

# Multi-UAV Navigation System

A

Thesis Submitted

in Partial Fulfillment of the Requirements

for the Degree of

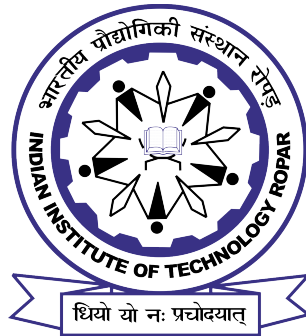
MASTER OF TECHNOLOGY

By

ARSHDEEP SINGH

Under the Supervision of

Dr. SHASHI SHEKHAR JHA



Department of Computer Science & Engineering


Indian Institute of Technology Ropar

July 24, 2021



# DECLARATION

This is to certify that the thesis entitled “**Multi-UAV Navigation System**”, submitted by me to the *Indian Institute of Technology Ropar*, for the award of the degree of Master of Technology, is a bonafide work carried out by me under the supervision of Dr. Shashi Shekhar Jha. The content of this thesis, in full or in parts, have not been submitted to any other University or Institute for the award of any degree or diploma. I also wish to state that to the best of my knowledge and understanding nothing in this report amounts to plagiarism.

Sign:   
\_\_\_\_\_

**Arshdeep Singh**  
**Department of Computer Science & Engineering,**  
**Indian Institute of Technology Ropar,**  
**Rupnagar-140001, Punjab, India.**

Date: 24-July-2021  
\_\_\_\_\_



# CERTIFICATE

This is to certify that the thesis entitled “**Multi-UAV Navigation System**”, submitted by Arshdeep Singh (2019CSM1001), a master’s student in the *Department of Computer Science & Engineering, Indian Institute of Technology Ropar*, for the award of the degree of Master of Technology, is a record of an original research work carried out by the candidate under my supervision and guidance. The thesis has fulfilled all requirements as per the regulations of the institute and in my opinion has reached the standard worthy of the award of the degree. The results embodied in this thesis have not been submitted to any other University or Institute for the award of any degree or diploma.

Sign: 

---

**Supervisor: Dr. Shashi Shekhar Jha**  
**Department of Computer Science & Engineering,**  
**Indian Institute of Technology Ropar,**  
**Rupnagar-140001, Punjab, India.**

Date: 24-July-2021



# ACKNOWLEDGEMENTS

I would like to take this opportunity to express my gratitude and respect to all those who helped me throughout the duration of this thesis. I would like to thank Dr. Shashi Shekhar Jha who offered me his assistance, guidance, and valuable ideas throughout the development of this thesis. A Master's thesis is a major milestone during the study period of a student. So it is a great pleasure to have the opportunity to achieve the completion of this work. Finally, I would like to thank the Department of Computer Science for letting me expand my knowledge.

Sincerely  
Arshdeep Singh





# ABSTRACT

The deployment of Unmanned Aerial Vehicles (UAVs) requires trajectory optimizations concerning the target reachability and other UAVs. This work presents a Multi-Agent Reinforcement Learning framework for solving Multi-UAV planning and navigation problem. The system involves having a depot of UAVs and they are tasked with reaching their target locations. The UAVs are expected to follow a safe trajectory while carrying out their tasks. This work focuses on solving the problem of target reachability and safe navigation. We have used Safety-Aware off-policy Multi-Agent Deep Deterministic Policy Gradient algorithm for solving the problem. Safe Reinforcement Learning will help in transitioning the control from simulation to real physical robots. We simulated our problem in Multi-Agent Particle environment and the Webots Simulator and provided a detailed analysis of the proposed solution. We demonstrate that Multi-Agent Reinforcement Learning is effective in solving the continuous control of robotic tasks.



# Contents

<b>List of Figures</b>	<b>v</b>
<b>List of Tables</b>	<b>vii</b>
<b>1 Introduction</b>	<b>1</b>
<b>2 Related Work</b>	<b>5</b>
<b>3 Preliminaries</b>	<b>9</b>
3.1 Markov Decision Processes . . . . .	9
3.2 Actor-Critic Algorithms . . . . .	10
3.2.1 Deep Deterministic Policy Gradient (DDPG) . . . . .	10
3.2.2 Multi-Agent Deep Deterministic Policy Gradient (MADDPG) . . . . .	11
3.3 Safe Reinforcement Learning . . . . .	12
<b>4 Proposed Method</b>	<b>13</b>
4.1 Safe-MADDPG . . . . .	13
<b>5 Environment</b>	<b>17</b>
5.1 Multi-Agent Particle Environment (MAPE) . . . . .	17
5.2 Webots Simulation Environment . . . . .	18
5.2.1 Combined Emitter-Receiver scheme . . . . .	19
5.3 Observation Space . . . . .	20
5.3.1 MAPE Environment . . . . .	20
5.3.2 Webots Environment . . . . .	20
5.4 Action Space . . . . .	21
5.4.1 MAPE Environment . . . . .	21
5.4.2 Webots Environment . . . . .	21
5.5 Reward Structure . . . . .	22
5.5.1 MAPE Environment . . . . .	22
5.5.2 Webots Environment . . . . .	22
5.6 Network Architecture . . . . .	23
5.6.1 MAPE Environment . . . . .	23
5.6.2 Webots Enviornment . . . . .	23
5.7 Network Hyperparameters . . . . .	24
<b>6 Evaluation</b>	<b>25</b>
6.1 MAPE environment . . . . .	25

---

6.1.1	Mean Rewards . . . . .	25
6.1.2	Inter-Agent Collisions . . . . .	26
6.1.3	Testing Final Policies . . . . .	27
6.1.4	Trajectory Plotting . . . . .	28
6.2	Webots Environment . . . . .	28
6.2.1	Mean Rewards . . . . .	29
6.2.2	Testing Final Policy . . . . .	29
6.2.3	Controller Variation . . . . .	30
6.2.4	Value Function Comparison . . . . .	31
6.2.5	Policy Loss and Critic Loss . . . . .	32
6.2.6	Trajectory Plotting . . . . .	33

# List of Figures

4.1	Safe-MADDPG . . . . .	14
5.1	Webots Simulation . . . . .	18
5.2	Deepbots integration [1] . . . . .	19
6.1	Mean Training Rewards (MAPE) . . . . .	26
6.2	Collisions per 1000 episodes . . . . .	26
6.3	Collisions per 1000 episodes w.r.t. agent count . . . . .	27
6.4	Test time collisions averaged over 5 runs . . . . .	27
6.5	Maximum reward (N=3) . . . . .	28
6.6	Minimum reward (N=3) . . . . .	28
6.7	Maximum reward (N=5) . . . . .	28
6.8	Minimum reward (N=5) . . . . .	28
6.9	Mean Training Rewards (Webots) . . . . .	29
6.10	Test time collisions (Webots) . . . . .	30
6.11	Pitch value . . . . .	30
6.12	Yaw value . . . . .	31
6.13	Current v.s. Target Q-value . . . . .	32
6.14	Policy and Value function loss . . . . .	32
6.15	UAVs Trajectory Plot . . . . .	33



# List of Tables

5.1	Reward structure for 2D case (MAPE) . . . . .	22
5.2	Reward structure for 3D case (Webots) . . . . .	23
5.3	Same Hyperparameters . . . . .	24
5.4	Different Hyperparameters . . . . .	24





# Chapter 1

## Introduction

---

**M**ULTI-agent systems find their applications in robotics in various domains, the prominent being Unmanned Vehicles. Quadcopters form a significant part of multi-agent systems under the aerial category. Unmanned Aerial Vehicles(UAVs), a.k.a. quadcopters, have seen a sudden push in e-commerce delivery businesses like Amazon, USPS [2]. Quadcopters will be effective in any complex environment, be it industrial or hazardous. It can provide essential applications such as healthcare delivery, food delivery or postal delivery, etc. In addition, quadcopters offer various advantages such as invariance to road traffic, the higher delivery rate under a specific timeline, easy deployment.

We formulated our problem as a multi-UAV navigation system where each quadcopter is modeled as an intelligent agent. The quadcopter is configured to start from its initial position and move towards its target through safe navigation. The problem is solved through a multi-agent Reinforcement Learning(MARL) framework. In addition, the quadcopters are geared towards learning cooperative behavior among themselves to maximize their reward collection. The problem of inter-UAV collisions is handled through safe action space constraints promoting safe navigation.

