MCD411

B.Tech Project

Research Project

Cooperative Multi-Agent Reinforcement Learning for UAVs

November 21, 2021

Submitted By:

Rachit Jain
2018ME10032

Sadanand Modak
2018ME10039

Supervisor
Prof. Arnob Ghosh

Co-Supervisor
Prof. Shaurya Shriyam

Department of Mechanical Engineering

Contents

1 Introduction
- 1.1 Why Reinforcement Learning
- 1.2 Multi-Agent Reinforcement Learning
- 1.3 Networked Communication Systems
- 1.4 Decentralized Architecture & Consensus Update
- 1.5 Differential Policy in MARL

2 Literature Review

3 Project Objectives & Work Plan
- 3.1 Problem Statement
- 3.2 Objectives
- 3.3 MDP Formulation
- 3.4 Methodology
- 3.5 Gantt Chart

4 Work Progress
- 4.1 Motivation for Deep-MARL Solutions
  - 4.1.1 Failures of Traditional Approaches
- 4.2 Why Independent Architecture?
- 4.3 Privacy over Communication
- 4.4 Relevant Theory
  - 4.4.1 Multi-Agent Deep Deterministic Policy Gradient (MADDPG)
  - 4.4.2 Multi-Agent Soft Actor-Critic (MASAC)
  - 4.4.3 Independent Advantage Actor-Critic (IA2C)
- 4.5 Experimental Setup
  - 4.5.1 Packages
  - 4.5.2 Algorithms
  - 4.5.3 Experiments
  - 4.5.4 Experimental Setups
  - 4.5.5 Implementational Details

5 Results & Discussion
- 5.1 Centralised Training and Decentralised Execution
  - 5.1.1 Cooperative Navigation for 3 UAVs with 3 Target Locations
  - 5.1.2 Predator-Prey Environment with 3 collaborative UAVs, 1 adversary UAV and 2 non-breachable landmarks
  - 5.1.3 Physical Deception with 2 cooperative agents and 1 adversary agent
- 5.2 Decentralised Training and Decentralised Execution with Networked Agents
  - 5.2.1 CACC Slow-down Scenario
  - 5.2.2 CACC Catch-up Scenario

6 Conclusion & Future Work

7 References

Abstract

In the recent decade, the UAVs have gained a lot of importance, be it for defence missions or for maintaining law and order by enabling surveillance of an area. For efficient use of the UAVs, they have to be made autonomous as only then it would be possible for them to do complex tasks. To achieve the autonomous behaviour of UAVs, they can be trained prior to their deployment in unknown environments to take appropriate decisions under different situations. This opens up the door for the use of Reinforcement Learning to find the optimal behaviour of the UAVs. Multi-agent systems, although more challenging to implement, are far more efficient than single-agent systems. Thus, in this work, we create and simulate a Multi-Agent Reinforcement Learning system for cooperative behaviour of multiple UAVs in an unknown environment. For scalability, deep neural networks are used as value function approximators.

In collaborative scenarios, decentralized control policy based on local observations and communications with connected neighbours is evaluated in different environments. The concept of differential privacy is explored with use of noisy observations in the communication networks. Each agent makes individual decision based on the observations made individually and from the communication made with neighbours in the network. This paper finally judges the learnability of multiple agents in a more practical, noise-enabled decentralised fashion environment which is less explored in literature.

Chapter 1: Introduction

Unmanned aerial vehicles (UAVs) are quadcopter-type of autonomous flying vehicles that are generally used to carry out missions in unknown environments. UAVs are capable of performing a variety of tasks that were previously carried out by humans in dangerous and complex environments, such as wildfire monitoring, search and rescue missions, and target-tracking, searching, or attacking, and so on. This massive applicability of UAVs has given rise to rising interests of the research community in this field.

In context of UAVs, “multi-agent" means having a fleet of multiple UAVs autonomously operating in the same environment. Although a single UAV can achieve quite a lot many tasks and is not a trivial object, however, the scale of operations and the complexity of tasks in a real-world environment setup is so huge, that it highlights the need of developing multi-UAV systems. Such applications include exploration and mapping of an unknown and previously unmapped environment, building of a low-cost communication network over long-range distances, search and rescue missions and so on. Multi-UAVs systems could be in a cooperative setting, where all UAVs function as one cooperative team, while it could also be in a competitive setting, where each of the UAVs behave “selfishly” and compete against each other. This work focusses on the cooperative setting, since cooperative formations of UAVs have become an important means of improving efficiency of the system as a whole.

1.1 Why Reinforcement Learning?

As the exact mathematical model and the dynamics may not be available for the environment in which UAV is functioning, the standard Dynamic Programming algorithm, which caters to the problem of Planning in which the model of the environment is known, cannot be used as is here.

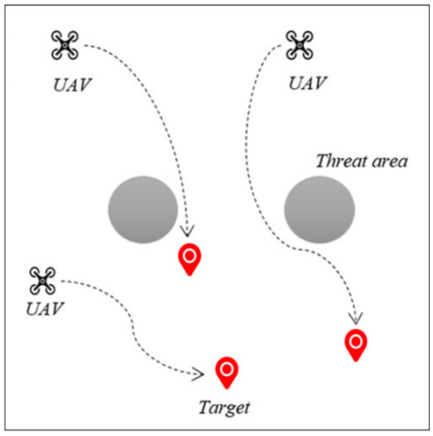


Fig: Multiple UAVs reaching targets [19]

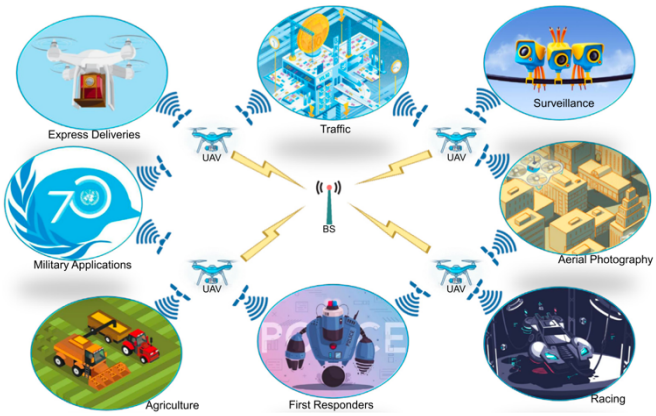


Fig: Applications of UAV Clusters [16]

This calls for the use of Reinforcement Learning (RL) techniques for the optimal control of the UAVs for enabling navigation in unknown and dynamic environments without having any prior knowledge of the environment. RL is a computational approach to learning from interaction with the environment, the so-called trial and error learning, which is a natural tendency of human beings, or for that matter any living creature. The two main components of RL are the agent and the environment. The environment is the world that the agent lives in and interacts with. The agent then decides on an action to take (based on a policy) based on observation of the state of the world it sees.

