callback method. These are all episodes with a unique combination of the two agents' starting locations. These combinations are not randomized each evaluation period, meaning the same 540 instances are used each time the callback function is called and the current policy is evaluated.

The slight variation in attainable rewards is explained by the difference in starting locations since the agents are penalized for every step they take to reach the goal. The range of maximum rewards for optimal behavior is 176-198, as provided in Table 3.1. Given these bounds, Figure 5.1 shows that the controller's policy directs the agents to their goals without collisions with 100% accuracy because the total rewards per episode all fall between 176 and 198.

Testing Policy A beyond the 540 fixed evaluation episodes used in training involves 1,000 randomly reset episodes. The random reset uses the same random generation used to reset non-evaluative episodes in training. In these instances, the goal locations still remain fixed. However, the starting locations of each agent are randomized when reset at the beginning of each episode to test the generalization of the policy. Figure 5.2 provides the results of 1,000 iterations of randomized testing episodes.

Any episode reward falling below 176 indicates a crash between agents (-35 per agent per crash) or that a non-optimal route was taken. Therefore, as seen in Figure 5.2, the policy learned to direct the agents to navigate an optimal route, regardless of randomized starting points, without any collisions among agents, over 998 of the 1,000 total episodes. In the two instances of crashes, the agents still made it to their goals and received their reward but were also penalized for their crash, which is why the reward for those two episodes totaled around 122 (+100 each for the goal, -35 each for the crash, -4 each for the steps taken).

This generalization gives an accuracy of 99.8% as opposed to the 100% accuracy of the partial evaluation episodes. The drop in accuracy is expected as the policy chosen at the end to use in testing is the one that performed the best over those specific 540 episodes.

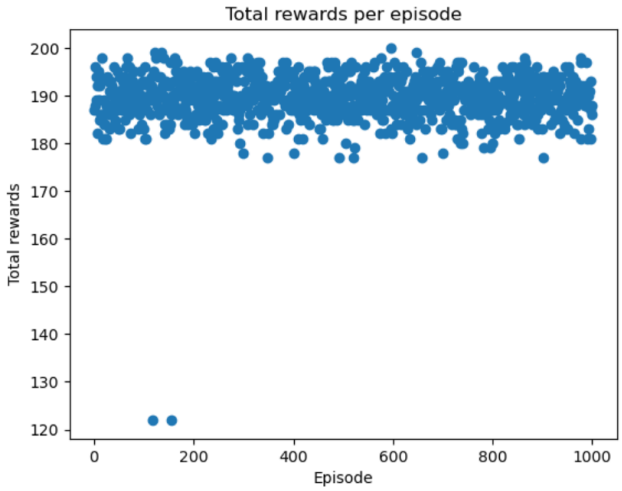



Figure 5.2: Scenario 1 Policy A performance results over 1,000 randomized testing episodes

Therefore, the evaluation episodes should see perfect performance, whereas the randomized episodes may contain imperfections.

It would be possible to achieve 100% accuracy if every possible unique combination out of the 2,162 possible were used in the callback evaluation. However, that would significantly increase the training time. Therefore, there is a trade-off between training time and performance.

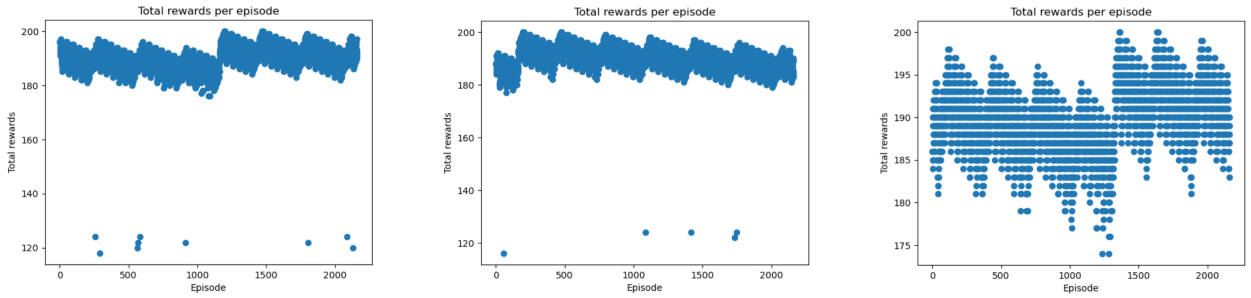
### 5.1.2 Full Evaluation with Increased Training Time - Policies B, C, and D

Policies B, C, and D train with the same parameters but different training strategies to highlight a trade-off between accuracy and training time. The differences in achieved accuracy are compared to the performance given more training versus less training.

First, a new policy, Policy B, was run with all of the same parameters, except the evaluation callback now included all 2,162 possible starting location combinations rather than the previous 540 episodes. This comprehensive evaluation indicates that every 2,000 episodes, as specified in eval_freq, the current policy is evaluated over 2,162 episodes, meaning the training will take much longer. Learning an optimal strategy is much more complex because it requires a policy that solves all 2,162 unique episodes.

Policy B trained for 2,000,000 timesteps again, but this time, the training took 5.66 hours rather than 2.28 hours. Figure 5.3a shows the policy performance over 1,000 randomized testing episodes. As seen in the graph, the policy did not yet achieve 100% accuracy, even though the policy was supposed to account for every possible starting combination. Instead, the policy achieved 99.1% accuracy. This drop in performance is because it is far more complex to train a policy capable of solving 2,162 variations rather than 540. Therefore, reaching 100% accuracy requires more training.

The graph's cyclical nature is explained by how the list of starting locations was generated. An iterative process was used to exhaust each agent's 2,162 unique combinations




(a) Policy B - 2M training timesteps

(b) Policy C - 4M training timesteps

(c) Policy D - 6M training timesteps

Figure 5.3: Scenario 1 Policies B, C, and D performance results for a comprehensive evaluation with differing training lengths

of (x,y) locations. Therefore, as each agent's x or y location increased, it incrementally increased or decreased the distance from the goal, incrementally shifting the maximum possible rewards for optimal behavior.

Similarly, the same process was repeated using a comprehensive evaluation of every unique combination for the callback method, but this time, the policy (Policy C) trained for 4,000,000 timesteps. This training took 15.16 hours. However, the policy still did not achieve 100% accuracy, as shown in Figure 5.3b. Instead, there were five crashes, giving an accuracy of 99.5%.

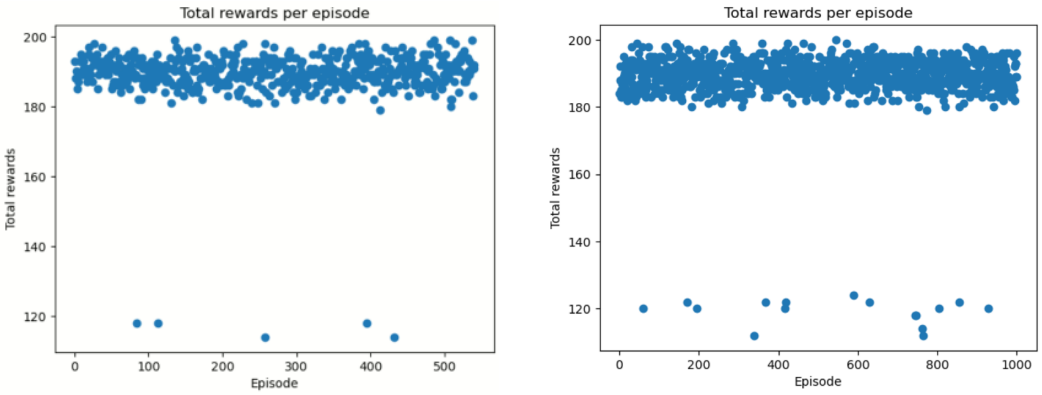
Finally, training the same callback method of a complete evaluation for 6,000,000 timesteps reached 100% accuracy, as shown in Figure 5.3c with Policy D. This training time took 20.13 hours. This policy ensures optimal behavior over every possible combination of agent starting locations.

### 5.1.3 Partial Evaluation with Decreased Training Time - Policy E

To illustrate the impact on the accuracy of decreasing training time, Policy E was run using the same baseline parameters with a partial evaluation of 540 episodes. However, this time, the policy only trained for 500,000 timesteps over 0.70 hours rather than 2,000,000. Unlike Policy A, Policy E did not perfect the 540 evaluation scenarios. As shown in Figure 5.4a, the performance included five episodes with crashes over the 540 episodes. Furthermore, when generalizing to 1,000 randomized testing episodes, Figure 5.4b shows that the performance is lower than Policy A, reaching an accuracy of only 98.4%. The results of this policy are still relatively good, but it does include 16 crashes. However, the training time was significantly less than Policy A, showcasing the trade-off between training time and performance.

### 5.1.4 Using Randomized Evaluation Episodes - Policy F

Another comparison for Scenario 1 compares the use of randomized evaluation episodes in the callback method instead of fixed evaluation episodes as done previously. As described in Section 4.3.1, using fixed scenarios is essential given the context of the reward system because different starting points give different maximum or minimum rewards. If randomized




(a) 540 fixed evaluation episodes (b) 1,000 randomized testing episodes

Figure 5.4: Scenario 1 Policy E performance results

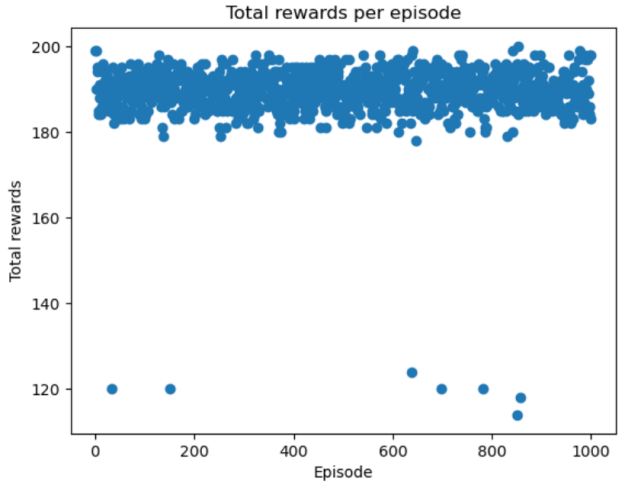



Figure 5.5: Scenario 1 Policy F performance results over 1,000 randomized testing episodes

episodes are used during the callback, the agents could get a "lucky" reset that gives them a higher attainable reward, such as starting right next to the goal. Even if they take a few wrong steps, it could still be a higher reward than if the agents start at the farthest possible locations and behave optimally. If the two situations were compared, it is possible that the policy for the favorable starting point would be selected as the best policy when the behavior was sub-optimal.