Safe RL involves constraint satisfaction while the agent interacts with the environment, thereby maximizing its rewards. Safety has been explored through various approaches. One such approach is reward shaping which encodes the safety constraints into the reward structure. Other approaches deal with formulating an additional controller which will ensure safe exploration into the environment.

Multi-agent planning and coordination is a challenging problem. The success of Deep RL in single-agent environments has prompted the research towards applying it to multi-agent environments. Deep Deterministic Policy Gradient [3] implements the continuous control by directly optimizing the *policy*. However, applying the policy gradient to a multi-agent scenario leads to *non-stationarity* of the environment. So, the authors in [4] proposed how to mitigate the *non-stationarity* using joint action space in the training of the agents. Our work builds upon the multi-agent deep deterministic policy gradient, which follows the pattern of centralized training with decentralized execution.

In this work, we propose how to solve multi-UAV coordination considering *safe* path navigation approach. The safety of the traditional RL algorithm should be ensured while deploying the algorithm over the actual hardware. To ensure that, we integrated the closed-loop solution provided by [5] in the form of a safety layer. This safety layer is applied to the algorithm, which does correction of the taken action by the UAV in case of constraint violation. We implemented our proposed Safe-MADDPG algorithm in two environments, namely Multi-Agent Particle Environment (MAPE) [6] and Webots Simulator [7]. The former is a 2D environment, while the latter is a 3D environment that follows real-world physics simulation. The algorithm follows a two-step approach; we first train the safety layer weights where random walks are simulated to make the agents learn *safe actions*. Then, the main MADDPG algorithm starts the training that follows UAV learning to reach their targets following safe action space, thereby avoiding a mid-flight crash.

Our contributions to the work are as follows:

- We have developed a target reaching environment over Multi-Agent Particle Environment(MAPE) open-sourced by [6].

- We implemented MADDPG [4] for the MAPE environment. To ensure the safety of actions, we have added a safety layer [5], thus proposing Safe-MADDPG. Further, sampling from Replay Buffer is explored by comparing Uniform Sampling with Prioritized Sampling.
- We further analyzed the scaling of the environment with an increasing number of agents and plotting the collisions, rewards, and paths generated over training and testing.
- Next, we implemented the algorithm by reconstructing the environment (from 2D to 3D) in Webots Simulator by Cyberbotics [7]. The Webots simulation follows real-world dynamics. For our problem, we have tested with the quadcopter : DJI Mavic Pro.
- Both MADDPG and Safe-MADDPG are implemented, and their performance is compared with respect to target reachability and collision avoidance.
- Experimental results depicting the effect of control dynamics (*Pitch*, *Yaw*) with respect to the UAV in the modeled environment and algorithm are further tested upon and thoroughly described.

The rest of the thesis is organized as follows. Chapter 2 describes the related work. Chapter 3 covers the preliminaries for our algorithm, and Chapter 4 describes our proposed algorithm. Chapter 5 cover the environmental design for both 2D(MAPE) and 3D(Webots) environment. As we transitioned from 2D to 3D, Chapter 6 covers the evaluation results for both the environments.



## Chapter 2

# Related Work

---

**T**HIS chapter outlines the progress of Reinforcement Learning in Multi-Agent Systems. Reinforcement learning has found considerable progress in robotics [8] and playing games [9]. Training RL algorithms usually falls under two domains, i.e., *value based* or *policy based* methods. The *policy gradients* by [10] overcomes the drawbacks of DQN [9] by improving the performance over large action space and extending it to the continuous control. [10] forms the basis algorithm for our problem in multi-agent scenario.

Multi-agent reinforcement learning (MARL) is gaining momentum recently. MARL has been researched in various areas such as joint action learning, cooperation and competitiveness, coordination, security, self play, learning to communicate, Transfer Learning, Imitation Learning, and Meta-Learning. It is also finding its application in Game Theory with respect to extensive form games. Recent work is following the approach of *centralized training* and *decentralized execution*. It has significantly improved the convergence of MARL systems. The same approach is followed in [4]. It can display complex emergent behavior between the agents with respect to their objectives. The authors used off-policy data for centralized training. The authors in [11] also used centralized critic for training their algorithm. However, under these scenarios, we have to fix the number of agents

beforehand. Also, the above papers take into account that the agents have a global view of the environment. However, these algorithms fail to ensure the safety of the actions, which is crucial for real-world scenarios. The continuous observation and action space fail to encode safety into the reward function.

Recently safety constraints are applied to the reinforcement learning algorithms taking real-world deployment criteria into considerations. [5] and [12] applies safety to the policy gradient algorithm, Deep Deterministic Policy Gradient(DDPG) [3]. The former applies a closed-loop solution, thereby finding optimal Lagrange multipliers for finding the optimal action correction. In contrast, the latter applies a Control Barrier Function(CBF) to the existing policy of the agent. Both papers apply it to the single-agent environment.

The authors in [13] apply a two-step approach to handle the inter-agent collisions. First, the algorithm is trained on the Optimal Reciprocal Collision Avoidance (ORCA) framework for collision avoidance. After that, the algorithm is trained on policy gradient till convergence. However, this approach becomes highly restricted in 3D environments where the agents are susceptible to falling into local minima. The authors in [14] provide a decentralized approach for UAV collision avoidance and path planning. The agents are trained using Proximal Policy Optimization [8]. However, it learns collision avoidance during learning which will not be practical for training real-world robots.

Our approach utilizes both [4] and [5] to achieve collision avoidance while UAVs are navigating to single or multi-target locations. The method follows applying [5] in a multi-agent setting. The algorithm takes two-step progress, first tuning the safety layer and then training the agents under MADDPG using safe actions through the safety layer.

UAVs are primarily tested in simulation before transferring to real-world UAVs. The main simulation environments are Microsoft AirSim, Gazebo, and Webots. [15] provides a complete analysis and comparison of various real-world robotic simulators and their strengths and weaknesses. [16] provides a framework for UAV control in Gazebo Environment. It collaborates with ROS integration which provides micro-controller transportation on real hardware systems. Most research papers have used real drones such as Parrot Drone, PhoenixDrone, CrazyFlie 2.0 for their experimentation. We have used the

DJI Mavic drone as provided by Webots simulator for experimentation. It also provides the necessary modules for streamlined transportation to the hardware controllers.





## Chapter 3

# Preliminaries

---

**I**N this chapter, we describe the methods used to solve for the safe navigation of the Multi-UAV coordination.

### 3.1 Markov Decision Processes

This work builds up upon Markov decision processes(MDPs). We will take  $N$  as the number of UAVs in our environment. The agents will take observations  $o_1, o_2, o_3, \dots, o_N$  and then they will execute the actions  $a_1, a_2, a_3, \dots, a_N$ . The observation space includes a concatenation of agent's position in global coordinates and other entities' relative positions, i.e.  $o_i : (s_1, s_2, \dots, s_N)$ . The action space consists of pitch value and yaw angle responsible for the motion of the agent.

Each agent will follow a policy  $\pi_{\theta_i} : o_i \times a_i \rightarrow \mathbb{R}$  and transitions into the next state  $s'_1, s'_2, s'_3, \dots, s'_N, \forall i \in N$ . The transition function is  $T : s_i \times a_1 \times a_2 \dots \times a_N \rightarrow s'_i, \forall i \in N$ . The agent will receive a reward value based on the state  $s_i$  and the action  $a_i$  taken i.e.  $r_i : s_i \times a_i \leftarrow \mathbb{R}, \forall i \in N$ . The reward function will be same for all agents. Each agent will try to maximize its sum of discounted rewards  $R_i$  i.e for each agent  $R_i = \sum_t^T \gamma^t r_i^t$ ,

where  $r_i^t$  is the single timestep reward generated,  $\gamma$  is the discount factor and  $T$  is the time horizon which will mark the end of full episode,  $\forall i \in N$ .

## 3.2 Actor-Critic Algorithms

Our work falls under the category of Model Free RL, where the agents don't require the dynamics of the environment. So it works towards optimizing the *policy*, which can be through policy gradients [17]. Other algorithms apply to approximate Dynamic Programming by evaluating *value functions* represented as Bellman Equations. Finally, Actor-Critic methods apply the previous two methods for finding the optimal control policies [18]. Actor-Critic algorithms can be On-policy or Off-policy. Under the On-policy algorithms, the policy which is used to select the action, the same policy is used for evaluation and gets improved over the timestep. Under the Off-policy algorithms, action selection is made from a different policy instead of the policy which is being evaluated and improved.