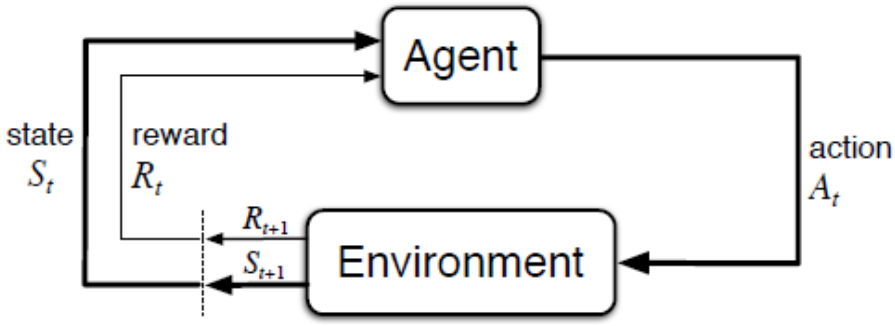


Fig: Basic RL Architecture [1]

The main distinction from supervised learning is the fact that it lacks a direct supervisor who tells "which action should be taken" under a given situation. The agent receives rewards for its "good" actions and gets "penalised" for the "bad" actions, which is the only supervisory feedback it gets from the environment. Thus, the central idea in the whole of RL is concerned with "learning of an optimal policy that maps states to actions" to guide the agent as to which action is preferable to be taken when in any state so as to get maximum reward.

1.2 Multi-Agent Reinforcement Learning

One natural extension to the single-agent RL setup, is the multi-agent RL setup which can then be used for multi-UAVs systems. However, with multi-agent setups, especially as we enter into real-world problems, the problem quickly and very easily becomes intractable due to the exponential increase in the size of state and action spaces. This need has given rise to the use of Deep Neural Networks (or any other general function approximators) to replace the traditional lookup tables used in medium to small scale single-agent RL setups. This combined field of Deep learning and Multi-agent Reinforcement learning is often referred to as the Multi-agent Deep Reinforcement Learning (MADRL) and is one of the hottest topics in RL research.

Single agent RL has been quite exhaustively explored and one may find a lot of papers on its use in control of a UAV. However, MADRL itself being quite a new field, its many possible applications to the multi-UAV systems hasn't been fully explored yet and the research in this area is limited. And since multi-UAVs have such dramatic advantages over a single-UAV in terms of perceptible area, and complexity of functions, it becomes essential to explore this area further. Therefore, this work focusses on cooperative multi-agent learning of UAVs using Deep Neural networks, for application in domains such as target assignment and path planning.

1.3 Networked Communication Systems

With multiple agents in a single environment, there are multiple learning scenarios that are possible; they could learn and execute collaboratively, with all information about states of the agents and observations available to other agents, or they could learn centrally but execute independently which allows them to learn from a united 'critic' model while different 'actor' strategies, or they could learn completely independently. The latter case would imply that the multi-agent reinforcement learning problem boils down to multiple single agent RL problems since each agent is independently present in the environment and the actions of other agents should affect the decision making of this system.

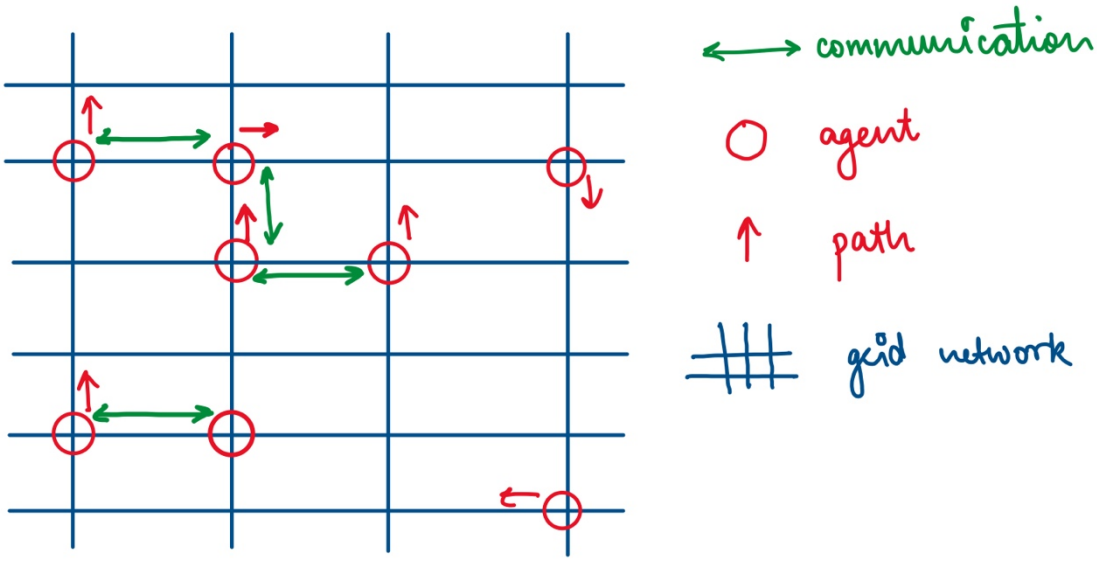


Fig: MARL setup with communication from neighbouring agents

Besides these common frameworks, one can also have a scenario where agents work independently, but under some local communication systems! This promotes the idea of learning with constrained communication from neighbouring agents and this work focussed heavily on this kind of communication in later sections.

1.4 Decentralised Architecture & Consensus Update

Generally in MARL problems, the formulation chosen for collaborative scenarios favour the choice of a central controller which receives the reward for all agents and determines the action for each agent on the basis of those observations received. This helps the system to achieve the global vision of improving the average episodic reward (AER) over number of episodes. However, for a decentralised formulation, it is important to understand that the decentralisation of the agents puts the central controller theory redundant.

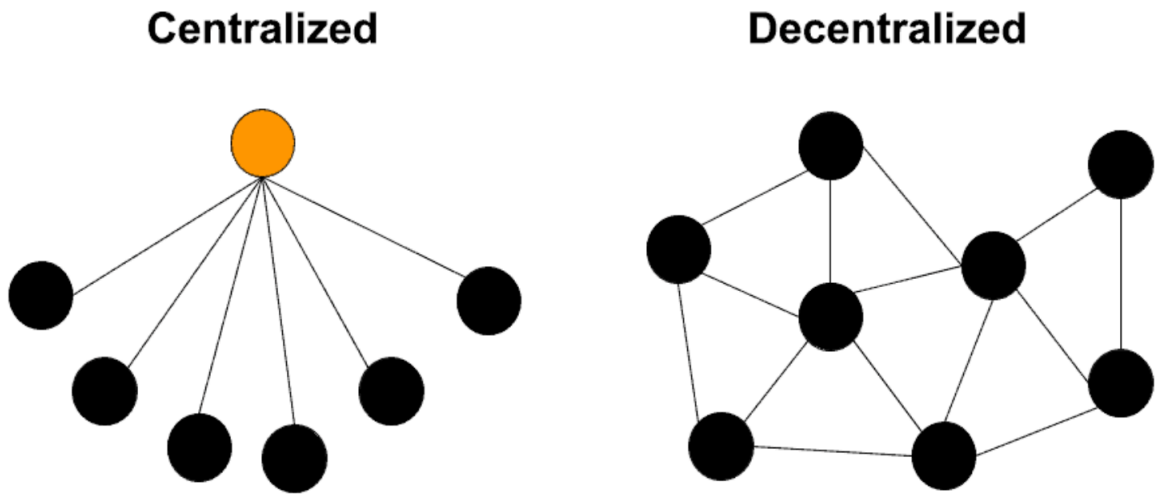


Fig: Centralized v/s Decentralized framework visualisation [28]

To take the observations of neighbouring agents into consideration, a consensus update is formulated over the communication done with these agents. In consensus policy, the observations coming in from all communication networks [28] are collectively viewed to generate the optimal policy for the current agent according to the ‘Consensus Policy’. This policy was developed primarily for blockchain for developing a system that makes update based on a consensus of multiple users, but has also received traction in the reinforcement learning domain.