To analyze such a phenomenon, Policy F is trained using 540 again episodes for a partial evaluation. However, the 540 episodes in this training do not remain fixed between evaluation periods.

The train time for 2,000,000 timesteps took 2.15 hours. The performance of Policy F over 1,000 randomized testing episodes is shown in Figure 5.5. There were seven crashes, giving an accuracy of 99.3%, which is lower than the baseline (Policy A), indicating the importance of fixing the evaluation episodes used in the callback method.


Figure 5.6: Scenario 1 Policy G - mean reward through training

### 5.1.5 Training without a Callback Method - Policy G

The last comparison for Scenario 1 compares policy performance without a callback method. As described in Section 4.3.1, the advantage of a callback method is that it captures the most successful policy found throughout training rather than using whatever is most recent at the conclusion of training, which could lead to selecting a sub-optimal policy.

To demonstrate this, Policy G was trained, but no callback method was used. Therefore, the training of 2,000,000 timesteps only took 0.65 hours because the policy did not stop to evaluate every 2,000 training steps over 540 or 2,162 episodes as done in previous training.

Figure 5.6 shows the policy over time throughout training. The graph shows the repeated divergence, as previously mentioned. Additionally, the graph shows that the last recorded policy gives a value of around 186 rather than the highest value recorded, around 189. Since these values are averaged over several episodes, three reward points could indicate several additional collisions.

Only randomized episodes are used to test this policy since no specific evaluation episodes are tied to it. Figure 5.7 shows the performance of 1,000 iterations of randomized testing episodes. There were 982 successful episodes where the agents reached their goals without crashing, giving an accuracy of 98.2%. There were two episodes involving crashing. The rest of the unsuccessful episodes have rewards in the negative thousands, meaning that the agents either stayed in place or searched the grid without finding their goals until the episodes eventually timed out, incurring the maximum time penalties. Therefore, although the accuracy is similar to other policies, the unsuccessful episodes were far worse than those of other policies.

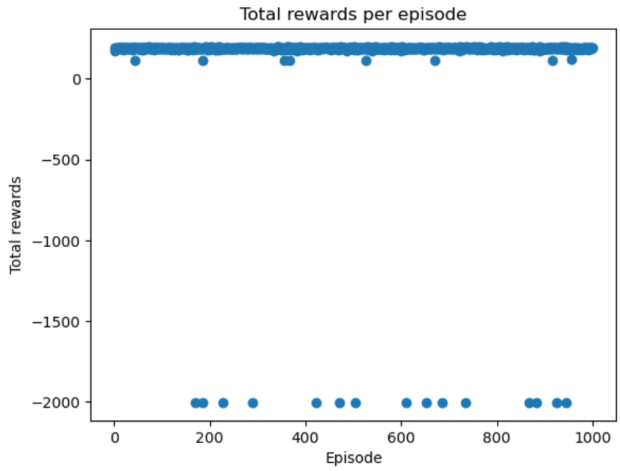



Figure 5.7: Scenario 1 Policy G performance results over 1,000 randomized testing episodes

Table 5.1: Summary of Scenario 1 policy comparisons

| Policy | Eval. episodes | Timesteps | Callback | Eval. type | Time (hrs) | Accuracy (%) |
|---|---|---|---|---|---|---|
| A | 540 | 2M | Yes | Fixed | 2.28 | 99.8 |
| B | 2,162 | 2M | Yes | Fixed | 5.66 | 99.1 |
| C | 2,162 | 4M | Yes | Fixed | 15.16 | 99.5 |
| D | 2,162 | 6M | Yes | Fixed | 20.13 | 100 |
| E | 540 | 500k | Yes | Fixed | 0.70 | 98.4 |
| F | 540 | 2M | Yes | Random | 2.15 | 99.3 |
| G | N/A | 2M | No | N/A | 0.65 | 98.2 |

## 5.2 Scenario 2: Two Vehicles in a Narrow Passage on a 10x10 Grid

Scenario 2 involves two agents traversing through a narrow passageway on a 10x10 grid, as illustrated in Figure 3.4. A significant difference from Scenario 1 is that the baseline policy for Scenario 2 incorporates a warm start. As explained in Section 4.3.2, a warm start enhances training by iteratively increasing complexity; in this case, the policy first trains in an obstacle-free environment before incorporating obstacles.

It is important to note that the successful policies learned for Scenario 1 cannot be used as a warm start for Scenario 2. Even though the first scenario encompassed two agents and an obstacle-free grid, the different grid dimensions prohibit the policy transfer. In order to import a previous policy into a new environment, the observation space of each environment must be the same size.

A limitation to this work is found through multiple trials with varied parameters. The policy with partial evaluation training does not reach 100% accuracy over fixed evaluation episodes, regardless of the length of training time. Likewise, the policy training with a full evaluation does not converge to 100% accuracy over randomized evaluation episodes, even after training for up to 40,000,000 timesteps over 42.60 hours.


Figure 5.8: Scenario 2 Policy A performance results over 163 fixed evaluation episodes

### 5.2.1 Baseline Policy

Like Scenario 1, the baseline policy, Policy A, trains using a partial evaluation for the callback function with 2,000 training timesteps occurring between each callback. As calculated in Table 4.5, a partial evaluation for Scenario 2 involves 163 starting combinations - 25% of the 650 total unique combinations. Even though the grid is larger than Scenario 1, Scenario 2 incorporates obstacles. Since agents cannot start in a grid space with an obstacle, there are fewer unique combinations where agents can start at the reset of an episode.

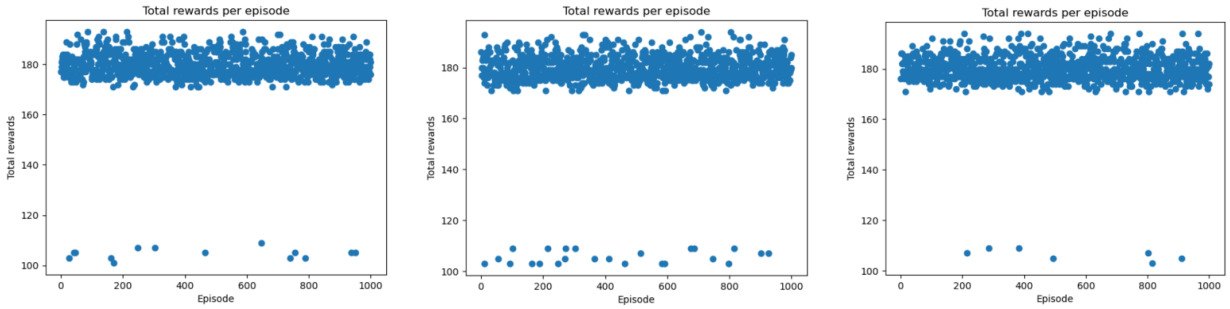
Policy A began as the pre-trained policy for a 10x10 obstacle-free grid with two agents so that the agents could first learn how to navigate around one another and avoid collisions before considering obstacles. The policy trained for 7,000,000 timesteps, taking 16.52 hours. This policy achieved 98.7% accuracy in the pre-training for the two agents reaching their goal locations without collisions.

Next, Policy A continued training but now with the obstacles that form a narrow passage. The training involved a partial callback evaluation for 2,000,000 timesteps, which took 1.08 hours. However, as shown in Figure 5.8, a policy capable of solving all 163 fixed evaluation episodes was not found. Instead, five episodes contained collisions.

This process was repeated with the warmed policy continuing to train for 4, 6, and 8 million timesteps. However, a more successful policy than Policy A was not found. Additional trials varied the discount factor, gamma, and the agents' reward for collisions. However, these changes did not improve the policy's performance, indicating a limitation of the approach not yet found in Scenario 1.

### 5.2.2 Full Evaluation

Similar to the partial evaluation results, training a policy with a comprehensive callback evaluation to achieve a higher accuracy was unsuccessful. Using the same warm start at the partial evaluation, Policy B trained using a comprehensive evaluation callback for many variations of timesteps ranging from 4 to 40 million, taking between 3.62 and 42.60 hours.




(a) 4 million additional training timesteps

(b) 12 million additional training timesteps

(c) 40 million additional training timesteps

Figure 5.9: Scenario 2 Policy B performance results over 1,000 randomized testing episodes after a varying number of continued training timesteps following a pre-trained warms start

Three examples of results are shown in Figure 5.9, with Figure 5.9c showing the best results the policy converged to. The non-linear increase in performance with increased training is attributed to the inherent randomness in training, such as starting locations upon episode resets. Although an accuracy as high as 99.3% is achieved, the policy did not ever converge to 100% accuracy, even after 40,000,000 training timesteps over 42.60 hours.

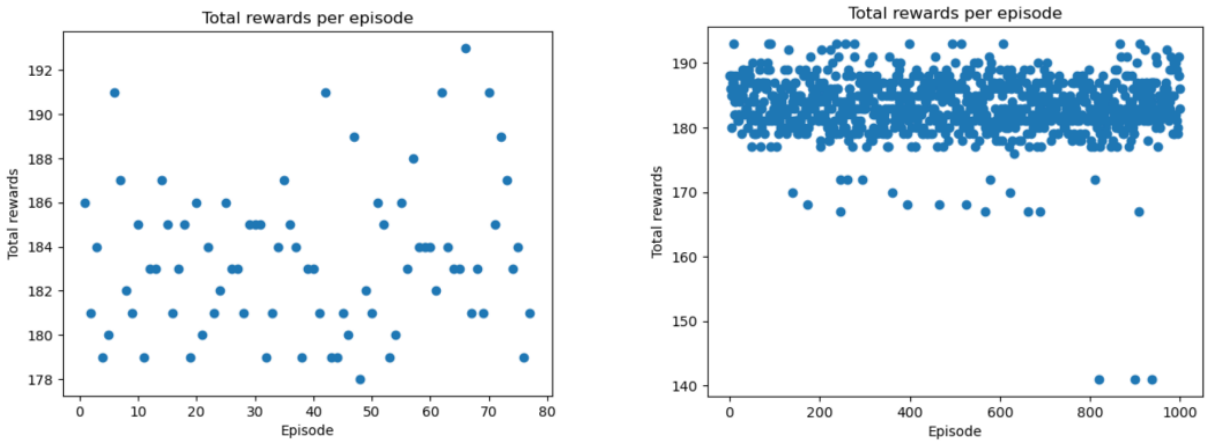
The inability of the policy to converge to a perfect solution, regardless of how long it trained for, exposes a limitation to this approach brought about by the complexity of the scenario.