### 3.2.1 Deep Deterministic Policy Gradient (DDPG)

Deep Deterministic Policy Gradient (DDPG) [3] is a model-free algorithm and it is a combination of [10] and [9]. It learns by maintaining a frozen target network. It follows a deterministic policy  $\mu(o)$ , and exploration is handled by adding noise  $\mathcal{N}$ .

$$\mu_i(o) = \mu_{i|\theta}(o) + \mathcal{N}$$

The DDPG uses soft updates for changing the target actor and critic network. It deviates from DQN, where the target network is changed after some threshold. The actor and the critic network maintains two copies for training stability and the target networks are updated as follows :  $\theta' \leftarrow \tau\theta + (1 - \tau)\theta'$  where  $\tau \ll 1$ .

### 3.2.2 Multi-Agent Deep Deterministic Policy Gradient (MADDPG)

Next, MADDPG provides an extension to DDPG [3]. However, it involves with *non-stationarity* of the environment. It solves it by making the critic network learn *centralized* action-value function to estimate the Q-values for each agent. The MADDPG uses Ornstein–Uhlenbeck (OU) Noise to handle the exploration in the continuous action space. The discrete action space is handled with one-hot logits space.

The Centralized Critic in MADDPG is represented as  $Q_i^\mu(o_i, a_1, a_2, \dots, a_n)$  where  $a_1, a_2, \dots, a_n$  are the individual actions of all the agents in the system. Each agent learns its Q-function  $Q_i^\mu$  separately. So this leads to the agents having separate individual rewards and due to this the agents can be modeled in both cooperative and competitive settings. The policy actor networks are updating their neural parameters according to the equation:

$$\nabla_{\theta_i} J \approx \mathbb{E}_{o, a \sim D} [\nabla_{\theta_i} \mu_i(o_i) \nabla_{a_i} Q_i^\mu(o_i, a_1, \dots, a_N) |_{a_i = \mu_i(o_i)}]$$

And the critic update equation is:

$$L(\theta_i) = \mathbb{E}_{o, a, r, o'} (y - Q_i^\mu(o_i, a_1, \dots, a_N))^2$$

where the TD target is set as:

$$y = r_i + \gamma Q_i^{\mu'}(o'_i, a'_1, \dots, a'_N)$$

Here, D is the replay buffer used for drawing samples for training as a minibatch. It stores the past history of the agent in the environment. Also,  $\mu'$  indicates the target network and is updated using soft parametric update by  $\tau$ .

As MADDPG is an off-policy algorithm, we sample the experience from the linear distribution in the case of uniform sampling. In the case of prioritized experience replay [19], we also take into account the probability and weight of the experiences. There is a change in priority after each pass through the network based on the TD error. The bias is also corrected in the loss function during the training. The probability of selection of experiences is based on :  $P(i) = \frac{P_i^\alpha}{\sum_k P_k^\alpha}$ . Here,  $P(i)$  is the probability of the action

selection,  $p$  is the priority,  $\alpha$  tunes the prioritization. The bias is corrected by using  $w_i \delta_i$  instead of  $\delta_i$ , where :

$$w_i = \left( \frac{1}{\text{BatchSize}} * \frac{1}{P(i)} \right)^\beta$$

### 3.3 Safe Reinforcement Learning

Safe Exploration in continuous action spaces is achieved through applying safety signals to the policy. The action space is trained over constraint satisfaction. After that, whenever an action violates a constraint, e.g., if it enters a No-Fly-Zone or comes in the vicinity of other UAVs, the action correction will come into play. The authors in [5] applied a closed-loop solution in the form of a safety layer which is an additional neural network layer. The constrained policy optimization equation is

$$\max_{\theta} E[\sum_t \gamma^t R(o_i, \mu_i | \theta(o_i))]$$

s.t.

$$c_k(o_i) \leq C_k, \forall k \in K$$

The constraints are applied over each positions of the UAVs w.r.t. the closest UAV. The policy from the actor is passed through the safety layer  $g(s, w)$ . Prior to running the policy gradients we first train the safety layer whose weights are learned through quadratic program viz

$$a_i^* = \operatorname{argmin}_{a|i} \frac{1}{2} \| a_i - \mu_i | \theta(o) \|^2$$

s.t.

$$c_k(o_i) + g(o, w)^T a_i \leq C_k, \forall k \in K$$

The optimal solution exists which is to find the optimal Lagrange multipliers  $\lambda^*$  by solving the KKT conditions w.r.t. the optimization problem. The above equation solves how little the safety layer has to perturb the original action in the euclidean norm space to satisfy the constraint criteria. After the layer is trained, safe action space is used in the next phase of the proposed algorithm.

## Chapter 4

# Proposed Method

---

**I**N our work, we propose a Safe-MADDPG based solution to handle Multi-UAVs coordination and navigation. Our method works on training each agent with a deterministic policy with each one having *same reward* structure. The algorithm is depicted in Fig. 4.1. We have made assumptions that the quadcopters have *global* view of the system, which means the exact location of the targets and other UAVs are available, which will be encoded in the observation space.

After the safety layer is learned in phase 1, when deep policy  $\pi_i$  will output action  $a_i$ , the safety layer will do minimal corrections to the original action and outputs action  $a_i^*$ . The actor UAVs will take corrected action in *execution* while joint action space of all the actors is taken during the training part as depicted in Fig. 4.1. The safety layer is shaded blue for each agent.

### 4.1 Safe-MADDPG

The algorithm is mentioned in *Algorithm 1*. Step 1 consists of training the safety layer, which is added to each policy  $\pi_i$  (Line 1-17). It starts with a random policy and runs

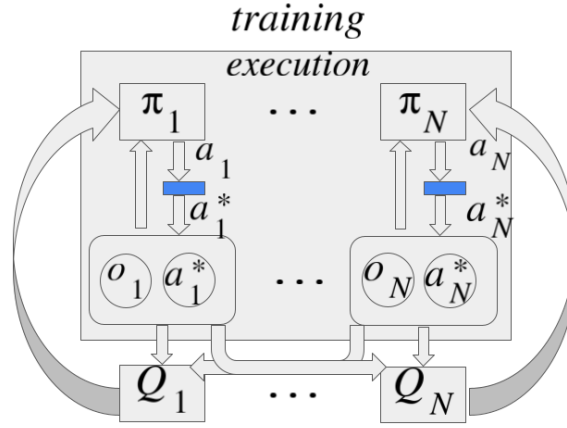


FIGURE 4.1: Safe-MADDPG

the episode up to Total Sample Steps. Line 3 resets the environment and each agent receives an observation  $o_i$ ,  $\forall i \in N$ . As each actor policy  $\pi_i$  outputs *pitch* and *yaw*, so Line 4 calculates the constraint values in L2 norm. The tuple will indicate the minimum and maximum value of *pitch* and *yaw* respectively required to satisfy the constraints. Under Line 5, agents take actions, observe a reward, and the next set of constraint values  $c_i^{\text{next}}$  are generated (Line 6). Lines 9-16 observe the gradient updates of the safety layer, where the mini-batch is drawn for each agent (Line 10). The observations are input to the safety layer (Line 11). Here, the next constraints are predicted through the safety layer as indicated in Line 13 by  $c_i^{\text{next.predicted}}$ . Line 14-15 indicates the squared loss calculation for each constraint  $C$  and weights update. Here,  $C$  means constraint on each action value. So, for 2-element action space, there are 4 constraints stressing maximum and minimum value.

Under Step 2 (Lines 18-39), the MADDPG loop starts. The algorithm is run for defined episodes and episode length. Line 19 initializes the exploration noise. Line 22 observes each agent gets an action  $a_i$  from each deep policy. Line 23 calculates optimal Lagrange multiplier  $\lambda_i^*$  for each constraint. The original action  $a_i$  is passed to the safety layer, and Line 24 calculates the adjusted optimal action  $a_i^*$ , which will ensure safe navigation for the agents. Lines 25-26 observe the taken action and storing history in to replay buffer. Lines 28-35 are the learning step for each policy. After drawing the mini-batch (Line 29), Line 31 calculates the mean square error between the current critic's value and the

target critic's value and updates the critic. Lines 32-33 update each actor policy. Line 37 implements soft target critic update.