1.5 Differential Policy in MARL

Differential privacy (DP) builds upon the concept of privacy, which when simply put, just refers to an individual's right to withhold some of the information with them and only share the part of the information that they deem safe to share with others. Data privacy, which is now a popular word nowadays, derives from this idea only.

Differential privacy is a framework or system which "technically" implements privacy to "technical" systems. More formally, it is a method of sharing information publicly about a dataset, by only describing the broad patterns of groups within it, and withholding the rest of the specifics (individual entries) of the dataset. For instance, given a dataset and a query on it and say the system produces the answer X for that query. Now, after removal of one of the entries of the dataset, say the answer that the system generates is Y. Now, if X and Y are equal, it means that the system has not learnt any specifics about that removed entry and has learnt only the broader patterns in the dataset, and thus, has implemented DP. Else, it is said that the system doesn't ensure DP.

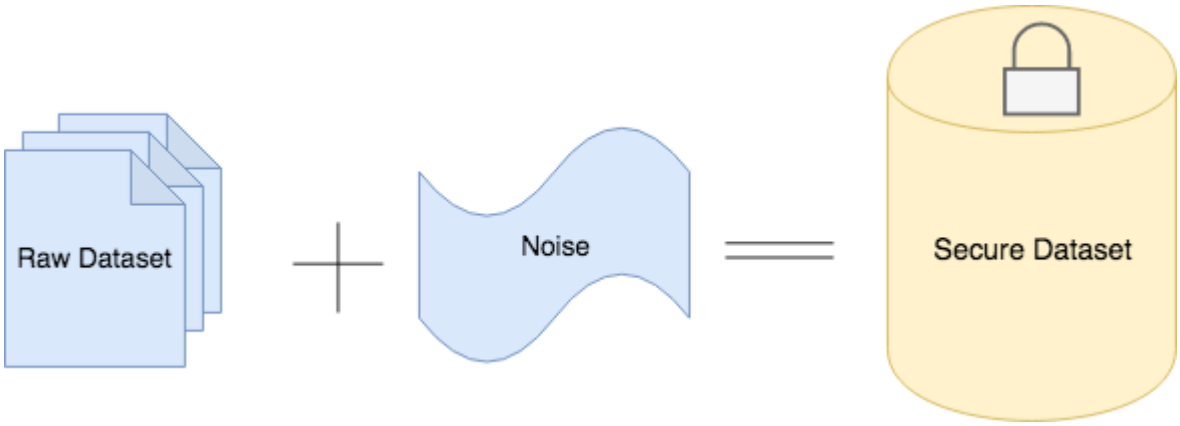


Fig: Noise added to secure the dataset [33]

More often than not, when DP is implemented in the Safe AI literature, the system is actually some "learning agent". For the MARL case, it is the different agents co-existing in the environment. The below figure shows the communication graph of agents, via which they are exchanging information about their states with one another. So adding noise to each of these communicating edges would ensure DP as none of the agents would receive exact information about the other agent's state.

The Laplacian noise works best with numeric entries, such as the observations in our case. Thus, in this work we make use of the noise drawn from Laplace distribution and add it to each of the neighbouring observations an agent receives.

$$\mathbf{Lap}(x|\mu, b) = \frac{1}{2b}\exp\left(-\frac{|x-\mu|}{b}\right)$$

There are two parameters that this noise takes: (1) the mean value μ, and (2) the $b > 0$ is the diversity, or the scale parameter that controls how wide the Laplacian distribution is.

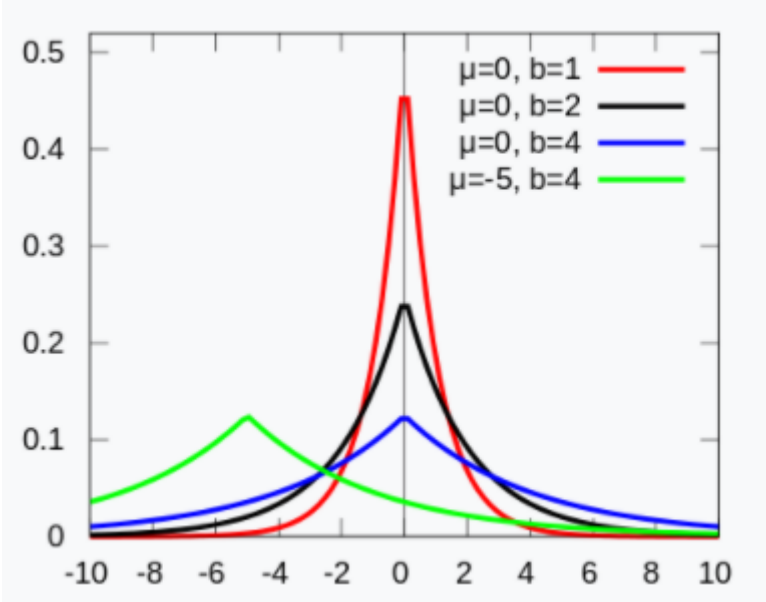


Fig: Laplacian Distribution at various parameter values [34]

The addition of random Laplacian noise to the agent's observations makes the problem even more realistic since in reality it is pretty natural to have imperfect communication due to communication over larger distances or unstable communication channels

Chapter 2: Literature Review

The basic structure of a general RL problem is formalized using the notion of a Markov Decision process that holds the Markov property that states that the next state and reward only depends on the most recent state, and not the entire history of states. The MDP is formalised as:

- S is the set of all valid states,
- A is the set of all valid actions,
- $R : S \times A \times S \to \mathbb{R}$ is the reward function, with $r_t = R(s_t, a_t, s_{t+1})$,
- $P : S \times A \to \mathcal{P}(S)$ is the transition probability function, with $P(s'|s, a)$ being the probability of transitioning into state s' if you start in state s and take action a,
- and ρ_0 is the starting state distribution.

Fig: MDP Description [13]

There are three main components of an MDP: state value function that conveys the goodness of a state, action-value function that represents how good it is to take a particular action in a particular state, and the policy which maps states to actions. The problem of learning the optimal policy is known as the control problem.