## 5.3 Scenario 2B: Two Vehicles in a Narrow Passage on a 7x7 Grid

Scenario 2B moves the narrow passage configuration onto a 7x7 grid rather than a 10x10 grid to explore a lower-complexity version of the scenario. Since the grid dimensions are the same as Scenario 1, the best policy from that scenario can be used as a warm start, alleviating the need to pre-train a new policy on an obstacle-free 7x7 grid.

As calculated in Table 3.1, the maximum rewards for optimal behavior on a 7x7 grid with two agents range from 176-198. The penalty found to work best for this scenario was -5 per agent. This value was found through trial and error by testing multiple iterations of a policy with varied reward values and analyzing which values produced the lowest crash frequency. Although a minor penalty can make it difficult to tell on a plot whether a collision occurred, collisions were logged separately to track performance accurately.

The baseline policy, Policy A, achieved 100% accuracy over the fixed evaluation episodes and generalized to 98% accuracy over randomized testing episodes. Using a comprehensive evaluation, Policy D was able to find 100% accuracy over all unique combinations of episode starting locations. The comparisons made with differing training times and callback evaluation strategies are all summarized in Table 5.2.




(a) 77 fixed evaluation episodes

(b) 1,000 randomized testing episodes

Figure 5.10: Scenario 2B Policy A performance results

### 5.3.1 Baseline Policy

A partial evaluation for a narrow passage configuration on a 7x7 grid yields 77 combinations - 25% of the 306 total unique possibilities. Using the policy found in Policy D from Scenario 1 as the warm start, Policy A continued training for 1,000,000 timesteps, which took 1.78 hours to achieve 100% accuracy. The results for Policy A over the 77 fixed evaluation episodes are shown in Figure 5.10a, with all episodes placing above 176 reward points and zero collisions logged. Taking this policy and generalizing it to 1,000 randomized testing episodes resulted in 20 episodes containing collisions, giving 98.0% accuracy, illustrated in Figure 5.10b.

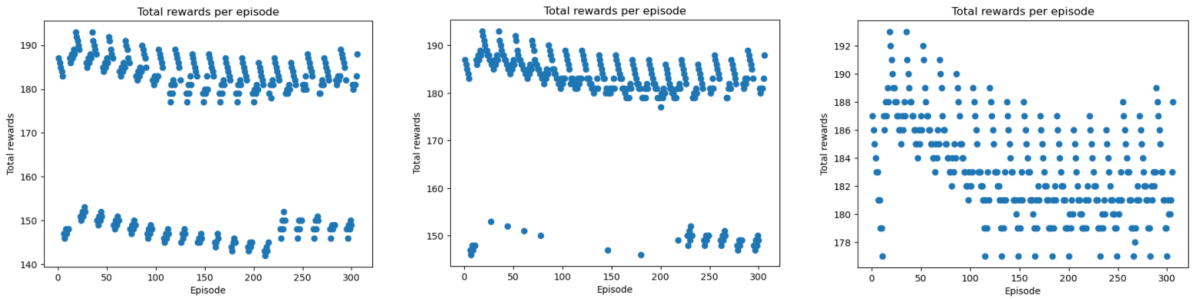
### 5.3.2 Full Evaluation with Increased Training Time - Policies B, C, and D

Next, the trade-off between accuracy and training time was explored by training policies with comprehensive callback evaluations and increased training time. Policy B, C, and D used the same warm start as Policy A, but each trained using a full rather than partial callback evaluation for additional timesteps. A full evaluation of this scenario comprises 306 possible combinations of starting locations.

Policy B trained for 3,000,000 timesteps (not including the warm start pre-training), taking 5.56 hours. When testing Policy B on all 306 unique episode configurations, there were 102 episodes containing 816 total crashes, yielding a 66.6% accuracy for a collision-free episode. The results for Policy B are displayed in Figure 5.11a, with the cyclical nature attributed to how the 306 starting combinations are enumerated.

Policy C trained for 6,000,000 additional timesteps after the warm start over 12.58 hours. While Policy C showed improvement over Policy B, the policy still did not achieve 100% accuracy over the comprehensive evaluation episodes. As seen in Figure 5.11b, Policy C reached 85.9% accuracy, achieving 263 perfect episodes but also containing 43 episodes with 342 total collisions.

While a natural progression would showcase Policy D training for a longer time than




(a) Policy B (b) Policy C (c) Policy D

Figure 5.11: Scenario 2B Policy B, C, and D performance results for a comprehensive evaluation

Policy C to achieve 100% accuracy, Policy D outperformed both Policy B and Policy C with less training time. Policy D trained only for 2,000,000 timesteps over 5.38 hours before finding a successful policy capable of achieving 100% accuracy over all 306 evaluation episodes. The results are provided in Figure 5.11c. While unusual, the randomness inherent to the training method, such as randomized starting locations in training or random policy initialization weights, means the policy may stumble onto more successful solutions earlier than expected. Since the policy continues building off that success, it can more quickly converge to an optimal solution.

### 5.3.3 Partial Evaluation with Decreased Training Time - Policy E

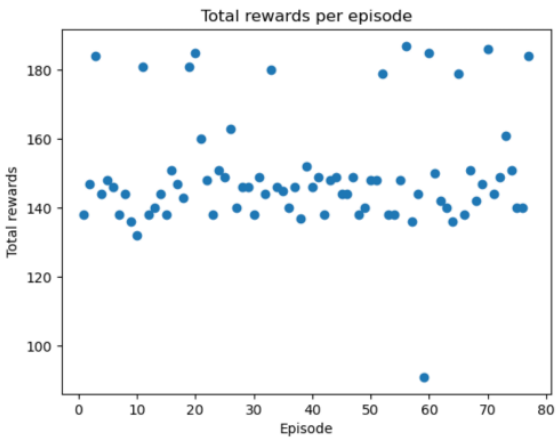
Policy E further explores the time versus performance trade-off by reducing the training timesteps from the Policy A baseline of 1,000,000 (not including the pre-training) to 250,000 timesteps for a partial callback evaluation. Training Policy E for 250,000 timesteps took only 0.15 hours but achieved only 12.9% accuracy over the 77 fixed evaluation episodes with 536 total collisions, as shown in Figure 5.12a. Policy E generalized to 16.7% accuracy as presented in Figure 5.12b, which contained 2,136 total collisions.

### 5.3.4 Using Randomized Evaluation Episodes - Policy F

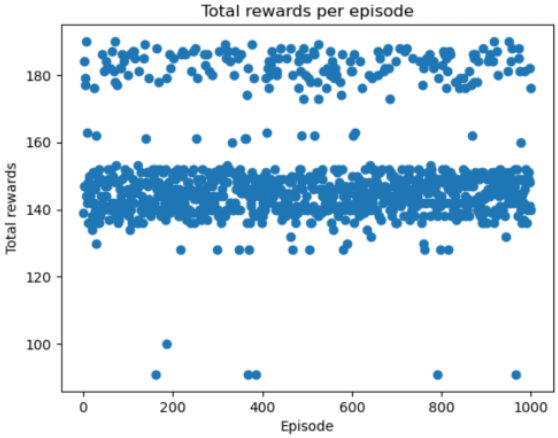
Policy F demonstrates the importance of using fixed evaluation episodes within the callback evaluation to maintain a consistent performance measure across policies. Policy F trained for 1,000,000 timesteps, as did Policy A, but Policy F's callback evaluation episodes were a randomized set of 77 episodes each time the policy was evaluated. As a result, Policy F did not achieve 100% accuracy over a set of 77 randomized evaluation episodes. Instead, after training for 0.43 hours, Policy F performed with 16.5% over 1,000 randomized testing episodes as presented in Figure 5.13.

### 5.3.5 Training without a Callback Method - Policy G

To compare the baseline, Policy A, against a policy that trains without a callback evaluation method, Policy G is trained for 1,000,000 timesteps without any pause for evaluation.




(a) 77 fixed evaluation episodes




(b) 1,000 randomized testing episodes

Figure 5.12: Scenario 2B Policy E performance results

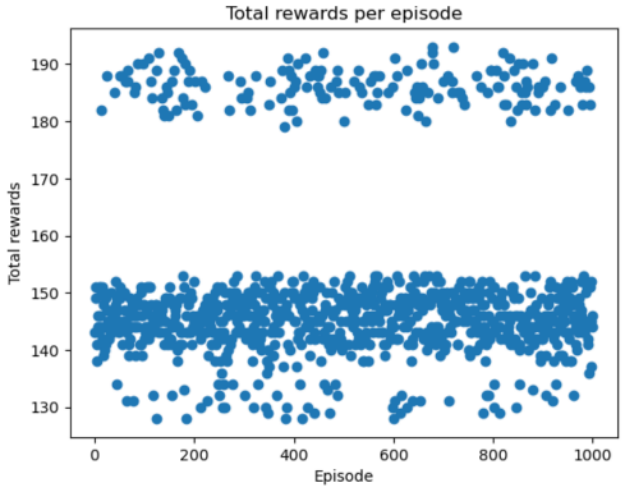



Figure 5.13: Scenario 2B Policy F performance results over 1,000 randomized testing episodes

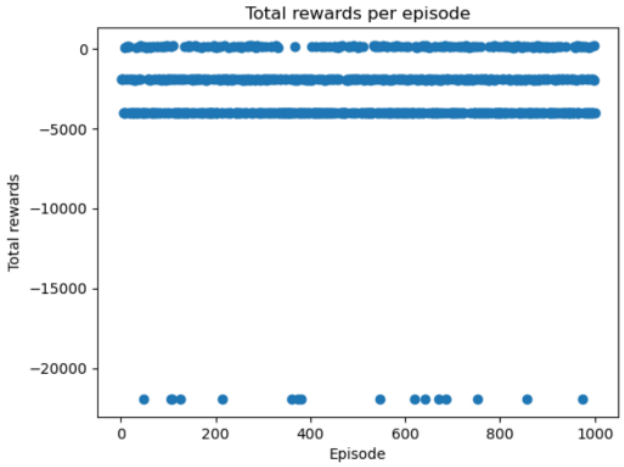



Figure 5.14: Scenario 2B Policy G performance results over 1,000 randomized testing episodes

Therefore, the policy evaluated where it ended after the one-millionth timestep rather than saving the most successful policies along the way. The policy found after the one-millionth training timestep is the policy saved and called upon for testing. The training took 0.38 hours and achieved 40.7% accuracy, but the 593 unsuccessful episodes contained 68,962 total collisions, as shown in Figure 5.14.