---

**Algorithm 1:** Safe-MADDPG for N agents
 

---

```

1 while  $epochs \leq TOTAL\ EPOCHS$  do
2   while  $step \leq TOTAL\ SAMPLE\ STEPS$  do
3     Reset Environment(state =  $o_i, \forall i \in N$ );
4     Calculate constraint values  $c_i$ ;
5     Execute actions ( $a_1, \dots, a_N$ ) and observe reward  $r_i$  and new obs  $o'_i$  for each
      agent  $\forall i \in N$ ;
6     Calculate constraint values  $c_i^{next}$ ;
7     Store ( $a_i, o_i, c_i, c_i^{next}$ ) in replay buffer D;
8   end
9   while update via mini-batch do
10    Sample a mini-batch for each agent  $i \in N$  : ( $a_i, o_i, c_i, c_i^{next}$ );
11    Observe safety layer :  $g_i(o_i, w_i)$ ;
12    Predict next constraint values;
13     $c_i^{next.predicted} = c_i + g_i(o_i, w_i)^T a_i$ ;
14    Calculate loss and update safety layer weights;
15     $L^k = \| c_i^{next} - c_i^{next.predicted} \|^2, \forall k \in K(\text{number of constraints})$ 
16  end
17 end

```

---

---



---

```

18 while episode number  $\leq$  TOTAL EPISODES do
19   Initialize exploration noise;
20   Reset environment (state =  $o_i$ ,  $\forall i \in N$ );
21   while timestep( $t$ )  $\leq$  TOTAL EPISODE LENGTH do
22     For each agent  $i$ , select action  $a_i = \mu_{i|\theta}(o_i) + \mathcal{N}_t$  w.r.t. the current policy and
        exploration;
23     calculate lagrange multipliers :  $\lambda_i^* = \frac{g(o_i, w_i)^T \mu_{i|\theta}(o_i) + c_i(o_i) - C_i}{g(o_i, w_i)^T g(o_i, w_i)}$ ;
24     correct each agent's action as :  $a_i^* = \mu_{i|\theta}(s) - \lambda_i^* g(o_i, w_i)$ ;
25     Execute corrected actions ( $a_1^*, \dots, a_N^*$ ) and observe reward  $r_i$  and new obs  $o'_i$ 
        for agent  $i$ ,  $\forall i \in N$ ;
26     Store ( $o_i, a_i^*, r_i, o'_i$ ) in replay buffer D;
27      $o_i \leftarrow o'_i$ ;
28     while agent  $i = 1$  to  $N$  do
29       Sampling of a randomized minibatch of  $S$  samples ( $o^j, a^j, r^j, o'^j$ ) from D;
30       set  $y_i^j = r_i^j + \gamma Q_i^{\mu'}(o_i^j, a_1^j, \dots, a_N^j)|_{a_i^j = \mu'_{i|\theta}(o_i^j)}$ ;
31       Update of the critic by minimization of the loss
          
$$L(\theta_i) = \frac{1}{S} \sum_j (y_i^j - Q_i^\mu(o_i^j, a_1^j, \dots, a_N^j))^2$$
;
32       Update of the actor by the use of sampled policy gradient;
33       
$$\nabla_{\theta_i} J \approx \frac{1}{S} \sum_j \nabla_{\theta_i} \mu_i(o_i^j)$$

34       
$$\nabla_{a_i} Q_i^\mu(x_i^j, a_1^j, \dots, a_N^j)|_{a_i = \mu_i(o_i^j)}$$
;
35     end
36     Update of the target network parameters for each agent  $i$ ;
37      $\theta'_i \leftarrow \tau \theta_i + (1 - \tau) \theta'_i$ ;
38   end
39 end

```

---



## Chapter 5

# Environment

---

**W**<sub>E</sub> have worked with two environments, Multi-Agent Particle Environments (MAPE) [6] which is based upon Open AI Gym [20]. It is a two-dimensional environment. Another is the DJI Mavic environment provided by Webots Simulation [7]. It is a three-dimensional environment satisfying real-world physics dynamics. DJI Mavic is a quadcopter having four propellers.

### 5.1 Multi-Agent Particle Environment (MAPE)

It is a simple multi-agent particle world comprised of continuous and discrete action space and essential simulated physics with observations integrated with rewards. The agents are configured to go towards their respective targets. The environment is set up using classes provided by Open AI Gym. The environment is open-sourced on GitHub. The action space is continuous and then each actor  $i, \forall i \in N$  will output action  $[a_i^x, a_i^y] \in \mathbb{R}^2$ . The locations of UAVs and their targets are randomized during the training for achieving the generalization of the algorithm. The positioning is done in the 2D coordinate environment whose scaling is ranging from  $-1.0$  to  $1.0$ .

## 5.2 Webots Simulation Environment

Webots is a 3D robot simulator. It provides all the features for robotic research, including prototypes for modeling robots, configuring single or multi robots through controllers, and transferring to real hardware. To provide the link between standard Deep RL algorithms and Webots Simulator, we have used Deepbots library [1]. It works on the emitter-receiver scheme of the simulator. The main components of the Webots simulator are :

- **World :** It is the main environment where simulation and training will take place. It consists of both object nodes, which can be static or spawnable. It consists of other information such as *global viewpoint*, *world info*, *cartesian coordinate system* (*NUE*, *EUN*, *NEU*) etc. In our case, *NUE* coordinate system is used which is represented as  $\{X, Z, Y\}$ . The scaling of the environment is set from  $-5.0$  to  $+5.0$ . The simulator is depicted in Fig. 5.1.

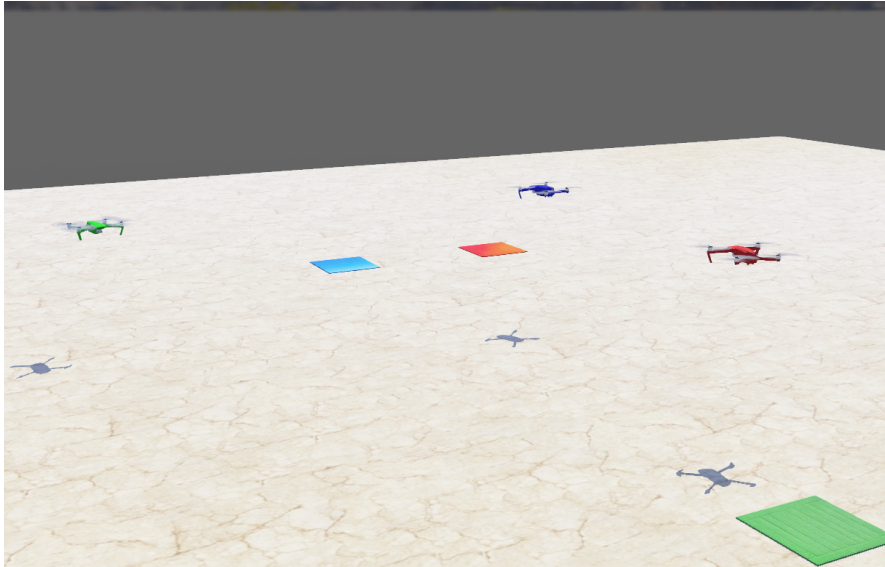


FIGURE 5.1: Webots Simulation

- **Supervisor Controller :** It is attached to the global world. It is the primary driver of the training. It helps in spawning the agents in the environment—simulation reset, observation, and action propagation, and reward distribution through its emitter and receiver nodes.

- Robot Controller : Each UAV is attached with a Robot Controller, and through its receiver node, it will receive action to take. It will propagate the following observation to the Supervisor Controller through its emitter node.
- Simulation Window : This is the main window where simulation and training happens. The timesteps can be run with three methods: step-by-step, real-time, or fast.

### 5.2.1 Combined Emitter-Receiver scheme

The Fig. 5.2 shows the working of the training in the Webots simulator through Supervisor and Robot Controller using Emitter-Receiver scheme. Here, the feedback abstraction of the standard RL loop is abstracted by both the controllers handling their respective tasks separately and integrating collectively.