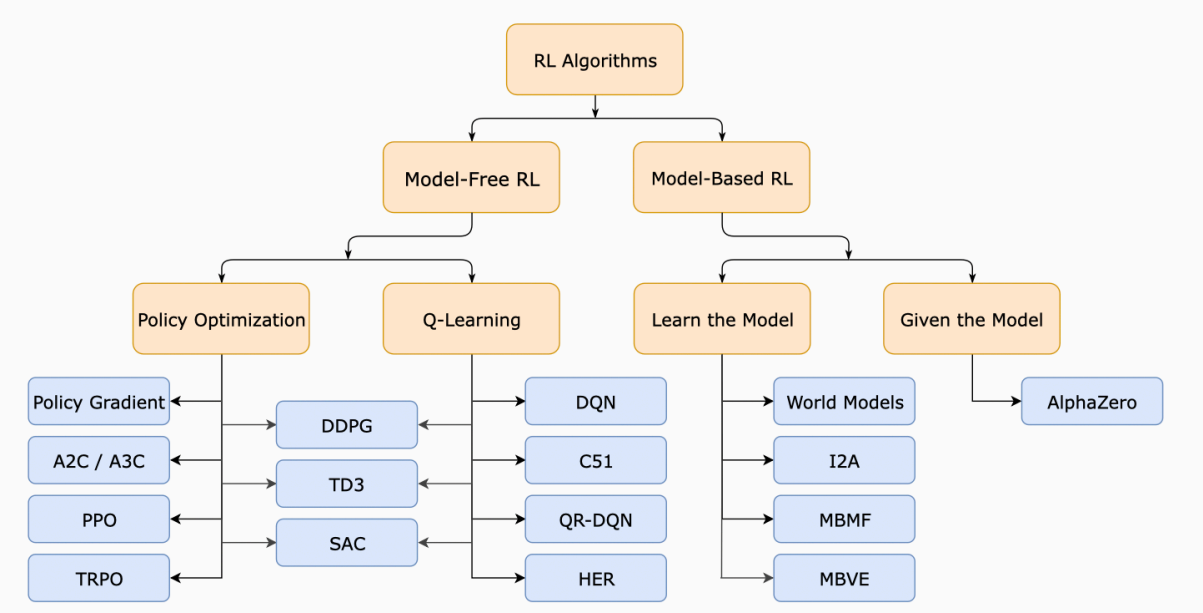


Fig: Taxonomy of DRL [13]

There have been quite a few recent developments in the field of Deep-RL. Broadly, the different algorithms proposed fall under one of the two categories: policy-based learning and value-based learning. In policy-based learning, the agent directly optimizes and learns the policy π parameterized by parameter vector Θ and this policy directly helps the agent choose actions. This is accomplished analytically calculating the gradient of the policy performance, called the policy gradient, then approximating the expectation using the sample trajectories and its returns, and then simply updating the policy network parameters by stochastic gradient descent.

$$\theta_{k+1} = \theta_k + \alpha \, \nabla_\theta J(\pi_\theta)\big|_{\theta_k}$$

$$\begin{aligned}\nabla J(\theta) &\propto \sum_s \mu(s) \sum_a q_\pi(s,a) \nabla \pi(a|s,\theta) \\ &= \mathbb{E}_\pi \left[\sum_a q_\pi(S_t,a) \nabla \pi(a|S_t,\theta) \right].\end{aligned}$$

Eqn: Policy Gradient

Monte Carlo Policy Gradient (also called REINFORCE) is the most basic algorithm based on the policy gradient optimization architecture.

Another way of doing control could be learning the value function for a particular policy using the Bellman expectation equation, and then improving upon that policy by acting greedily with respect to the value function.

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

Eqn: Bellman Expectation Equation

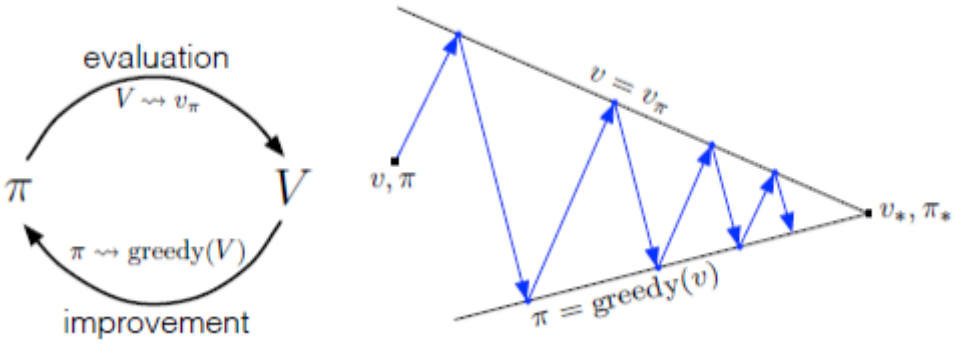


Fig: Generalised Policy Iteration [1]

However, a better approach to this could be directly utilizing the Bellman optimality equation, which is independent of policy, and directly learn the optimal value function.

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

Eqn: Bellman Optimality Equation

The value-based methods are primarily based on Q-learning method which was proposed by Watkins [22] and they tend to exploit this bellman optimality equation to directly learn the parameterized state-value function. However, sometimes Q-learning tends to be a bit unstable and so Mnih et al. [23] introduced the DQN algorithm which was one of the first steps in the combination of deep learning and reinforcement learning. Mnih et al. introduced two novel but

fundamental “tricks” to stabilize any algorithm that bootstraps, ie, the one uses an estimate of the subsequent value function (instead of the sampled return) to estimate the current value function: (1) experience replays, and (2) fixed Q-targets. These two “tricks” are used by many modern DRL algorithms. Here, experience replays means to feed the episodes again and again to the neural network model, instead of throwing it away after it is used once. And the fixed-Q targets basically refers to using two separate neural networks, where the parameters of one of them are continuously updated while they are fixed for some interval of time before they are updated for the other neural network.

QMIX [26] is a novel value-based method that proposes a centralized training-decentralized execution type of algorithm. In this, the global action value is calculated by using a separate mixing network to “mix” the individual action values of the agents. QMIX lies between the two extremes of IQL (independent Q learning), which decomposes a multi-agent problem into a collection of simultaneous single-agent problems that share the same environment, and a fully centralised Q-learning approach in which the execution of each of the agents is not independent.

However, there are many newer algorithms that retain the best of both worlds, as they learn both the policy as well as the value function, which keep updating each other to imitate somewhat of a Generalized Policy Iteration kind of framework. These are termed actor-critic methods, where the actor is the policy network, and critic is the value function network. A2C/A3C [24] is an example of one such popular actor-critic algorithm.

One of the state-of-the-art actor critic algorithms is the DDPG (Deep Deterministic Policy Gradient) algorithm [25]. It is an actor-critic algorithm which has Q-learning (critic) and Policy Gradient algorithm (actor) in an off-policy manner, i.e., the policy of concern is different than the policy from which actions are sampled by the agent. The actor (policy network) takes the state as input and outputs a continuous deterministic action. The critic (Q-value network) takes in state and action as an input and outputs the Q-value corresponding to it.

Applying traditional approaches to the UAV domain, Autonomous UAV Navigation [15] was one of the initial ideas to be explored in the reinforcement learning domain. It started to explore the dynamic nature of multi-agent UAV domain environment. Using a PID controller to control the parameters of the quadrotor, geometry based Q-Learning approach was incorporated with distance information to reduce the time to reach the target. However, this paper was far from being scaled as it used a 2D representation to simplify the environment and constraints. Thus, UAV moved at a constant altitude which is not a scalable approach to go forward with.