Given PPO's tendency to diverge and converge repeatedly, the ending policy is likely not the best. Therefore, the training may have been as successful as the baseline Policy A. However, Policy A recorded the optimal strategy when found, while Policy G relied on the most recently attempted policy.

### 5.3.6 "Cold Start" Policy - Policy H and I

The last comparison for Scenario 2B illustrates the process enhancement of using a warm start. In these iterations, there was no environment change; all training occurred in an environment with obstacles.

Policy H trained for the same timesteps as Policy A but did not utilize a pre-trained warm start. The 1,000,000 training timesteps took 10.4 hours and achieved 0% accuracy over 1,000 testing episodes, as seen in Figure 5.15. The extreme training time taken to train for 1,000,000 episodes is due to the complete lack of success; rather than both agents finding their goal and ending an episode after 5-10 steps, the agents could not master this behavior, dragging out each episode near the maximum amount of timesteps before timing.

Furthermore, Policy I also compares the process without a warm start but by training for the same cumulative number of timesteps of both the pre-trained warm start (6,000,000) and the continued training of the baseline, Policy A (1,000,000). Therefore, Policy H trained for 7,000,000 total timesteps without an environment change for a warm start, which took 78.75 hours. The results for Policy I are presented in Figure 5.16, demonstrating the 21.1% accuracy that Policy I achieved over 1,000 testing episodes.

The top line on the plot depicts episodes where agents were able to reach their goals, although it is hard to distinguish which of them included collisions given the scale of the

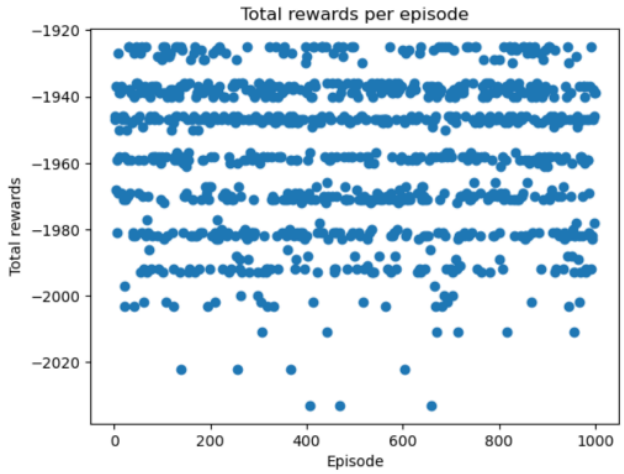



Figure 5.15: Scenario 2B Policy H performance results over 1,000 randomized testing episodes

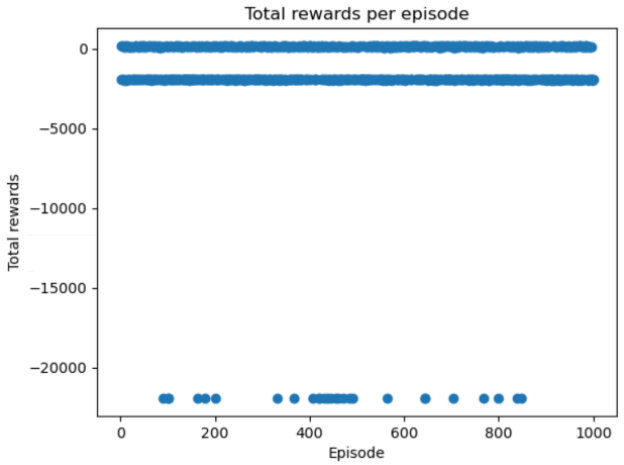



Figure 5.16: Scenario 2B Policy I performance results over 1,000 randomized testing episodes

y-axis. However, only those without collisions are included in the accuracy percentage. The middle line depicts episodes that reached the maximum timesteps without agents reaching their goals, incurring the maximum penalty for time. Lastly, the bottom line represents episodes where agents repeatedly crash and incur time penalties, maxing out the possible penalties. The results from both this policy and Policy H showcase that the complexity of the narrow passage scenario is too difficult to learn without a warm start.

## 5.4 Scenario 3: Two Delivery Robots in a Dense Urban Environment

Scenario 3 involves two delivery robots navigating a dense urban environment, as Figure 3.5 illustrates. Like the narrow passage on a 10x10 grid in Scenario 2, the dense urban environment scenario uses the same obstacle-free pre-training as a warm start since both scenarios

Table 5.2: Summary of Scenario 2B policy comparisons

| Policy | Eval. episodes | Timesteps | Callback | Eval. type | Time (hrs) | Accuracy (%) | Warm Start |
|---|---|---|---|---|---|---|---|
| A | 77 | 1M | Yes | Fixed | 1.78 | 98.0 | Yes |
| B | 306 | 3M | Yes | Fixed | 5.56 | 66.6 | Yes |
| C | 306 | 6M | Yes | Fixed | 12.58 | 85.9 | Yes |
| D | 306 | 2M | Yes | Fixed | 5.38 | 100 | Yes |
| E | 77 | 250k | Yes | Fixed | 0.15 | 16.7 | Yes |
| F | 77 | 1M | Yes | Random | 0.43 | 16.5 | Yes |
| G | N/A | 1M | No | N/A | 0.38 | 40.7 | Yes |
| H | 77 | 1M | Yes | Fixed | 0.65 | 0.0 | No |
| I | 77 | 7M | Yes | Fixed | 78.75 | 98.2 | No |

share the same observation space - a 10x10 grid. However, rather than only leveraging one warm start, the baseline policy for Scenario 3 incorporates several warm starts to increase the complexity iteratively. This is done by incorporating 10% of the obstacle-filled grid per training cycle to allow the policy to learn how to navigate the field of obstacles one row at a time.

The baseline policy for Scenario 3, Policy A, involved ten warm starts in total to reach 97.7% accuracy over the partial fixed evaluation episodes, generalizing to 97.2% accuracy over 1,000 randomized testing episodes. Using a comprehensive evaluation, Policy B trains for longer than Policy A and achieves 98.0% accuracy over all 3,080 possible starting combinations.

### 5.4.1 Baseline Policy

To train the baseline policy for Scenario 3, Policy D from Scenario 1 is leveraged as the warm start for obstacle-free navigation on a 10x10 grid. Policy A for Scenario 3 then involves continuing that training while incorporating the obstacle-filled grid 10% at a time.

In Phase 1 of Policy A, the previous pre-trained policy from Scenario 1 was copied. Next, Phase 2 of Policy A trained on an environment that only incorporated obstacles within the top row on the grid. All other rows remained obstacle-free as in the environment for Scenario 1. The agents could only start within the top row of the grid in both the randomized and fixed evaluation episodes. For Phase 3, the next row of obstacles was added to the environment for Policy A to learn. In this iteration of training, agents could start anywhere within the top two rows of the grid - the rows that contained obstacles. This process continued in Phases 3-10 until all rows of obstacles were incorporated to form a complete environment.

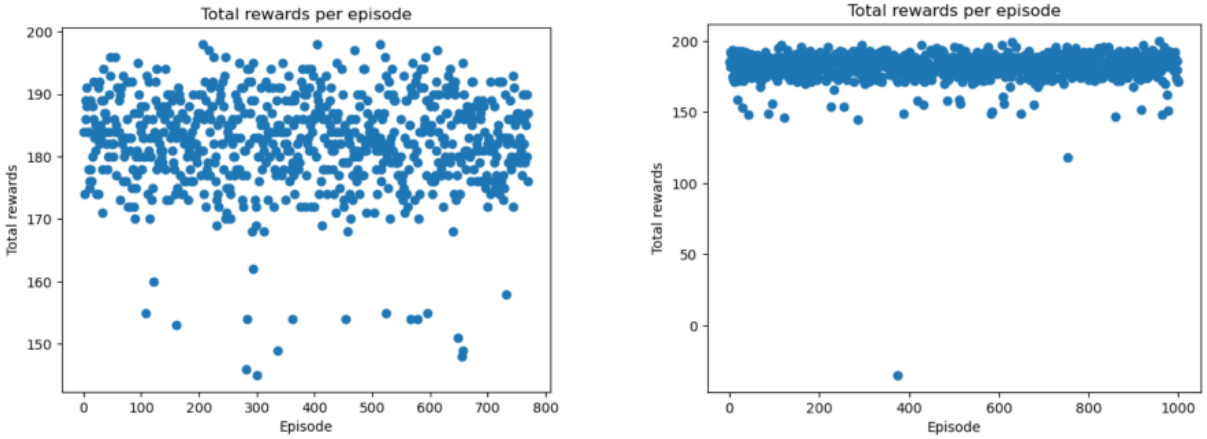
While the policy from Scenario 1 copied in Phase 1 leveraged a comprehensive evaluation, Phases 2 through 10 utilized a partial evaluation - 25% of the total possible combinations. The number of evaluation episodes increased as the phase number increased since the agents' starting locations must be located within rows with obstacles incorporated. Therefore, a partial evaluation for Phases 2-10 contained between 10-770 fixed evaluation episodes.