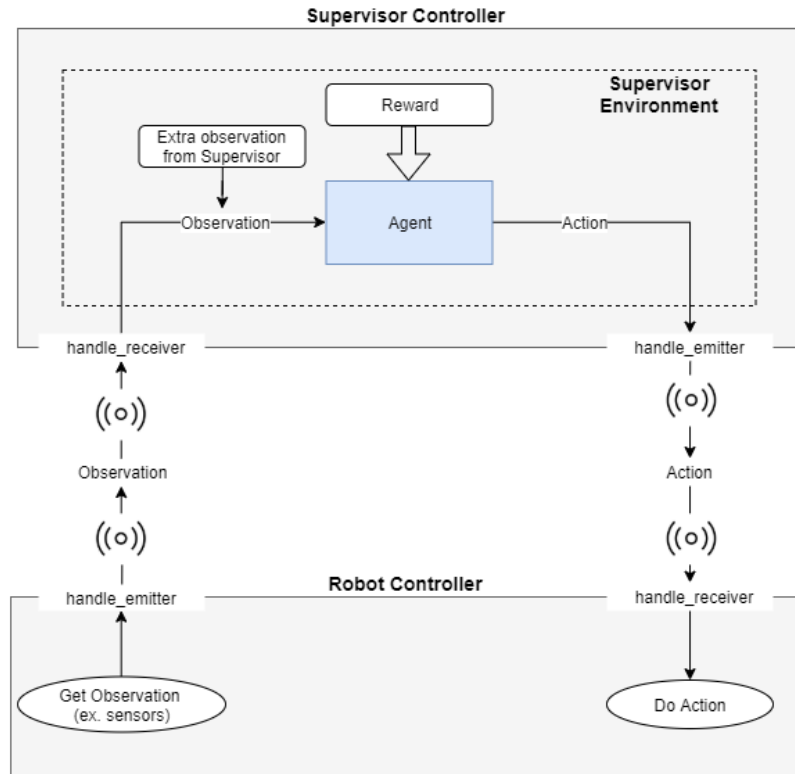


FIGURE 5.2: Deepbots integration [1]

### 5.3 Observation Space

#### 5.3.1 MAPE Environment

The following input features are passed into the agent neural network. Let the velocity and position of the agent be denoted as  $V^{\text{Agent}}$ ,  $P^{\text{Agent}}$ ; the velocity and position of other agents are denoted as  $V^{\text{OtherAgent}}$ ,  $P^{\text{OtherAgent}}$ ; and the position of the agent's target as  $P^{\text{Target}}$ . As it is a 2D environment, for an entity(UAV/Target)  $i$ ,  $P^{\text{entity}} : [p_i^x, p_i^y]$  and  $V^{\text{entity}} = [v_i^x, v_i^y]$ .

The observation space will be a concatenated tuple of  $\langle P^{\text{Agent}}, V^{\text{Agent}}, P^{\text{Target}}, P^{\text{OtherAgent}} \rangle$  and it is represented as:

$$o_i = \begin{cases} P_i^{\text{Agent(a)}} = [p_{i|a}^x, p_{i|a}^y] \\ V_i^{\text{Agent(a)}} = [v_{i|a}^x, v_{i|a}^y] \\ P_i^{\text{Target(t)}} = [p_{i|t}^x, p_{i|t}^y] \\ \sum_{j|j \neq i} P_j^{\text{Agent(oa)}} = [p_{j|oa}^x, p_{j|oa}^y], \forall j \in N \end{cases}.$$

where  $V \in \mathcal{R}^2$ ,  $P \in \mathcal{R}^2$ .

#### 5.3.2 Webots Environment

Each entity  $i$  (UAV/Target) in the environment has their position represented as  $P_i$  and movable entities will have their velocities represented as  $V_i$ . The observation of each agent consists of  $P^{\text{Agent}}$ ,  $V^{\text{Agent}}$ ,  $V^{\text{OtherAgent}}$  and  $P^{\text{Target}}$ . Other positions are relative to the current UAV. The  $P_i$  is a 3-element tuple describing the coordinates and for an entity  $i$  and it is represented as  $P_i^{\text{entity}} : [p_i^x, p_i^y, p_i^z]$ . And velocity is 6-dimensional tuple which represents linear and angular velocity along the three dimensions and for an entity  $i$  it is represented as  $V_i^{\text{entity}} : [l_i^x, l_i^y, l_i^z, a_i^x, a_i^y, a_i^z]$  where  $l_i$  and  $a_i$  are linear and angular velocity along the three dimensions respectively.

The observation space will be a concatenated tuple of  $\langle P^{\text{Agent}}, V^{\text{Agent}}, P^{\text{Target}}, P^{\text{OtherAgent}} \rangle$  and is represented as :

$$o_i = \begin{cases} P_i^{\text{Agent}(a)} = [p_{i|a}^x, p_{i|a}^y, p_{i|a}^z] \\ V_i^{\text{Agent}(a)} = [l_{i|a}^x, l_{i|a}^y, l_{i|a}^z, a_{i|a}^x, a_{i|a}^y, a_{i|a}^z] \\ P_i^{\text{Target}(t)} = [p_{i|t}^x, p_{i|t}^y, p_{i|t}^z] \\ \sum_{j|j \neq i} P_j^{\text{Agent}(oa)} = [p_{j|oa}^x, p_{j|oa}^y, p_{j|oa}^z], \forall j \in N \end{cases}.$$

where  $V \in \mathcal{R}^6$ ,  $P \in \mathcal{R}^3$ .

While training the simulation, UAV is first made to reach a certain threshold height of 1.0 meters. After that, the training loop starts. During the ascent phase, it will receive observations and taking steps without storing any history in *Experience Replay Buffer*. Each episode comprises ascent + learning. The entities' positions are randomized for generalization.

## 5.4 Action Space

### 5.4.1 MAPE Environment

The action space is continuous and then each actor  $i, \forall i \in N$  will output action  $[a_i^x, a_i^y] \in \mathcal{R}^2$ . The range of action value of each action coordinate is between  $-1.0$  to  $+1.0$ .

### 5.4.2 Webots Environment

The action space in Webots is 2 dimensional tuple consisting of *Pitch* and *Yaw* disturbance i.e  $a_i : [\text{pitch}_i, \text{yaw}_i]$ . *Pitch* is defined as rotation of the quadcopter around front to back axis. *Yaw* is defined as rotation of the quadcopter around the vertical axis. In our setup, *Roll* value is fixed through out the simulation and its value is set to  $1.0$ . The *Pitch* and *Yaw* disturbances are fed into the quadcopter through the Robot Controller. For each agent UAV, we have the deep policy as:

$$\pi_i \rightarrow \{\text{pitch}_i, \text{yaw}_i\}, \forall i \in N$$

## 5.5 Reward Structure

### 5.5.1 MAPE Environment

Let's suppose agent's current location is denoted by  $P^{\text{Agent}} : [p_{i|a}^x, p_{i|a}^y]$ . Let the position of the target is denoted by  $P^{\text{Target}} : [p_{i|t}^x, p_{i|t}^y]$ . To handle the landscape of the environment, we will define a constant  $C$ . The constant  $C$  will help to scale the environment landscape. As the scaling of the 2D environment is from  $-1.0$  to  $+1.0$ ,  $C$  here is set to  $0.1$ . After one Timestep, an agent will get a reward as listed in Table 5.1:

TABLE 5.1: Reward structure for 2D case (MAPE)

$-0.1 * \sqrt{(p_{i t}^x - p_{i a}^x)^2 + (p_{i t}^y - p_{i a}^y)^2}$	negative of the relative distance to the target
$+10$	if agent reaches the target
$-10$	if the agent collides with another agent

### 5.5.2 Webots Environment

The reward structure is a combination of distance penalty, collision penalty and angular penalty. The angle between the UAV and its target is calculated in each timestep and represented as a scalar value. It affects the *yaw* value as the UAV tries to position itself towards the target. If during the navigation, the UAVs collide en-route then a collision penalty is imposed. Distance value affects the pitch value necessary for driving the UAV to the target. Further previous distance(*prev\_dist*) and current distance(*curr\_dist*) (w.r.t. single Timestep) is the recorded and UAV motion is penalized accordingly. Distance between agent(a) and target(t) is calculated as :

$$\sqrt{(p_{x|a} - p_{x|t})^2 + (p_{y|a} - p_{y|t})^2 + (p_{z|a} - p_{z|t})^2}$$

The reward structure is listed in the Table 5.2 below:

TABLE 5.2: Reward structure for 3D case (Webots)

Reward Value	Conditions
-4	if the angle between agent and its target is $> 1.5$ Radians
-2	if angle $< 1.5$ Radians and $curr\_dist > prev\_dist$
-2	if angle $< 0.1$ Radians and $curr\_dist > prev\_dist$
+5	if angle $< 1.5$ Radians and $curr\_dist < prev\_dist$
+10	if angle $< 0.1$ Radians and $curr\_dist < prev\_dist$
-10	if there is a collision

## 5.6 Network Architecture

### 5.6.1 MAPE Environment

The actor UAVs are modeled with 2-layer Multilayer Perceptrons in PyTorch. The input to the actor-network will be the observation  $o_i, i \in N$ . For 3-Agent environment, it will be a 10-element tensor. So, the input to the neural network will be 10 dimensional. The number of neurons of the hidden layer is set to 64. We have used *Tanh* non-linearities for the network.