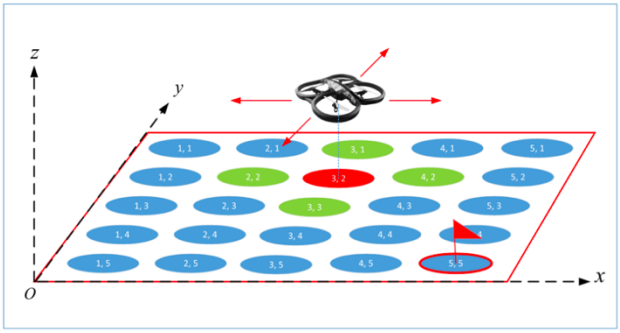


Fig: Quadrotor UAV in a 2D action space [15]

To push the limits from 2D to 3D action space, RL was applied to the bigger problem [17], but still using a single agent setup. Another paper [16] gave further flexibility to the UAV with large coverage and reliable transmission efficiency due to the wide applicability of UAV Clusters. This paper introduced the multi-agent problem in the reinforcement learning domains and asserted the necessity of cooperation and collaboration amongst agents to reach an optimal

solution. Deep RL was to be used to solve the dynamic allocation problem of wireless channel, so as to optimize the time delay of UAV data transmission.

To strengthen the idea of UAV clusters working as a team in a multi-agent setup, a new paper [18] proposed a distributed Multi-Agent Reinforcement Learning (MARL) algorithm for a team of UAVs The proposed MARL algorithm allows UAVs to learn cooperatively to provide a full coverage of an unknown field of interest while minimizing the overlapping sections among their field of views. It did a good job in bringing out the aspects to be taken care of while doing MARL, but did not use state-of-the-art neural or deep learning approaches. Rather, game-theoretic corelated equilibrium was used to solve the problem, with numerical function approximation techniques to save from the curse of dimensionality.

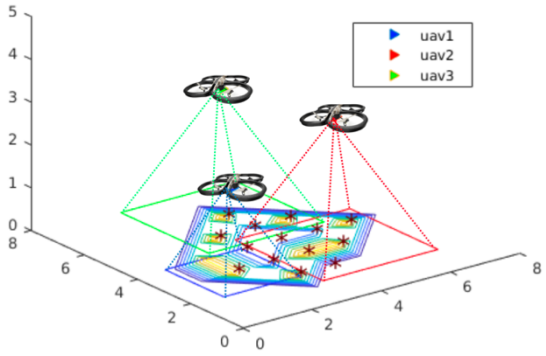


Fig: 3D rep of UAV Team covering field [18]

Compiling all ideas in the multi-UAV domain, a recent paper [19] applied MARL using 'deep' algorithms to solve for target assignment and path planning problems; two of the most critical problems to be solved in the UAV application (MUTAPP). The policy network uses a 2-layer 128-unit multi-layer-perceptron policies which supported real-time computations. Pre-defined altitude was pre-assigned to the UAVs to avoid collision. Decentralised execution with centralised training framework was carried out for the STAPP problem.

This last paper along with many ideas from other research papers in the literature review inspired a lot of implementation details that were used while experimentation for our use-case.

To simulate the independent framework with fully decentralized multi-agent reinforcement learning with local communication networks under Consensus Update, literature related to related environments and the possible algorithms that could be utilised were explored.

Recently, a novel decentralized approach to Multi-agent RL was proposed [35] in which the first-of-its-kind independent decentralized MARL algorithm was proposed named the IA2C (Independent Advantage Actor-Critic). In this, the training is decentralized and each agent gets to see the states of only the neighbouring agents. The scalability of this approach is pretty good and it lends itself to ATSC (Adaptive Traffic Signal Control) very well.

There are two kinds of policies in general in the MARL domain, (1) communicative, and (2) non-communicative policies [36]. The communicative policies, such as DIAL, CommNet, and NeurComm, are the ones that use global information of the states of all the agents, but with a delay, ie, this global information is available to each of the agents after some time-delay and not instantaneously. On the other hand, non-communicative policies, such as IA2C, ConseNet, and FPrint, are the ones that use only neighbourhood information, but that is available to each of the agents instantaneously.

Chapter 3: Project Objectives & Work Plan

3.1 Problem Statement

UAVs are capable of performing a variety of tasks previously done by humans in dangerous and complex environments, such as wildfire monitoring, search and rescue missions, and target-tracking, searching, or attacking, and so on. For complex applications, such as exploration and mapping of an unknown and previously unmapped environment, building of a low-cost communication network over long-range distances, or search and rescue missions, multi-UAV cooperative systems could prove to be very useful. Since the environment is unknown and the state and action spaces are so huge, it is necessary to use Multi-Agent Deep Reinforcement Learning methods such as MADDPG.

Since MADRL is itself a newly-emerging field, and the amount of research in the field of multi-UAV systems is limited, seeing the dramatic advantages of multi-UAV systems over a single-UAV in terms of perceptible area, and complexity of functions, it becomes essential to explore this area further. Therefore, this work aims at developing a cooperative multi-agent learning system of UAVs using Deep Neural networks, for application in domains such as target assignment and path planning.

Another focus point of the work was about exploring fully decentralised Multi-Agent Reinforcement Learning problem with communication from neighbouring agents in the network with the overall goal of maximising the globally averaged return over the network. This decentralized training and decentralized execution problem is extremely new in this RL field and literature for the same is quite limited. Besides, the practical application of noise enabled communication and its effect on the learnability of the agents was explored to gain insights into the different practical settings that could be implemented in related frameworks.

3.2 Objectives

Most traditional RL algorithm lack conditioning on actions taken by other agents, when deciding for their actions. This converts the non-stationarity of a dynamic environment into stationarity for each individual agent. Nevertheless, this poses significant problems if the parameters need to be changed for each agent, at every time step. Thus, advantages of policy gradient need to be incorporated, allowing backpropagation of return as every agent plays a role in the optimization function.

Since traditional algorithms like Q-Learning require the structure of the environment to be same during training and testing, there is a need for a general purpose Multi-Agent Actor-Critic algorithm to be developed. After exploring the utility and advantages of such algorithms, focus would be shifted on the decentralized architecture. Learnability of the agents in different environments is to be explored with varying the noises to analyse the effect of changed parameters in different settings.

In this light, these are the following objectives that we aim to achieve:

- Study the fundamentals of Reinforcement Learning and the state-of-the-art research in MADRL
- Execution and Simulation of Centralised algorithms (DDPG, MADDPG)
- Develop a system for cooperative multi-agent learning of UAVs using MADRL
- Exploring Decentralized Training with Decentralized Execution for Networked Agents with Consensus Update
- Execution and Simulation of Decentralised algorithms (IA2C) in different environments
- Evaluating effect of noise in communication channels on global average episodic rewards
- Brainstorming about delay in communication signals which can mimic practical settings more closely.