The training details per phase for Policy A are provided in Table 5.3. The non-linear progression for training is explained by the differing increase in complexity brought about by each phase since both the number and placement of obstacles are unique to each row. As seen in Table 5.3,

After training for a combined total of 57.8 million timesteps over 78.05 hours, Policy A achieved 97.7% accuracy over the 770 fixed evaluation episodes making up the complete

Table 5.3: Scenario 3 Policy A training phases

| Phase | Training Timesteps (million) | Training Time (hours) |
|---|---|---|
| 2 | 0.1 | 0.03 |
| 3 | 0.2 | 0.08 |
| 4 | 4 | 4.21 |
| 5 | 4 | 2.49 |
| 6 | 2 | 1.95 |
| 7 | 5 | 5.83 |
| 8 | 29 | 43.57 |
| 9 | 10 | 14.64 |
| 10 | 3.5 | 5.25 |
| Total | 57.8 | 78.05 |




(a) 770 fixed evaluation episodes

(b) 1,000 randomized testing episodes

Figure 5.17: Scenario 3 Policy A performance results

environment, as shown in Figure 5.17a. Generalizing Policy A to 1,000 randomized testing episodes revealed 97.2% accuracy, as shown in Figure 5.17b. This means that the agents could navigate the maze of obstacles, regardless of starting position, and reach their goals without collision 97.2% of the time.

### 5.4.2 Full Evaluation with Increased Training Time - Policy B

To compare the accuracy versus training time trade-off, Policy B leveraged a comprehensive warm start through each phase. The number of evaluation episodes for a comprehensive evaluation ranged from 42 to 3,080 episodes. As seen in Table 5.4, Policy B trained for 61.6 million timesteps in total, which took 140.04 hours. As a result, Policy B achieved 98.0% over the 3,080 comprehensive evaluation episodes.

Table 5.4: Scenario 3 Policy B training phases

| Phase | Training Timesteps (million) | Training Time (hours) |
|---|---|---|
| 2 | 0.1 | 0.05 |
| 3 | 2 | 1.69 |
| 4 | 4 | 2.36 |
| 5 | 5 | 3.27 |
| 6 | 2.5 | 4.73 |
| 7 | 5 | 5.45 |
| 8 | 26 | 67.58 |
| 9 | 9 | 28.63 |
| 10 | 8 | 26.28 |
| Total | 61.6 | 140.04 |

## 5.5 Scenario 4: More than Two Agents

Scenario 4 explores the scalability of this approach by experimenting with the number of agents and the grid size. Three policies are trained to direct three, four, and five agents using a 15x15 obstacle-free grid,

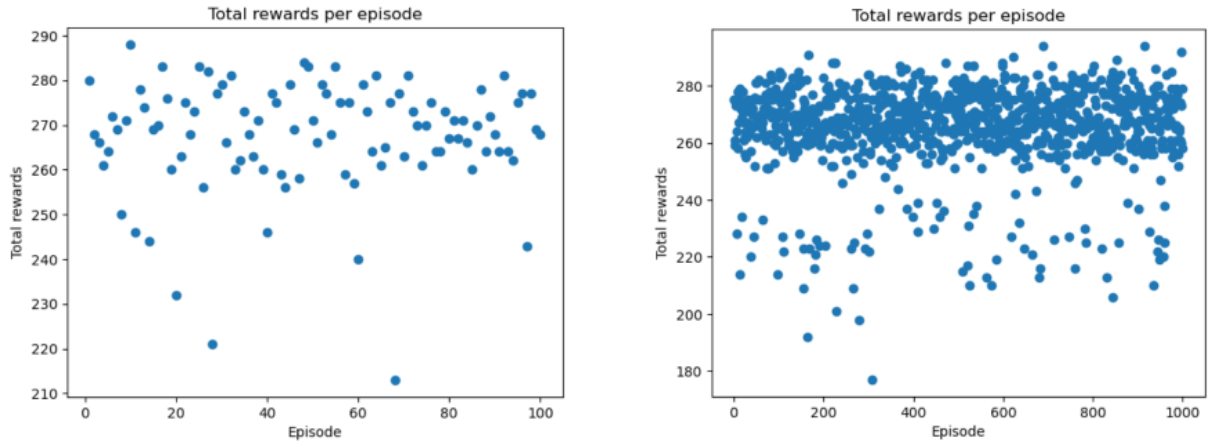
In addition to the expanded grid size, increasing the number of agents exponentially scales the number of unique starting locations. A partial evaluation using only 25% of the total unique combinations would range from 2,698,410 to 128,827,490,220 fixed evaluation episodes. Furthermore, learning to navigate the actions for three or more agents is an exponentially more complex task as compared to learning to navigate two agents. Therefore, rather than using the same approach from previous scenarios for the evaluation callback, the training for Scenario 4 uses a partial evaluation with 100 episodes, regardless of the number of total unique combinations.

Additionally, the maximum number of timesteps within an episode was increased to 5,000 to allow additional time for the controller to explore and find a successful solution. Likewise, the crash penalty was lowered to -20 to incentivize exploration since more collisions will naturally take place with more agents moving around the grid.

With these adjustments to the training approach, the environment containing three agents reaches 93.9% accuracy among 1,000 randomized testing episodes. However, the environments containing four and five agents reach 0% accuracy after training for over seven days. These results indicate that this approach is insufficient for environments containing more than three agents without further adjustments to the method.

### 5.5.1 Three Agents

A policy without any warm starts trained to direct three agents on a 15x15 obstacle-free grid for 10,000,000 timesteps. This training took 9.97 hours to complete and achieved 95.0% accuracy over the 100 fixed evaluation episodes, containing ten total collisions. This performance generalized to 93.9% accuracy over 1,000 randomized testing episodes. The results for this policy are provided in Figure 5.18




(a) 100 fixed evaluation episodes

(b) 1,000 randomized testing episodes

Figure 5.18: Scenario 4 policy performance results for three agents on a 15x15 grid

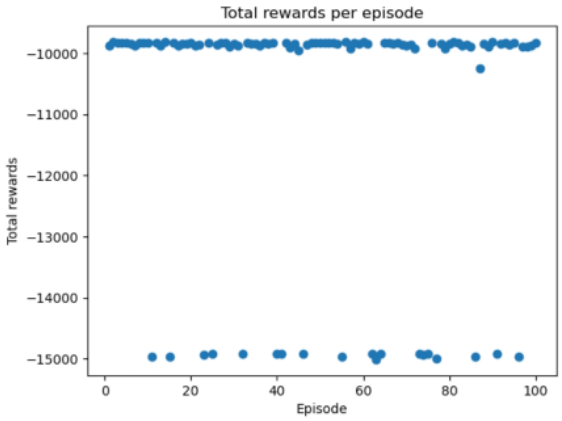
### 5.5.2 Four Agents

Similarly, a new policy trained to navigate four agents on a 15x15 obstacle-free grid for 10,000,000 timesteps. However, this training took 172.43 hours because the controller could not find a successful policy. Instead, each episode lasted until the maximum number of timesteps was reached (in this case, 5,000), and the episode terminated, unlike other scenarios where the controller quickly learned a moderately successful policy and episodes terminated after a few timesteps.

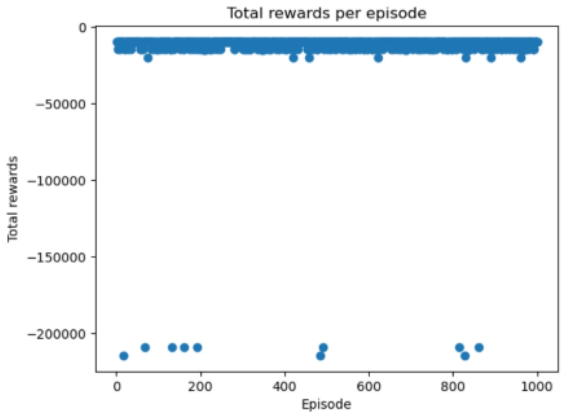
The policy here achieved 0% accuracy over the 100 fixed evaluation episodes and 0% over the 1,000 randomized testing episodes. These results can be seen in Figure 5.19. The two distinguished rows formed in 5.19a differ in how many agents reached their goal within each episode. The line around -10,000 indicates that two agents reached their goals while the other two each incurred a -1 timestep penalty for 5,000 timesteps until the episode terminated. The line around -15,000 indicates that only one agent reached its goal while the other three did not. Any interim points represent episodes that contained either repeated collisions between agents or an agent reached its goal after some time delay, during which it incurred a time penalty. While the lines are more challenging to see in 5.19b given the scale, the points on the bottom of the graph represent repeated collisions among two agents until the episode terminated after 5,000 timesteps.

### 5.5.3 Five Agents

Finally, a new policy was trained to assign deconflicting actions to five agents on a 15x15 obstacle-free grid for 10,000,000 timesteps. Similar to the four-agent scenario, the training took 181.05 hours, and the policy achieved 0% accuracy over both the 100 fixed evaluation episodes and the 1,000 randomized testing episodes. These results are provided in Figure 5.20. As in the four-agent scenario, the lines at each additional -5,000 indicate how many agents reached their goal per episode, and the interim points represent either repeated collisions or goals reached after some delay in time.

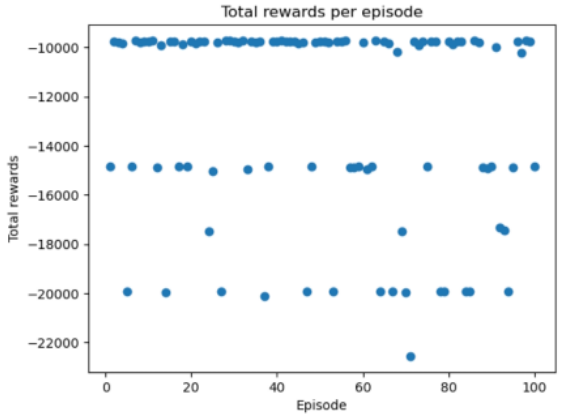



(a) 100 fixed evaluation episodes




(b) 1,000 randomized testing episodes

Figure 5.19: Scenario 4 policy performance results for four agents on a 15x15 grid




(a) 100 fixed evaluation episodes


(b) 1,000 randomized testing episodes

Figure 5.20: Scenario 4 policy performance results for five agents on a 15x15 grid

# Chapter 6

# Discussion

The results obtained from the experimental scenarios provide important insights into the effectiveness of utilizing a centralized approach with PPO for multi-agent reinforcement learning across diverse environments. This discussion summarizes the findings from each scenario, evaluates the performance resulting from differing training strategies, addresses the limitations and implications of this work, and identifies areas for future research.

## 6.1 Scenario Performance

The chosen approach resulted in various levels of performance across the experimental scenarios. The highest accuracy levels across all policies within a scenario occurred in Scenario 1, which is reflective of the relatively low level of complexity compared to other scenarios. Alternatively, the lowest accuracy levels were found in Scenario 4, indicative of the exponentially increased complexity introduced by incorporating additional agents. Limitations are found within Scenarios 2, 3, and 4, confirming that 100% accuracy cannot be reached regardless of training time, as is achievable in Scenarios 1 and 2B.