### 5.6.2 Webots Environment

The actor UAVs are modeled with 3-layer Multilayer Perceptrons in PyTorch. The input to the actor-network will be the observation  $o_i, i \in N$ . It will be a 30-element tensor for

the three-agent environment. So that will be input to the neural network. The number of neurons of the hidden layer is set to 64. The output from each network is 2-dimensional, which is then taken action. The centralized critic takes the joint action space for training. For non-linearity, *Relu* activation is applied to the hidden layers and *LeakyRelu* for the output layer of the network. Batch normalization is done before training with batch size of 1024 is used. And since it is a deterministic action algorithm, so OU noise has been set to provide exploration in the environment.

## 5.7 Network Hyperparameters

The tables below list the values of the various hyperparameters used for training the network. Table 5.3 lists the parameters which are the same in training both the MAPE and Webots environments. Next, Table 5.4 lists the parameters which are different for the two environments.

	MAPE / Webots
Training Threads	1
Tau	0.01
Learning Rate	0.01
Hidden Dims	64
Buffer Length	1e6
Final Noise Scale	0.0
Batch Size	1024

TABLE 5.3: Same Hyperparameters

	MAPE	Webots
Exploration Episodes	25000	2000
Init Noise Scale	0.4	0.3
Episode Length	60	1200

TABLE 5.4: Different Hyperparameters



## Chapter 6

# Evaluation

---

**I**N this chapter, we discuss the performance evaluation of both the environments.

### 6.1 MAPE environment

For MAPE, we conducted our experiments with the number of UAVs set to 3, 5 and 7. Also, we compared our approach (Safe-MADDPG) with MADDPG and PER-MADDPG.

#### 6.1.1 Mean Rewards

When the number of UAVs was set to 5, the reward structure is generated, as mentioned in Fig. 6.1. The Safe-MADDPG and PER-MADDPG are having higher rewards over the trained episodes than the MADDPG algorithm. Safe-MADDPG can avoid the collisions with other agents and hence better achieve its tasks effectively, thereby reaching the targets efficiently than other algorithms.

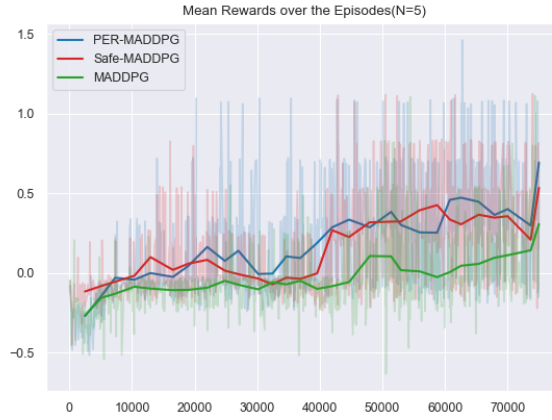


FIGURE 6.1: Mean Training Rewards (MAPE)

### 6.1.2 Inter-Agent Collisions

Next, we plotted collisions per 1000 episodes for all three algorithms. So, as mentioned in Fig. 6.2, the collisions in Safe-MADDPG are getting reduced over training time. Here also the number of UAVs is set to 5. Similar plot curves are obtained with 3 and 7 agents. Safe-MADDPG is efficiently learning the error correction over the episodes. However, for MADDPG and PER-MADDPG, the graphs are near equal range. The results are shown in Fig. 6.2 and the number of agents in the environment is set to 5.

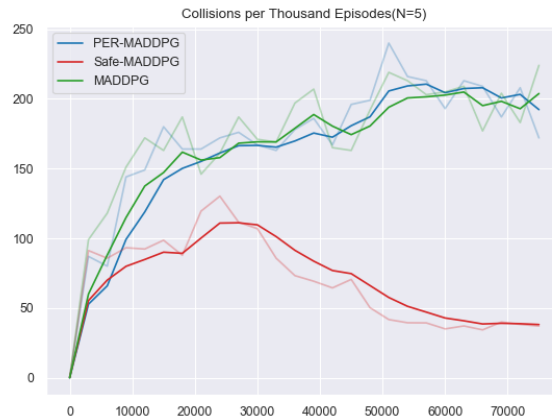


FIGURE 6.2: Collisions per 1000 episodes

Next, we compared the collisions per 1000 episodes w.r.t. number of agents in the environment (Fig. 6.3). The scaling of the environment is set as  $-1.0$  to  $+1.0$ . An increment is observed with respect to  $N$ . The main reason should be the scaling which is

set fixed for all the simulations. It is causing the environment to have narrow safe path for which collisions are not mitigated effectively. The results are shown in Fig. 6.3. The scaling of the agents is fixed for  $N = 3, 5$ , and  $7$ .

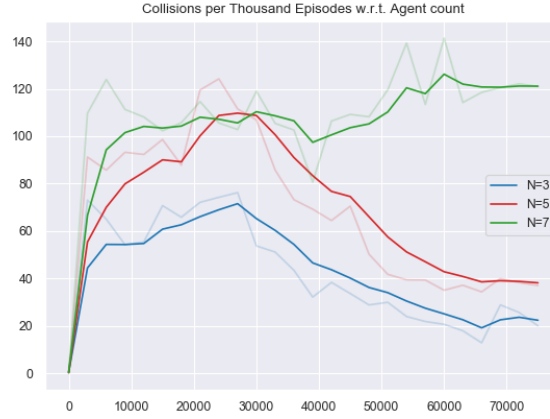


FIGURE 6.3: Collisions per 1000 episodes w.r.t. agent count

### 6.1.3 Testing Final Policies

Next, we compared the test-time collisions per 1000 episodes averaged over 5 runs w.r.t. number of agents. The scaling of the environment is set as  $-1.0$  to  $+1.0$ . The results are shown in Fig. 6.4. Here also, a similar increment is observed with respect to  $N$ . The main reason is the scaling which is set fixed for all the simulations. However, safe-MADDPG is performing better during test time.

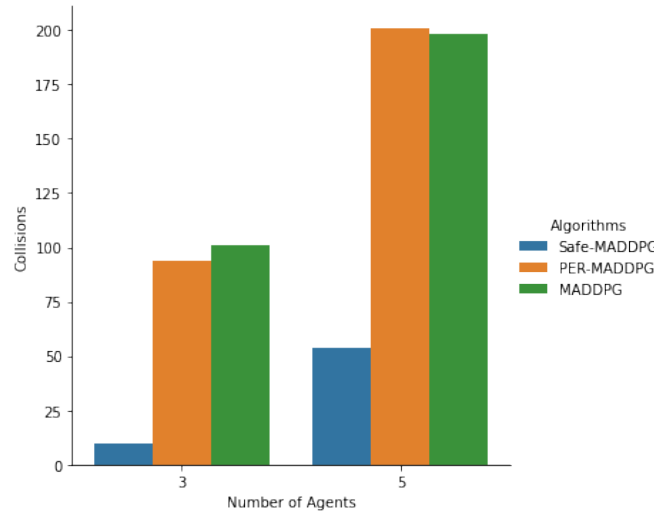


FIGURE 6.4: Test time collisions averaged over 5 runs

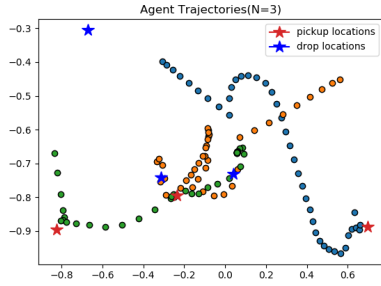


FIGURE 6.5: Maximum reward (N=3)