3.3 MDP Formulation

One of the first aims was to formulate the MDP for a general MARL problem for UAV application. The below table presents this MDP. For a fully-observed MDP, the observation space O will be the same as state space S i.e. S = O, while for a partially-observed MDP (POMDP), the observation O will only have perceivable location information of agents and targets.

State Space (S)	Agent Locations
	Target Locations
Observations (O)	Location of agents
	Target location as seen by the agent
Actions (A)	Moving to a connected node
Rewards (R)	Negative Reward for each time-step passed
	Negative Reward for collision with other agents
	Positive Reward if the target is achieved
Probability (P)	The probability that the agent transitions from one location to another depending upon states and actions of all agents

Fig: 3D rep of UAV Team covering field [18]

To see the realisation of this general MDP, we present it for two MARL problems:

Application 1: Target Tracking problem for UAVs
- S: Discrete space of joint locations of all UAVs and Targets in 2D
- O: Locations of other UAVs and Targets within field of view
- A: Discrete action space (left, right, forward, backward, stay)
- R: Loss for collision; Gain for target within FoV

Application 2: An intersection negotiation task with traffic from all directions
- S: Discrete space of joint locations of all cars (fully observed, hence S = O)
- A: Continuous action space for travel direction and discrete displacement along the chosen direction
- R: Loss for collision; Gain for distance travelled; Gain for reaching correct lane

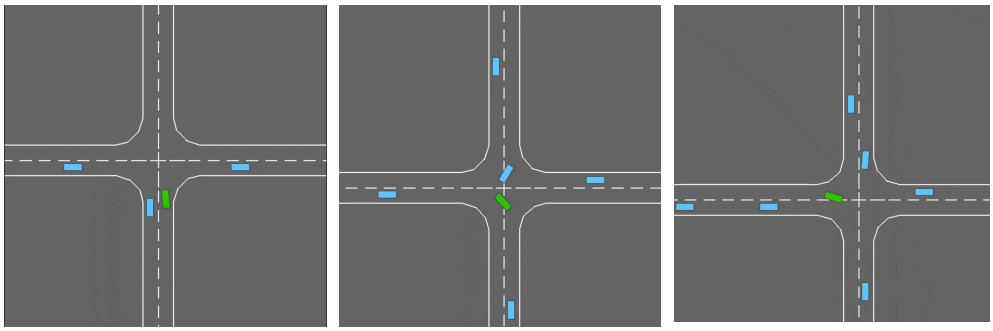


Fig: Intersection negotiation task representation

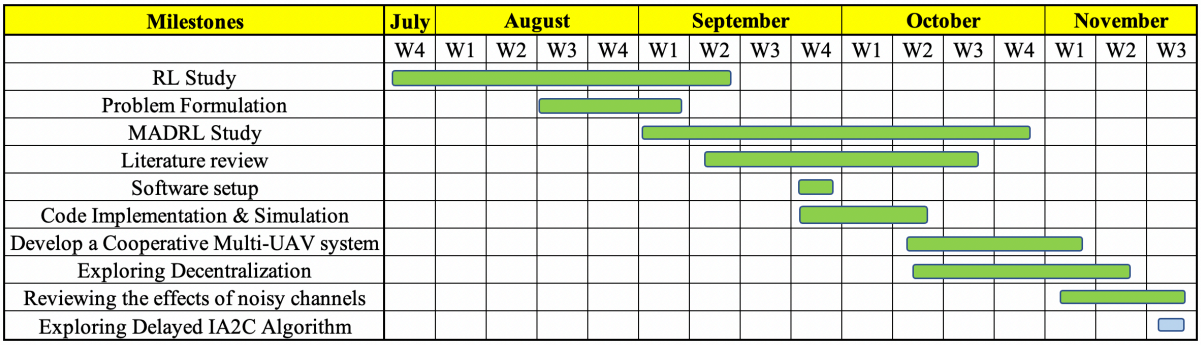
Along with simulating the multi-agent RL problem in the centralised training setting, our work aims to extend this application even further, by introducing noisy communication channels in some way and evaluating its effects on the overall learnability of the algorithms with this parametric change. This would be under the decentralised training and fully decentralised execution framework with networked neighbouring agents. The combination of both of these environments would allow us to judge the better algorithm and architecture for the Multi-UAV Cooperative problem we aim to deliver on.

3.4 Methodology

- Studied the single-agent Reinforcement Learning from a standard RL textbook (Sutton & Barto), followed by the study of the Deep-RL and MARL algorithms from various online resources and research papers
- Carried out the literature review of reinforcement learning applied to UAVs
- Simulated and analysed the MADDPG and DDPG algorithms [21] by training a model for 20,000 episodes for a simple multi-agent case, for both cooperative and competitive setups
- Aim to develop an algorithm, or in other words, adapt an existing Deep-RL algorithm to the multi-UAV setup using actor-critic method such as MADDPG for target assignment and path planning, in a partially-observable environment

- Exploring the possibility of having a Decentralized-Training and Decentralized-Execution of the above algorithm, with and without communication, taking inspiration from the works done on QMIX [26] and Zhang et. Al. [28]
- Introducing noise in the communication channels to explore the learnability of these systems for a more practical setting.

3.5 Gantt Chart



Chapter 4: Work Progress

4.1 Motivation for Deep-MARL Solutions

In the collaborative control system of application in UAVs, target assignment and path planning are two significant problem domains that need solving. The problem complicates further when multi-UAVs collaborate and/or compete in the same environment with numerous constraints to be satisfied. Alongside, simultaneous optimization of both problems (MUTAPP) needs sophisticated tools since step-wise real-time calculation for an optimal policy becomes infeasible. The dynamic nature of the environments requires global environmental information to be processed by all agents (UAVs).

Failure of Traditional Approaches

Due to multi-agents coming into the picture, each individual agent has to learn not only its actions as per the constraints posed by the environment, but also learn from the actions taken by other agents and making sure that the it follows all the constraints even in the dynamically changing environment.

To highlight the caveats in traditional approaches when extended to multi-agent and dynamic environment scenarios and to motivate the need of novel solutions; both the sides to the problem are considered below and in subsequent headings.

Non-Stationarity

As suggested previously, traditional methods in the reinforcement learning domain are challenged by the non-stationarity of the environment i.e. the change in the environment cannot be explained by the actions taken by the agents taken individually. Using a simple policy gradient approach can have high variance as the number of agents increases.

Curse of Dimensionality

The curse of dimensionality [4] comes into existence and makes collective training almost impossible with its exponential computational complexity. Return of each agent is correlated to the actions taken by the other agents and maximising the return of one agent individually is not possible. Hence, there is a need to develop a more robust and flexible algorithm.

Decentralised Execution

Apart from not being generalisable to multi-agent setups, most traditional model-free algorithms use approximation of the true action-value function Q^{π} for each agent independently. Traditional temporal difference learning and actor-critic algorithms need to be modified, else their application is generally infeasible in the multi-agent setting, especially when using decentralised execution with centralised training.[2]

4.2 Why Independent Architecture?

Having a central controller system is costly to install in a practical setting for many applications like sensor networks, intelligent transportation systems, etc. Also, communication is difficult to be executed from a single controller to all the agents and might even have problems of lag and mis-communication and this constant exchange and storage of information incessantly increases the communication overhead at a single controller which further reduces the scalability of the multi-agent system.