### 6.1.1 Tradespace Explanation for Scenarios 1 and 2B

This discussion compares Scenarios 1 and 2B policies by visualizing a tradespace of training time versus accuracy, as seen in Figures 6.1 and 6.2. Within these tradespaces, the utopia point - the point with optimal value for both time and accuracy, is denoted by the yellow star. The blue curve, known as the Pareto frontier, represents the optimal policies. A "dominated solution" is a point that is beaten out by another point with better values for both time and accuracy, shown in orange. Points that reside closer to the utopia point along the Pareto frontier are the non-dominated solutions, shown in green.

### 6.1.2 Scenario 1

The primary objective of Scenario 1 was for two agents to navigate a 7x7 obstacle-free grid and reach their respective goals without collisions. The results demonstrated that the policy learned by the central controller was capable of mastering this task with up to 100% accuracy.

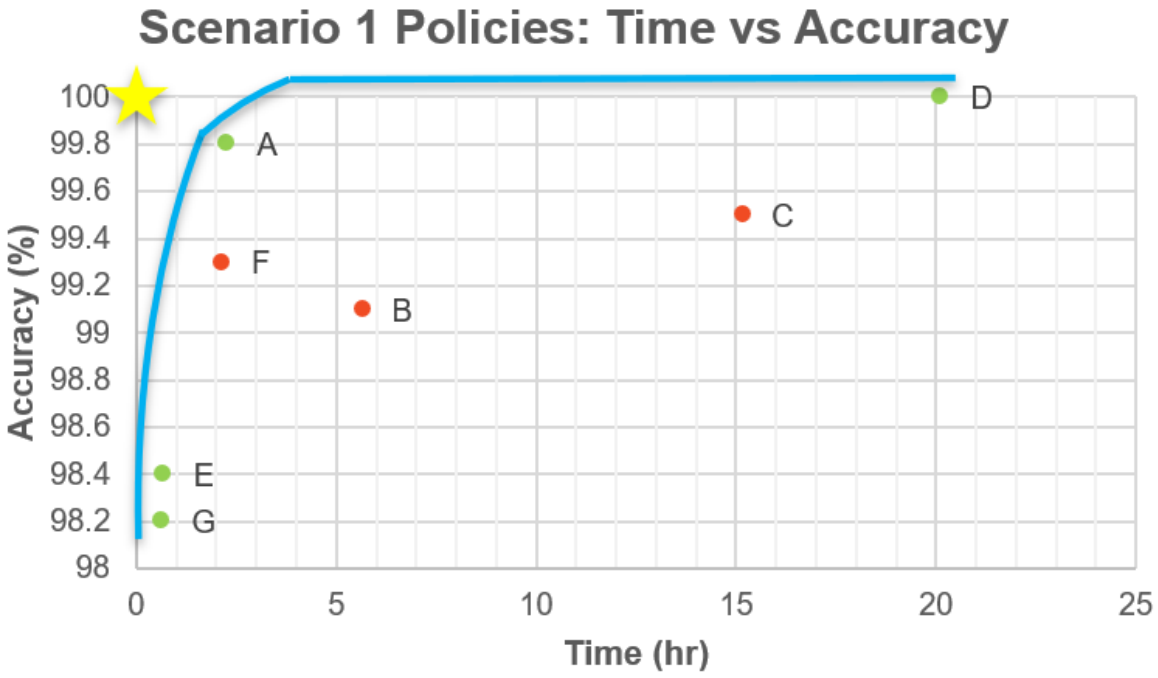



Figure 6.1: Tradespace of Scenario 1 policies

As visualized in Figure 6.1, Policies A, D, E, and G reside along the Pareto frontier, indicating they are all non-dominated policies. No other policies achieve better performance than those options regarding both time and accuracy.

Policy G achieved the lowest accuracy among all policies within Scenario 1 at 98.2% accuracy across 1,000 randomized testing episodes. This is because Policy G did not leverage an evaluation callback strategy. Thus, the policy chosen for the testing episodes was the last attempted policy during training, rather than using the best policy found and saved throughout training as transpired with other policies. However, not using an evaluation callback resulted in the least training time of any policy. Therefore, Policy G exists on the Pareto frontier despite its low performance. Therefore, if less training time is valued far more than accuracy, Policy G may be the best training strategy to select for an application.

Alternatively, Policy D achieved the highest accuracy, reaching 100% accuracy over all 2,162 unique combinations of starting locations for episode resets. This was accomplished by leveraging a comprehensive evaluation within the callback strategy, which required more training time than partial evaluations, such as in Policy A. Policies B and C also utilized a comprehensive evaluation. However, the limited training time relative to Policy D prohibited the policies from reaching 100% accuracy, highlighting the trade-off of training time versus accuracy. Although Policy D required the most training time, it is a non-dominated solution since it achieved the highest accuracy among all policies. Therefore, if 100% accuracy is valued above increased training time, this would be the best training strategy to select in an application.

While Policy G dominates in terms of minimizing training time and Policy D dominates

in terms of accuracy, Policies A and E find a middle ground, with Policy A being the most balanced among all non-dominated policies. Policy A falls along the Pareto frontier after the sharp increase in accuracy associated with additional training time but before that increase begins to plateau. Therefore, without strongly favoring one metric over the other, Policy A is the recommended training approach for any application.

### 6.1.3 Scenarios 2 and 2B

The task of Scenario 2 is for the controller to direct two agents on a 10x10 grid through a narrow passage. However, after training for up to 40,000,000 timesteps, 100% accuracy could not be reached over the partial evaluation episodes. This indicates a limitation in the approach where a level of complexity was reached that prohibited the controller from perfectly mastering the task. Scenario 2B moved the task onto a 7x7 grid, decreasing the complexity, which then resulted in a policy reaching 100% accuracy.

As illustrated in Figure 6.2, the same four policies as in Scenario 1 - Policies A, D, E, and G - reside on the Pareto frontier, representing the non-dominated solutions. However, unlike Scenario 1, the two extreme points on either end - Policies H and I - are not on the frontier. Although Policy I trained for longer than Policy D, it did not reach the 100% accuracy accomplished by Policy D, which means that Policy D dominates Policy I in both metrics. Likewise, Policy H had the lowest accuracy levels even though it trained for longer than Policy E, meaning that Policy E dominated Policy H.

As in Scenario 1, Policy A represents an optimal training strategy that balances time and accuracy. If accuracy is more valued than training time in a given application, Policy D should be selected since the increased training time led to 100% accuracy. Alternatively, if the training time metric is more critical, Policy E should be selected because it requires decreased training time from the baseline strategy.

### 6.1.4 Tradespace Findings

Together, the tradespace diagrams show that Policies A, D, E, and G are the non-dominated solutions for both Scenario 1 and Scenario 2B. This implies that these four training strategies are optimal for either environment. In choosing a training strategy for an application, one of these four should be selected based on the relative value placed on training time versus accuracy. Policies B, C, F, H, or I should not be chosen as other options dominate them in both time and accuracy metrics.

In either environment, Policy A is the recommended training strategy if there is not a strong preference towards one metric over the other. If accuracy is strongly valued, then Policy D would be the recommended training strategy. Alternatively, policies E or F would be better-suited solutions overall where training time is weighted relatively heavily.

### 6.1.5 Scenario 3

The objective for Scenario 3 is for the controller to navigate the agents through a maze simulating a dense urban environment. Splitting the training into ten phases of warm starts

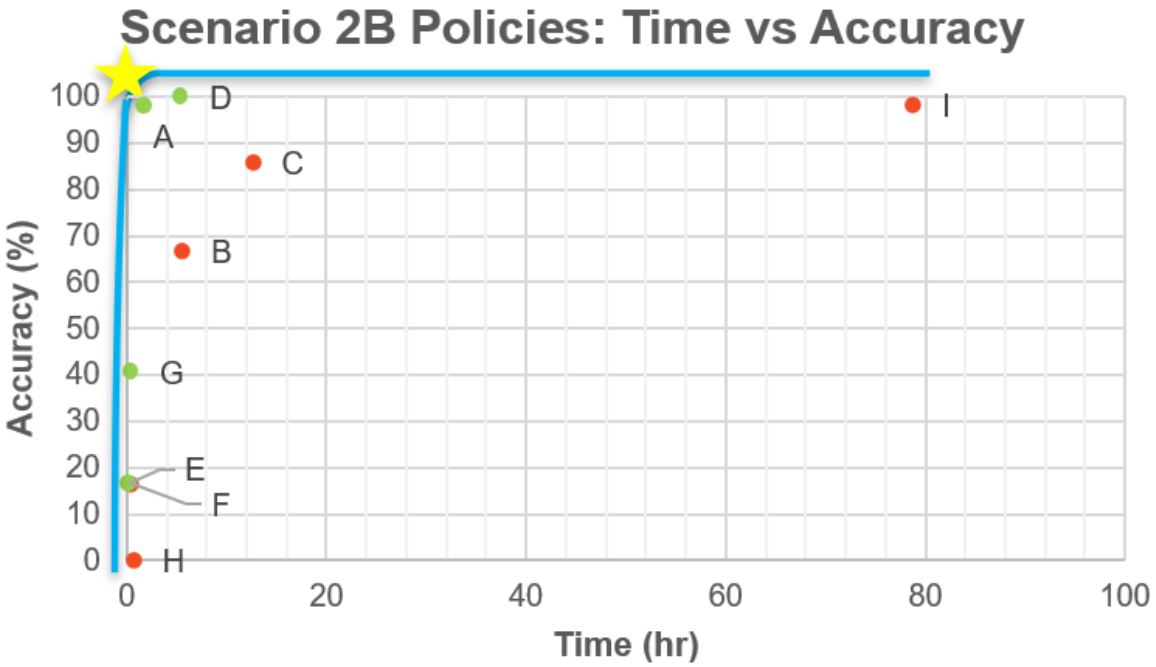



Figure 6.2: Tradespace of Scenario 2B policies

allowed the controller to learn and solve the environment one row at a time, resulting in an accuracy of 97.2% for a partial evaluation and 98.0% for a comprehensive evaluation.