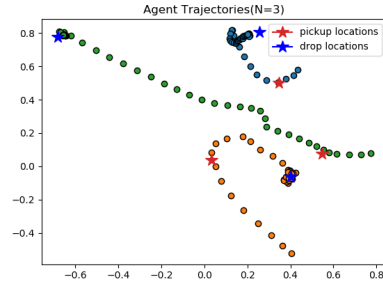


FIGURE 6.6: Minimum reward (N=3)

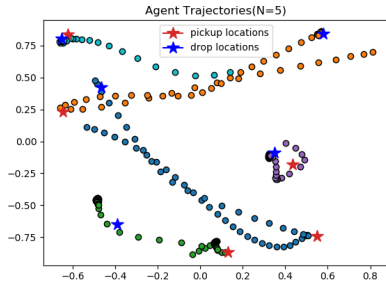


FIGURE 6.7: Maximum reward (N=5)

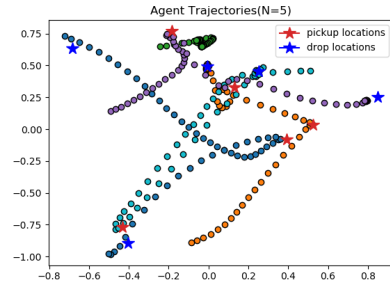


FIGURE 6.8: Minimum reward (N=5)

### 6.1.4 Trajectory Plotting

Next, we will discuss the safe behavior exhibit by the agents. We plotted the trajectories taken by the agents for  $N = 3$  and 5. Two plots are plotted for each  $N$ , one wherein an episode, the agent gets the least reward, and the other where the agent episode gets the maximum reward. When  $N = 3$ , it is visible that the green agent slightly diverts its direction while passing a nearby orange agent. It receives a maximum reward of 56.19 (Fig. 6.5) recorded during the evaluation episodes, and the lowest reward recorded is 11.02 (Fig. 6.6). For the latter, the agent's paths are quite congested. Similar behaviour is exhibited when  $N = 5$ . It receives a maximum reward of 88.02 (Fig. 6.7) recorded during the evaluation episodes, and the lowest reward recorded is 24.22 (Fig. 6.8).

## 6.2 Webots Environment

For the Webots environment, we formulated an environment consisting of three quadcopters and their target locations. The locations of the UAVs and their targets were randomized for better generalization.

### 6.2.1 Mean Rewards

The following Fig. 6.9 shows the mean rewards collected over the training. The mean value shows an upward trend indicating learning progress. Similar configurations with hyperparameter tuning have been tested, and the rewards graphs show a similar pattern. The hyperparameter tuning involves adjusting the batch size for training, setting the optimal learning rate, the value of  $\tau$  parameter, number of exploration episodes, and noise exploration percentage.

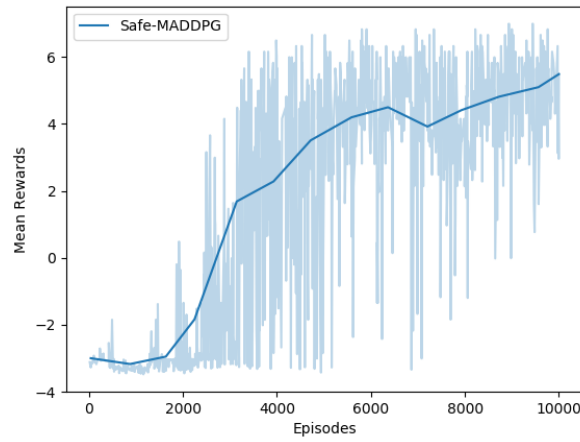


FIGURE 6.9: Mean Training Rewards (Webots)

The dependency on pitch and yaw disturbance is leading up to the formation of the graph plot. The value has a considerable effect on the motion of the UAV, as it is the deriving parameter for configuring the UAV in the direction of the target required. Here, the reward plot of a single agent is shown. Similar graphs are obtained for other agents in our three-agent environment.

### 6.2.2 Testing Final Policy

Next, we compared the test-time collisions per 1000 episodes averaged over 3 runs w.r.t. number of agents. It is tested with a three-agent environment. The scaling of the environment is set as  $-5.0$  to  $+5.0$ . The results are shown in Fig. 6.10. Here, Safe-MADDPG is performing better than MADDPG.

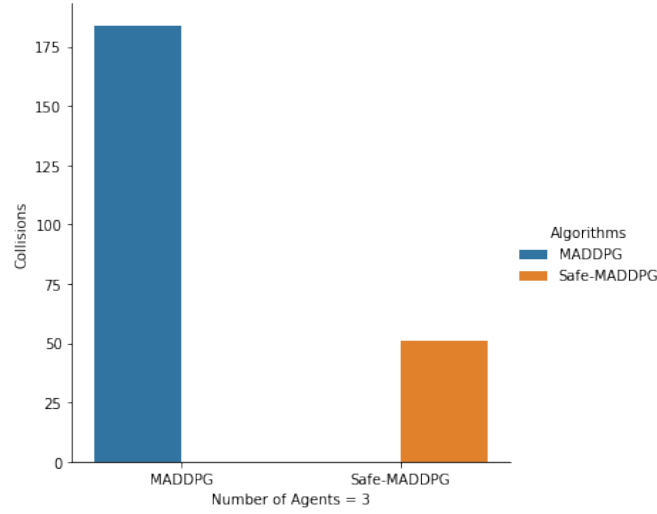


FIGURE 6.10: Test time collisions (Webots)

### 6.2.3 Controller Variation

Next, we will discuss the controller variation to the quadcopter with respect to the input *pitch* and *yaw* value. The value of *pitch* and *yaw* disturbances are plotted with respect to each Timestep. These are the values each policy network of the UAV will output. Each Robot Controller, which is attached to each quadcopter, will receive these values from the central Supervisor Controller and then process them using *emitter-receiver* scheme. The scaling of the environment is set from  $-5.0$  to  $+5.0$ .

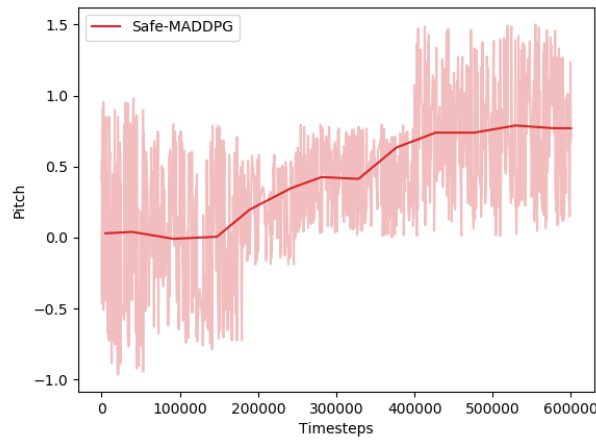


FIGURE 6.11: Pitch value

The *pitch* value graph in Fig. 6.11 shows that during the initial training phase, the values are unstable and start becoming positive mid-way and are mostly positive at the

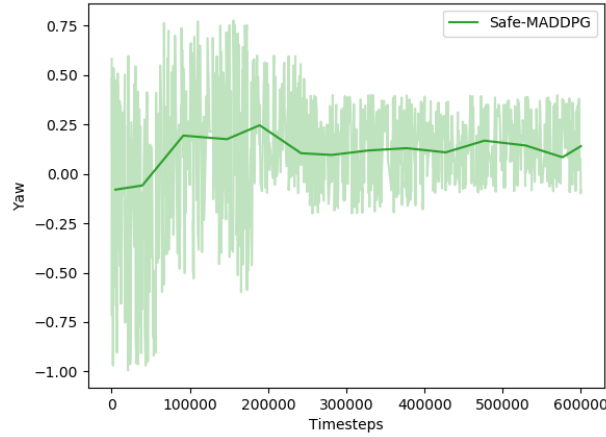


FIGURE 6.12: Yaw value

end of the learning. The *yaw* graph (Fig. 6.12) also indicates initial instability and then becomes centered around optimal value. This makes sense as the pitch is the primary driver of the *forward* motion of the quadcopter. During the evaluation, it is found out that the first quadcopter aligns it along with the target, focusing mainly on yaw (low positive pitch). Then, when it has positioned itself towards the target, there is a then high positive pitch (low yaw angle).