Under this decentralized structure, the action is taken individually by each agent without any information about the policies or actions of the other agents. For the critic step where the agent takes decision about the action, a consensus update via local communication from neighbouring agents is done over the network. The algorithms would be trained in an online fashion so that scalability is not an issue in the practical setting.

4.3 Privacy over Communication

Relating more with the practical setting, it is important to note that perfect communication between agents over communication channels is rather less frequent and there are always some unwanted disturbances that can prompt into the communication channels and disturb the information that is received by the agents, making some systems which are hugely dependent on the accuracy of the observations (ex: defence systems) out of control!

Additionally, there in many practical settings, the agents interacting in the environment do not wish to provide the complete information about their states and observations to other agents over communication channels. To incorporate some measure of privacy, these agents add statistical noise while communicating to a certain extent.

Differential privacy (DP) builds upon the concept of privacy, which when simply put, just refers to an individual's right to withhold some of the information with them and only share the part of the information that they deem safe to share with others. Data privacy, which is now a popular word nowadays, derives from this idea only. This has also been discussed in detail in the previous sections.

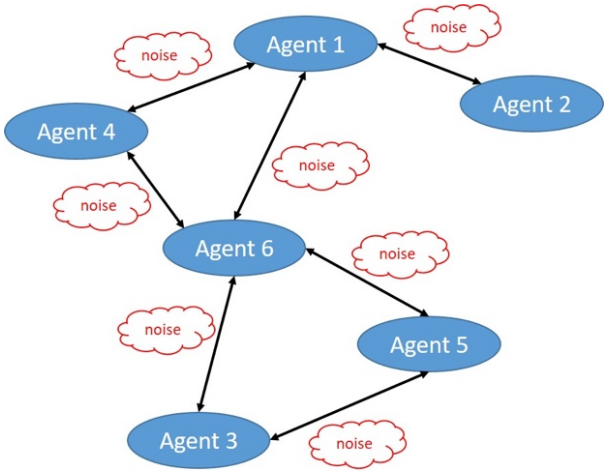


Fig: Differential Privacy [33]

With the aim to develop, simulate and evaluate more practical setups and having understood the motivation behind certain objectives and methodology involved in the work, we go onto discussing the relevant theory for the same.

4.4 Relevant Theory

4.4.1 Centralised Training and Decentralised Execution

In recent years, some algorithms have come into picture that make efforts to satisfy some of the objectives that we aim to implement related to exploring cooperative communication under de-centralised execution setting. Centralised training and de-centralised execution have been handled effectively by two such algorithms called MADDPG and MASAC. These have been discussed in detail below.

Multi-Agent Deep Deterministic Policy Gradient (MADDPG)

In Multi-Agent Reinforcement Learning (MARL) [2], some algorithms used decentralised approaches that effectively, reduce the system to a set of independent agents who undertake actions without considering the effect of actions of other agents. In this case, if collision has to be avoided, the algorithms needs to be modified to use a 'global master' that understands the changes in environment and teaches each of the agent about the actions taken by the other. However, the agents still execute independently, but train collectively. This is called centralised training and de-centralised execution. This is the core idea of MADDPG which uses deep deterministic policy gradient approach for the multi-agent setup, with individual actors doing execution i.e. taking actions, while a centralised critic, critiquing those actions based on the response taken from the global environment.

Algorithm 1: Multi-Agent Deep Deterministic Policy Gradient for N agents

for episode $= 1$ to M **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial state $\mathbf{x}$
  for $t = 1$ to max-episode-length **do**
    for each agent i, select action $a_i = \boldsymbol{\mu}_{\theta_i}(o_i) + \mathcal{N}_t$ w.r.t. the current policy and exploration
    Execute actions $a = (a_1, \ldots, a_N)$ and observe reward r and new state $\mathbf{x}'$
    Store $(\mathbf{x}, a, r, \mathbf{x}')$ in replay buffer $\mathcal{D}$
    $\mathbf{x} \leftarrow \mathbf{x}'$
    for agent $i = 1$ to N **do**
      Sample a random minibatch of S samples $(\mathbf{x}^j, a^j, r^j, \mathbf{x}'^j)$ from $\mathcal{D}$
      Set $y^j = r_i^j + \gamma\, Q_i^{\boldsymbol{\mu}'}(\mathbf{x}'^j, a'_1, \ldots, a'_N)\big|_{a'_k = \boldsymbol{\mu}'_k(o_k^j)}$
      Update critic by minimizing the loss $\mathcal{L}(\theta_i) = \frac{1}{S}\sum_j \left(y^j - Q_i^{\boldsymbol{\mu}}(\mathbf{x}^j, a_1^j, \ldots, a_N^j)\right)^2$
      Update actor using the sampled policy gradient:

$$\nabla_{\theta_i} J \approx \frac{1}{S}\sum_j \nabla_{\theta_i} \boldsymbol{\mu}_i(o_i^j) \nabla_{a_i} Q_i^{\boldsymbol{\mu}}(\mathbf{x}^j, a_1^j, \ldots, a_i, \ldots, a_N^j)\big|_{a_i = \boldsymbol{\mu}_i(o_i^j)}$$

    end for
    Update target network parameters for each agent i:

$$\theta'_i \leftarrow \tau\theta_i + (1-\tau)\theta'_i$$

  end for
end for

Fig: Psuedo-Code for MADDPG Algorithm [2]

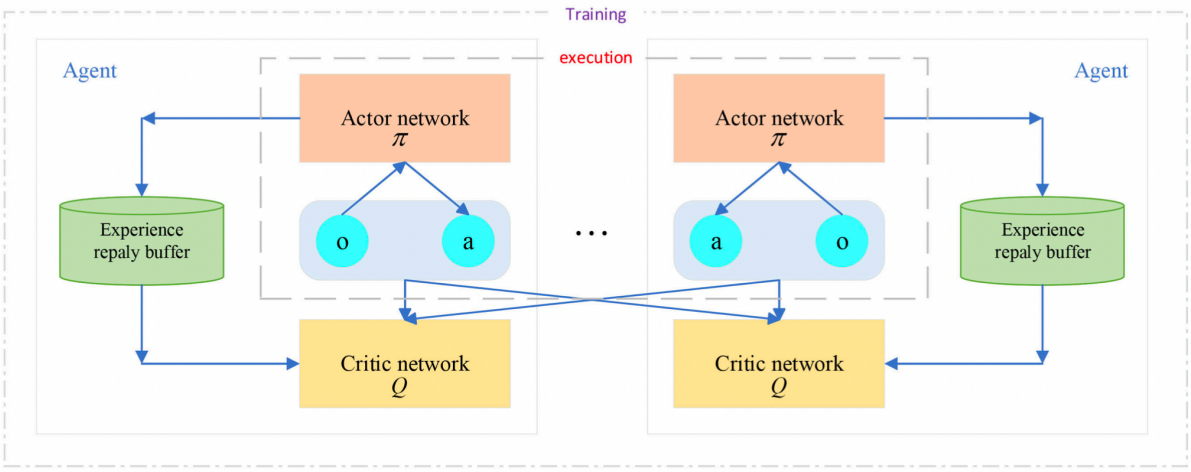


Fig: Centralised Training with De-centralised Execution [19]