Similar to Scenario 2, the comprehensive evaluation was unable to achieve 100% accuracy after training for up to 61.6 million timesteps in total, reflecting a similar limitation where higher levels of complexity are involved that make it too difficult for the controller to reach perfection. The time versus accuracy trade-off is reflected by the fact that Policy B, the comprehensive evaluation strategy, trained for nearly double the amount of time (140.04 hours) as Policy A, the partial evaluation strategy (78.05 hours), for only a 0.8% increase in accuracy. Therefore, unless the training time is trivial for an intended application, Policy A is the recommended approach for training in an environment such as Scenario 3.

### 6.1.6 Scenario 4

The objective of Scenario 4 is to test the controller's ability to direct three, four, and five agents at a time without collision on a 15x15 grid. A successful policy was found for the three-agent scenario with 93.9% accuracy. However, the policies for the four and five-agent scenarios achieved 0% accuracy after more than seven days of training. The results illustrate that while this approach can scale to include a larger grid size with three agents, this method may not be well-suited for four or more agents without further adjustments to training.

One way of employing this approach for a scenario requiring four or more agents could be to treat each agent's incorporation as an additional warm start. However, continuing training after a warm start requires each iteration of training to maintain the exact observation space

dimensions, and adding new agents typically changes those dimensions. Therefore, a warm start could be implemented by starting each episode with all but two agents already at their goals. Once a successful policy is found for those two agents, a third agent could begin starting from a non-goal location. This process could be repeated, allowing the controller to build upon the knowledge acquired from navigating the next lowest number of agents until all desired agents have been incorporated.

## 6.2 Limitations

Limitations for this approach were revealed in Scenarios 2, 3, and 4, where the controller was unable to learn a policy capable of achieving 100% accuracy. If the controller were only responsible for learning a fixed route from the same set of starting locations, such a limit would most likely not have been reached. However, these scenarios reached a level of complexity that was too difficult for a policy to perfectly master the task for every unique combination of agent starting locations.

Furthermore, Scenario 4 uncovered another limitation: the inability to continue using a 25% partial evaluation with an increased number of agents. Each additional agent exponentially scaled the number of unique starting combinations. Therefore, for a scenario involving three or more agents, it is not reasonable to use a 25% partial evaluation because the training would take far too long and likely never converge to a policy capable of solving so many variations in starting positions. Although a smaller percentage of evaluation episodes could be used for a callback evaluation, this might lead to less generalizability beyond the fixed episodes, lowering the value of employing such a strategy for a real-world application.

## 6.3 Implications for Real-World Applications

This work provides insight into the value of applying RL to multi-agent scenarios for real-world path deconfliction of autonomous robotics. When considering this approach, essential characteristics include the size of the intended environment, the number of agents, and whether solving one path is desired or, alternatively, paths from every possible starting position. The results indicate that this method can learn to solve a larger obstacle-free grid with higher accuracy than one with obstacles. However, limiting the grid size when incorporating obstacles allows the controller to achieve perfect results.

Furthermore, unlike other pathfinding algorithms discussed earlier, this approach is most valuable for its flexibility in solving various environments with few minor adjustments. This is especially valuable when the system can be pre-trained with warm starts and re-trained for the additional complexity required for a specific task in connection with that application. For instance, maintaining pre-trained policies for agents navigating multiple sets of different-sized environments will decrease the training time necessary when the obstacle configuration of a particular task is added.

Additionally, this approach works best when limiting the number of agents to three or less. Otherwise, alternative methods, such as the proposed version of an iterative warm start approach or a decentralized approach to learning, may be better suited to limit the

exponential scaling of complexity associated with incorporating additional agents.

Lastly, such limitations were found because the objective task was for the controller to solve the environment for every possible combination of agent starting locations. If the only task were for the controller to solve one path from fixed starting locations, such a limit to complexity would not have been reached, and the approach likely could have handled much larger environments with far more agents and obstacles.

# Chapter 7

# Conclusion

This research aims to demonstrate the application of RL for the navigation of multi-agent autonomous robotic systems in dynamic and uncertain environments. This conclusion summarizes the key findings, limitations of the approach, contributions to the field, implications for real-world applications, and areas for future work.

## 7.1 Summary of Key Findings

This research demonstrated the effectiveness of applying a centralized RL approach using PPO to multi-agent environments. The results illustrate the value of such an approach for environments containing up to three agents with differing obstacle configurations.

The experimental scenarios encompassed a range of complexity, from simple obstacle-free scenarios to environments containing a complex configuration of obstacles. Throughout these simulations, the centralized MARL approach learned successful policies with up to 100% accuracy in simpler scenarios. In more complex scenarios, accuracy decreased slightly, revealing the impact of additional complexity and indicating a need for adjustments to the training method to reach 100% reliability.

Various training strategies were explored, illustrating the prominent trade-off between accuracy and training time. Comprehensive evaluation callbacks resulted in the highest accuracy but required the longest training time, while partial evaluations reduced training time but were less accurate. Incorporating warm starts proved to be an effective way to learn higher complexity scenarios, but it also lengthens training time relative to simpler scenarios that do not require a warm start. A thorough analysis of the various training strategies identified several optimal strategies for an application based on the user's relative valuation of training time versus accuracy metrics.

Additionally, the research tested the scalability of this approach by incorporating three, four, and five agents at a time into a larger environment than the two-agent scenarios and found that current training strategies are insufficient for environments with more than three agents. However, the method's ability to adapt to various environments with minor adjustments suggests that advancements to expand the system could be made with further tuning in future work.

Most importantly, the results highlighted the adaptability of RL for the coordination

of multiple agents across diverse scenarios, which is where this approach derives its most significant advantage over traditional pathfinding algorithms. The method's flexibility ensures that MARL can be applied to a wide range of real-world applications, enhancing its practicality as a tool that can be maintained and deployed as needed for various operations as they arise.

## 7.2 Limitations

While this research showed promising results for applying MARL to multi-agent autonomous systems, certain limitations were identified. First, achieving 100% accuracy in scenarios with increased complexity proved challenging, indicating a need for further refinement of the method or alternative approaches. Additionally, further adjustments are necessary to transform this approach into one capable of handling environments containing more than three agents. However, the specific enhancements identified, such as iterative warm starts to incorporate agents one at a time, could very well be the solution to advancing this approach beyond its current limitations.

## 7.3 Contributions to the Field

This thesis advances the field of multi-agent autonomous robotics by providing insight into the effectiveness and limitations of applying centralized RL with PPO for navigation within diverse multi-agent scenarios. The proposed framework illustrates how leveraging training measures such as warm starts and evaluation callback strategies can significantly enhance the performance of such a tool. Lastly, this work identifies optimal training strategies and offers recommendations based on a user's relative preferences regarding training time and accuracy, contributing valuable guidelines for implementing RL for multi-agent environments.

## 7.4 Practical Implications

Leveraging a centralized RL approach for multi-agent coordination provides a robust solution for dynamic and unpredictable environments. The flexibility inherent in this approach makes MARL a practical tool for various industries, such as the military or emergency response, where real-time coordination of autonomous systems is needed on a case-by-case basis. Furthermore, incorporating warm starts enhances the flexibility of this approach because it allows the owner of such a tool to maintain generically trained policies upon which further training with more specific features can occur as the opportunity arises, limiting the training time needed to prepare for other operations. Such versatility is particularly valuable for applications requiring rapid deployment for operations involving changing conditions.

## 7.5 Opportunities for Future Work

Opportunities to further this work include incorporating moving obstacles to represent other vehicles that must be avoided within the environment that are not controlled by this system. Additionally, moving this work into a three-dimensional environment would be more realistic for many use cases, particularly aerial vehicles, expanding the potential for real-world applications. Likewise, incorporating a continuous rather than discrete action space would add greater flexibility in application and more closely represent vehicle movements within an environment. Lastly, further testing the scalability of this approach by limiting the combinations of starting locations for which the controller must solve would provide insight into the viability of this approach for more complex scenarios when numerous combinations are not necessary.

# Appendix A

# Comprehensive List of Stable Baselines PPO Parameters

Table A.1: Comprehensive list of Stable Baselines PPO parameters [39]

| **Parameter** | **Description** |
|---|---|
| policy | The policy model to use (MlpPolicy, CnnPolicy, ...) |
| env | The environment to learn from (if registered in Gym, can be str) |
| learning_rate | The learning rate, it can be a function of the current progress remaining (from 1 to 0) |
| n_steps | The number of steps to run for each environment per update |
| batch_size | Minibatch size |
| n_epochs | Number of epochs when optimizing the surrogate loss |
| gamma | Discount factor |
| gae_lambda | Factor for trade-off of bias vs variance for Generalized Advantage Estimator |
| clip_range | Clipping parameter |
| clip_range_vf | Clipping parameter for the value function |
| normalize_advantage | Whether to normalize or not the advantage |
| ent_coef | Entropy coefficient for the loss calculation |
| vf_coef | Value function coefficient for the loss calculation |
| max_grad_norm | The maximum value for the gradient clipping |
| use_sde | Whether to use generalized State Dependent Exploration (gSDE) instead of action noise exploration |
| sde_sample_freq | Sample a new noise matrix every n steps when using gSDE |
| rollout_buffer_class | Rollout buffer class to use |
| rollout_buffer_kwargs | Keyword arguments to pass to the rollout buffer on creation |
| target_kl | Limit the KL divergence between updates |

| | |
|---|---|
| stats_window_size | Window size for the rollout logging |
| tensorboard_log | The log location for tensorboard |
| policy_kwargs | Additional arguments to be passed to the policy on creation |
| verbose | Verbosity level |
| seed | Seed for the pseudo random generators |
| device | Device (cpu, cuda, ...) on which the code should be run |
| _init_setup_model | Whether or not to build the network at the creation of the instance |

# References


[1] A. Dhaliwal, "The rise of automation and robotics in warehouse management," in *Transforming Management Using Artificial Intelligence Techniques*, CRC Press, 2020, pp. 63–72.

[2] X. Zhang, X. Ma, J. Zhou, and Q. Zhou, "Summary of medical robot technology development," in *2018 IEEE International Conference on Mechatronics and Automation (ICMA)*, 2018, pp. 443–448. DOI: 10.1109/ICMA.2018.8484458.