The number of *exploration episodes* and the *exploration noise* has a direct effect on the behavior of both plots. If the exploration is saturated to 0.0 very early in the training phase, then it will directly impact the early saturation of the *pitch* and *yaw* values.

#### 6.2.4 Value Function Comparison

Next we plotted the *critic's current* value against the *target* value (Fig. 6.13). It is calculated during training and done through using `torch.mean()` on the batch size. The batch size is maintained at 1024. The q-value tells how good a state is. As the graph shows an upward trend, the agent can find itself in good states as the training progresses towards completion.

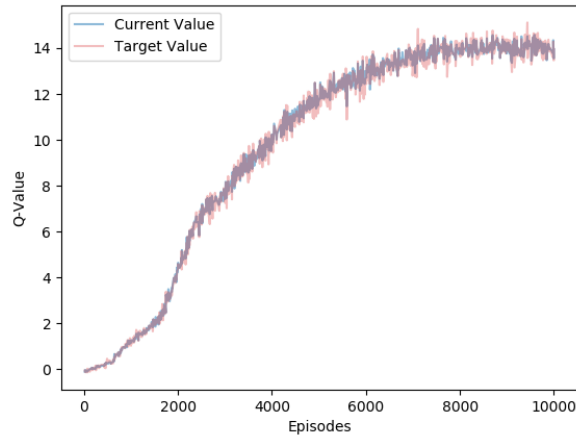


FIGURE 6.13: Current v.s. Target Q-value

### 6.2.5 Policy Loss and Critic Loss

Next, we plot the *policy loss* and *value loss* (Fig. 6.14). The value function loss remains very low (close to 0.0) throughout the training. Our previous graph shows that the current q-value remains close to the target q-value, leading to a low-value loss. The policy loss keeps getting low, indicating the agent learning the optimal behavior and reaching the destination using safe actions. It aligns with the reward graph; as the latter gets higher, the policy loss starts getting low.

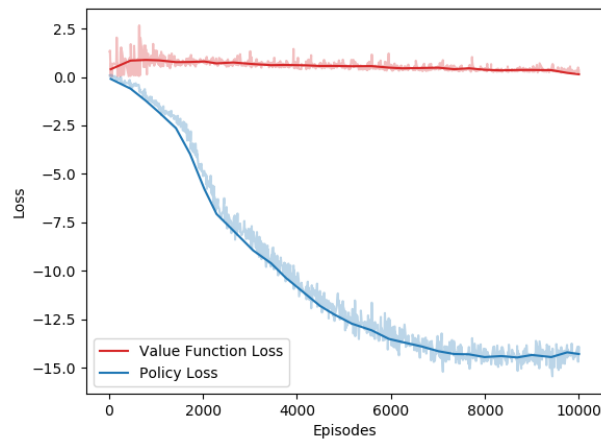


FIGURE 6.14: Policy and Value function loss



### 6.2.6 Trajectory Plotting

Next, we plotted the trajectory of the UAV by running a sample evaluation episode. Here, the UAVs are tasked with taking off their start positions, reaching their target, and hovering over them. Firstly, all the UAVs rise to their configured altitude of 1.0 meters. After that, the UAVs start moving towards the target, taking safe actions, thereby avoiding collisions with other UAVs flying mid-air. The trail of the paths taken is as depicted in Fig. 6.15. The ascent and descent of the UAV are made deterministic by default. The simulation video is available [here](#).

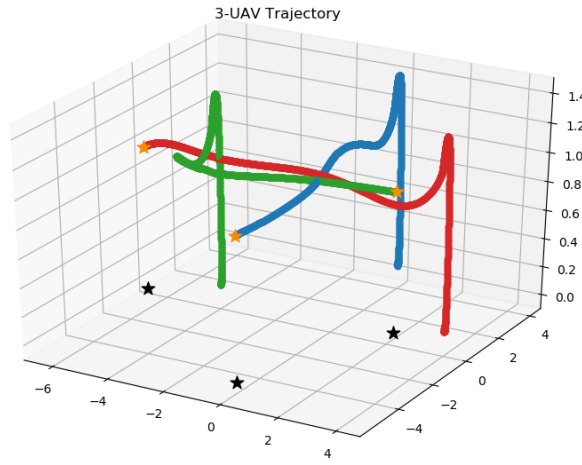


FIGURE 6.15: UAVs Trajectory Plot



# Bibliography

- [1] M. Kirtas, K. Tsampazis, N. Passalis, and A. Tefas, “Deepbots: A webots-based deep reinforcement learning framework for robotics,” in *Artificial Intelligence Applications and Innovations*. Cham: Springer International Publishing, 2020, pp. 64–75.
- [2] CNET. Ups, amazon delivery drones a step closer to reality with new us rules. [Online]. Available: <https://www.cnet.com/news/ups-amazon-delivery-drones-a-step-closer-to-reality-with-new-us-rules/>
- [3] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, “Continuous control with deep reinforcement learning,” *arXiv preprint arXiv:1509.02971*, 2015.
- [4] R. Lowe, Y. Wu, A. Tamar, J. Harb, P. Abbeel, and I. Mordatch, “Multi-agent actor-critic for mixed cooperative-competitive environments,” *CoRR*, vol. abs/1706.02275, 2017. [Online]. Available: <http://arxiv.org/abs/1706.02275>
- [5] G. Dalal, K. Dvijotham, M. Vecerik, T. Hester, C. Paduraru, and Y. Tassa, “Safe exploration in continuous action spaces,” *CoRR*, vol. abs/1801.08757, 2018. [Online]. Available: <http://arxiv.org/abs/1801.08757>
- [6] R. Lowe, Y. Wu, A. Tamar, J. Harb, P. Abbeel, and I. Mordatch, “Multi-agent actor-critic for mixed cooperative-competitive environments,” *Neural Information Processing Systems (NIPS)*, 2017.
- [7] Webots, “<http://www.cyberbotics.com>,” open-source Mobile Robot Simulation Software. [Online]. Available: <http://www.cyberbotics.com>
- [8] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, “Proximal policy optimization algorithms,” *CoRR*, vol. abs/1707.06347, 2017. [Online]. Available: <http://arxiv.org/abs/1707.06347>
- [9] 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.*, “Human-level control through deep reinforcement learning,” *nature*, vol. 518, no. 7540, pp. 529–533, 2015.
- [10] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, “Deterministic policy gradient algorithms,” in *International conference on machine learning*. PMLR, 2014, pp. 387–395.
- [11] T. Rashid, M. Samvelyan, C. S. De Witt, G. Farquhar, J. Foerster, and S. Whiteson, “Qmix: Monotonic value function factorisation for deep multi-agent reinforcement learning,” *arXiv preprint arXiv:1803.11485*, 2018.

- [12] R. Cheng, G. Orosz, R. M. Murray, and J. W. Burdick, “End-to-end safe reinforcement learning through barrier functions for safety-critical continuous control tasks,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, 2019, pp. 3387–3395.
- [13] D. Wang, T. Fan, T. Han, and J. Pan, “A two-stage reinforcement learning approach for multi-uav collision avoidance under imperfect sensing,” *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 3098–3105, 2020.
- [14] P. Long, T. Fan, X. Liao, W. Liu, H. Zhang, and J. Pan, “Towards optimally decentralized multi-robot collision avoidance via deep reinforcement learning,” 2018.
- [15] J. Collins, S. Chand, A. Vanderkop, and D. Howard, “A review of physics simulators for robotic applications,” *IEEE Access*, vol. 9, pp. 51 416–51 431, 2021.
- [16] K. Gamagedara, M. Bisheban, E. Kaufman, and T. Lee, “Geometric controls of a quadrotor uav with decoupled yaw control,” in *2019 American Control Conference (ACC)*. IEEE, 2019, pp. 3285–3290.
- [17] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour, “Policy gradient methods for reinforcement learning with function approximation,” in *Proceedings of the 12th International Conference on Neural Information Processing Systems*, ser. NIPS’99. Cambridge, MA, USA: MIT Press, 1999, p. 1057–1063.
- [18] N. Heess, G. Wayne, D. Silver, T. P. Lillicrap, Y. Tassa, and T. Erez, “Learning continuous control policies by stochastic value gradients,” *CoRR*, vol. abs/1510.09142, 2015. [Online]. Available: <http://arxiv.org/abs/1510.09142>
- [19] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, “Prioritized experience replay,” 2016.
- [20] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, “Openai gym,” 2016.