[3] T. Asafa, T. Afonja, E. Olaniyan, and H. Alade, "Development of a vacuum cleaner robot," *Alexandria Engineering Journal*, vol. 57, no. 4, pp. 2911–2920, 2018, ISSN: 1110-0168. DOI: https://doi.org/10.1016/j.aej.2018.07.005. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1110016818300899.

[4] Y. Li, "Deep reinforcement learning: An overview," *arXiv*, p. 10, 2018. [Online]. Available: https://arxiv.org/abs/1810.06339.

[5] F. F. Arias, B. Ichter, A. Faust, and N. M. Amato, "Avoidance critical probabilistic roadmaps for motion planning in dynamic environments," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, 2021, pp. 10 264–10 270. DOI: 10.1109/ICRA48506.2021.9560974.

[6] P. Tucnik, T. Nachazel, P. Cech, and V. Bures, "Comparative analysis of selected path-planning approaches in large-scale multi-agent-based environments," *Expert Systems with Applications*, vol. 113, pp. 415–427, 2018, ISSN: 0957-4174. DOI: 10.1016/j.eswa.2018.07.001. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0957417418304160.

[7] S. Erke, D. Bin, N. Yiming, Z. Qi, X. Liang, and Z. Dawei, "An improved a-star based path planning algorithm for autonomous land vehicles," *International Journal of Advanced Robotic Systems*, vol. 17, no. 5, 2020. DOI: 10.1177/1729881420962263.

[8] P. K. Sharma, P. K. Sharma, E. G. Zaroukian, R. Fernandez, A. Basak, D. E. Asher, T. Pham, L. Solomon, and M. E. Hohil, "Survey of recent multi-agent reinforcement learning algorithms utilizing centralized training," in *Artificial Intelligence and Machine Learning for Multi-Domain Operations Applications III*, ser. Artificial Intelligence and Machine Learning for Multi-Domain Operations Applications III, vol. 11746, [Place of publication not identified] : SPIE, 2021, 117462K-117462K–12, ISBN: 1-5106-4329-X.

[9] C. Zhou, B. Huang, and P. Fränti, "A review of motion planning algorithms for intelligent robots," *Journal of Intelligent Manufacturing*, vol. 33, no. 2, pp. 387–424, 2022, ISSN: 1572-8145. DOI: 10.1007/s10845-021-01867-z.

[10] U. Acar and G. Blelloch. "Diderot - analyzing and understanding deep neural networks." (2022), [Online]. Available: https://diderot.one/courses/121/books/494/chapter/6872#atom-508869 (visited on 03/14/2024).

[11] C.-Y. ( Huang, C.-Y. Lai, and K.-T. ( Cheng, "Chapter 4 - fundamentals of algorithms," in *Electronic Design Automation*, L.-T. Wang, Y.-W. Chang, and K.-T. ( Cheng, Eds., Morgan Kaufmann, 2009, pp. 173–234, ISBN: 9780123743640. DOI: 10.1016/B978-0-12-374364-0.50011-4. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780123743640500114.

[12] H. Wang, Y. Yu, and Q. Yuan, "Application of dijkstra algorithm in robot path-planning," in *2011 Second International Conference on Mechanic Automation and Control Engineering*, 2011, pp. 1067–1069. DOI: 10.1109/MACE.2011.5987118.

[13] E. DHULKEFL, A. DURDU, and H. TERZİOĞLU, "Dijkstra algorithm using uav path planning," *Konya Journal of Engineering Sciences*, vol. 8, pp. 92–105, 2020. DOI: 10.36306/konjes.822225.

[14] Z. Zhang and Z. Zhao, "A multiple mobile robots path planning algorithm based on a-star and dijkstra algorithm," *International Journal of Smart Home*, vol. 8, no. 3, pp. 75–86, 2014.

[15] A. Ammar, H. Bennaceur, I. Chaari, A. Koubaa, and M. Alajlan, "Relaxed dijkstra and a* with linear complexity for robot path planning problems in large-scale grid environments," *Soft Computing*, vol. 20, pp. 1–23, Jul. 2015. DOI: 10.1007/s00500-015-1750-1.

[16] X. Li, "Path planning of intelligent mobile robot based on dijkstra algorithm," *Journal of Physics: Conference Series*, vol. 2083, no. 4, p. 042 034, Nov. 2021. DOI: 10.1088/1742-6596/2083/4/042034. [Online]. Available: https://dx.doi.org/10.1088/1742-6596/2083/4/042034.

[17] X. Li, X. Hu, Z. Wang, and Z. Du, "Path planning based on combinaion of improved a-star algorithm and dwa algorithm," in *2020 2nd International Conference on Artificial Intelligence and Advanced Manufacture (AIAM)*, 2020, pp. 99–103. DOI: 10.1109/AIAM50918.2020.00025.

[18] M. Naveed, D. E. Kitchin, and A. Crampton, "Monte-carlo planning for pathfinding in real-time strategy games," in PlanSIG, 2010.

[19] S. Karaman, M. R. Walter, A. Perez, E. Frazzoli, and S. Teller, "Anytime motion planning using the rrt*," in *2011 IEEE International Conference on Robotics and Automation*, 2011, pp. 1478–1483. DOI: 10.1109/ICRA.2011.5980479.

[20] Y. Lin, W. Zhang, C. Mu, and J. Wang, "Application of improved rrt algorithm in unmanned surface vehicle path planning," in *2022 34th Chinese Control and Decision Conference (CCDC)*, 2022, pp. 4861–4865. DOI: 10.1109/CCDC55256.2022.10034282.

[21] W. Lan, X. Jin, T. Wang, and H. Zhou, "Improved rrt algorithms to solve path planning of multi-glider in time-varying ocean currents," *IEEE Access*, vol. 9, pp. 158 098–158 115, 2021. DOI: 10.1109/ACCESS.2021.3130367.

[22] A. Zanardi, P. Zullo, A. Censi, and E. Frazzoli, *Factorization of multi-agent sampling-based motion planning*, 2023. arXiv: 2304.00342 [cs.RO].

[23] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2015.

[24] J. Peters, "Policy gradient methods," *Scholarpedia*, vol. 5, no. 11, p. 3698, 2010, revision #137199. DOI: 10.4249/scholarpedia.3698. [Online]. Available: http://www.scholarpedia.org/article/Policy_gradient_methods?source=post_page-----acc0344f5d72----------.

[25] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, *Proximal policy optimization algorithms*, 2017. arXiv: 1707.06347 [cs.LG].

[26] K. Feng, X. Huang, W. Wang, Z. Ji, and B. Cheng, "Representation enhancement-based proximal policy optimization for uav path planning and obstacle avoidance," *International Journal of Aerospace Engineering*, vol. 2023, p. 6 654 130, 2023, ISSN: 1687-5966. DOI: 10.1155/2023/6654130. [Online]. Available: https://doi.org/10.1155/2023/6654130.

[27] G. Zhan, X. Zhang, Z. Li, L. Xu, D. Zhou, and Z. Yang, "Multiple-uav reinforcement learning algorithm based on improved ppo in ray framework," *Drones*, vol. 6, no. 7, 2022, ISSN: 2504-446X. DOI: 10.3390/drones6070166. [Online]. Available: https://www.mdpi.com/2504-446X/6/7/166.

[28] Y. Xu, Y. Li, Q. Liu, J. Gao, Y. Liu, and M. Chen, "Multi-agent pathfinding with local and global guidance," in *2021 IEEE International Conference on Networking, Sensing and Control (ICNSC)*, vol. 1, 2021, pp. 1–7. DOI: 10.1109/ICNSC52481.2021.9702234.

[29] Y. Yang, L. Juntao, and P. Lingling, "Multi-robot path planning based on a deep reinforcement learning dqn algorithm," *CAAI Transactions on Intelligence Technology*, vol. 5, no. 3, pp. 177–183, 2020. DOI: https://doi.org/10.1049/trit.2020.0024. eprint: https://ietresearch.onlinelibrary.wiley.com/doi/pdf/10.1049/trit.2020.0024. [Online]. Available: https://ietresearch.onlinelibrary.wiley.com/doi/abs/10.1049/trit.2020.0024.

[30] J. Park, J. Kim, and T. K. Park, "Implementation of ppo for multi-agent path finding with dynamic obstacles,"

[31] Z. Rongcai, X. Hongwei, and Y. Kexin, "Autonomous collision avoidance system in a multi-ship environment based on proximal policy optimization method," *Ocean Engineering*, vol. 272, p. 113 779, 2023, ISSN: 0029-8018. DOI: https://doi.org/10.1016/j.oceaneng.2023.113779. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0029801823001634.

[32] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, *Openai gym*, 2016. arXiv: 1606.01540 [cs.LG].

[33] R. Stern, "Multi-agent path finding – an overview," in *Artificial Intelligence: 5th RAAI Summer School, Dolgoprudny, Russia, July 4–7, 2019, Tutorial Lectures*, G. S. Osipov, A. I. Panov, and K. S. Yakovlev, Eds. Cham: Springer International Publishing, 2019, pp. 96–115, ISBN: 978-3-030-33274-7. DOI: 10.1007/978-3-030-33274-7_6. [Online]. Available: https://doi.org/10.1007/978-3-030-33274-7_6.

[34] A. Nandy and M. Biswas, "Reinforcement learning basics," in *Reinforcement Learning : With Open AI, TensorFlow and Keras Using Python*. Berkeley, CA: Apress, 2018, pp. 1–18, ISBN: 978-1-4842-3285-9. DOI: 10.1007/978-1-4842-3285-9_1. [Online]. Available: https://doi.org/10.1007/978-1-4842-3285-9_1.

[35] M. Sewak, *Deep reinforcement learning*. Springer, 2019.

[36] W. Zhu and A. Rosendo, *Proximal policy optimization smoothed algorithm*, 2020. arXiv: 2012.02439 [cs.LG].

[37] Y. Wang, H. He, C. Wen, and X. Tan, *Truly proximal policy optimization*, 2020. arXiv: 1903.07940 [cs.LG].

[38] M. Sun, V. Kurin, G. Liu, S. Devlin, T. Qin, K. Hofmann, and S. Whiteson, *You may not need ratio clipping in ppo*, 2022. arXiv: 2202.00079 [cs.LG].

[39] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, *Stable-baselines3: Reliable reinforcement learning implementations*, 2021. [Online]. Available: http://jmlr.org/papers/v22/20-1364.html.