Multi-Agent Soft Actor-Critic (MASAC)

Deep Reinforcement Learning combines the computational strength and flexibility of deep learning algorithms and the decision-making of reinforcement learning, and is currently a hot research topic in the field of artificial intelligence.[5] While MADDPG is the most popular model-free MARL, it is found to be much faster on learning than traditional algorithms, but can be further optimised, along with faster convergence speed which is hindered due the deterministic single action output of policy network. MASAC, on the other hand, makes each agent's policy network output action with a random strategy and propose a MADRL algorithm based on maximum entropy.[6]

Algorithm 1: Multi-agent soft actor-critic algorithm

Initialize parameters θ^1, θ^2 and ϕ_i $\forall i \in \mathcal{I}$
$\bar{\theta}^1 \leftarrow \theta^1$, $\bar{\theta}^2 \leftarrow \theta^2$, $\mathcal{D} \leftarrow \emptyset$
repeat
  for each environment step **do**
    $a_{i,t} \sim \pi_{\phi_i}(a_{i,t}|s_{i,t})$, $\forall i \in \mathcal{I}$
    $\boldsymbol{s}_{t+1} \sim \mathcal{P}_i(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t, \boldsymbol{a}_t)$, where $\boldsymbol{a}_t = \{a_{1,t}, \ldots, a_{N,t}\}$
    $\mathcal{D} \leftarrow \mathcal{D} \bigcup \{\boldsymbol{s}_t, \boldsymbol{a}_t, r(\boldsymbol{s}_t, \boldsymbol{a}_t,), \boldsymbol{s}_{t+1}\}$
  end for
  for each gradient update step **do**
    Sample a batch of data, $\mathcal{B}$, from $\mathcal{D}$
    $\theta^j \leftarrow \theta^j - \iota_Q \nabla_\theta J_Q(\theta^j)$, $j = 1, 2$
    $\phi_i \leftarrow \phi_i - \iota_\pi \nabla_{\phi_i} J_{\pi_i}(\phi_i)$, $\forall i \in \mathcal{I}$
    $\alpha \leftarrow \alpha - \iota_\alpha \nabla_\alpha J_\alpha(\alpha)$
    $\bar{\theta}^j \leftarrow \tau\theta^j + (1-\tau)\bar{\theta}^j$, $j = 1, 2$
  end for
until convergence

Fig: Psuedo-Code for MASAC Algorithm [6]

4.4.2 Decentralised Training and Decentralised Execution with Networked Agents

Having motivated the utility of independent learning agents in the previous section, recent literature related to decentralised training was further explored. In constrast to the literature that aims to reduce the problem of multiple agents into multiple single agents, our work aims to utilise the former problem and introduce local communication amongst neighbouring agents. This allows the problem to be formulated as a MARL problem and keeps the decentralized motives into check.

A couple of algorithms exist that can be modified to be utilized in this setting. Some of the IA2C algorithms that contain non-communicative policies which utilize neighbourhood information only are PolicyInferring [29], FingerPrint [30] and ConsensusUpdate [31]. In general, the motivation behind using IA2C algorithm is discussed below.

Independent Advantage Actor Critic (IA2C):

Recently, a novel decentralized approach to Multi-agent RL was proposed [3s] in which the first-of-its-kind independent decentralized MARL algorithm was proposed named the IA2C (Independent Advantage Actor-Critic). In this, the training is decentralized and each agent gets to see the states of only the neighbouring agents. The scalability of this approach is pretty good and it lends itself to ATSC (Adaptive Traffic Signal Control) very well.

Algorithm 1 Advantage actor-critic - pseudocode

// Assume parameter vectors θ and θ_v
Initialize step counter $t \leftarrow 1$
Initialize episode counter $E \leftarrow 1$
repeat
  Reset gradients: $d\theta \leftarrow 0$ and $d\theta_v \leftarrow 0$.
  $t_{start} = t$
  Get state s_t
  repeat
    Perform a_t according to policy $\pi(a_t|s_t;\theta)$
    Receive reward r_t and new state s_{t+1}
    $t \leftarrow t+1$
  until terminal s_t **or** $t - t_{start} == t_{max}$
  $R = \begin{cases} 0 & \text{for terminal } s_t \\ V(s_t,\theta_v) & \text{for non-terminal } s_t \text{ //Bootstrap from last state} \end{cases}$
  for $i \in \{t-1,\ldots,t_{start}\}$ **do**
    $R \leftarrow r_i + \gamma R$
    Accumulate gradients wrt θ: $d\theta \leftarrow d\theta + \nabla_\theta \log \pi(a_i|s_i;\theta)(R - V(s_i;\theta_v)) + \beta_e \partial H(\pi(a_i|s_i;\theta))/\partial\theta$
    Accumulate gradients wrt θ_v: $d\theta_v \leftarrow d\theta_v + \beta_v(R - V(s_i;\theta_v))(\partial V(s_i;\theta_v)/\partial\theta_v)$
  end for
  Perform update of θ using $d\theta$ and of θ_v using $d\theta_v$.
  $E \leftarrow E+1$
until $E > E_{max}$

Fig: Psuedo-Code for IA2C Algorithm [35]

4.5 Experimental Setup

4.5.1 Centralised Training and Decentralised Execution

To satisfy a part of the requirements involved in our problem statement, it was imperative to try and simulate some reinforcement learning architecture. With the meteoric rise in the acceptance of deep learning setups and their application in almost every machine learning task, reinforcement learning has also been deeply connected with the same. Also, with the advent of multi-agent task analysis coming into the bigger picture in the last 5-6 years, algorithms that were previously limited to single agent are being modified and developed for multi-agent setups in recent times in the literature.

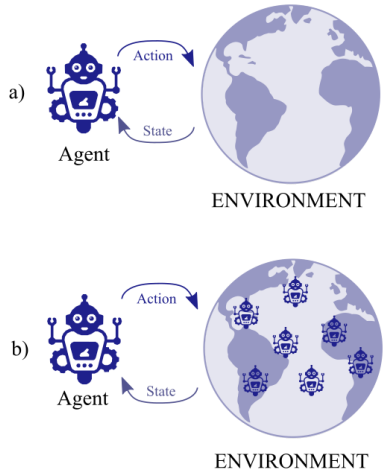


Fig: In multi-agent reinforcement learning, each agent perceives the environment to be consisting of other agents as well, which makes the environment dynamic due to the changing actions of all the agents

Packages

To experimentally implement and simulate such deep reinforcement learning algorithms, the following packages have found to be vital. Apart from these, each algorithmic implementation requires some other requirements which had to be installed in a virtual environment inside Python due to the variety of dependencies that exist for each.

1. Python >= 3.6
2. Gym >= 0.10.5
3. PyTorch >= 1.4
4. Tensorflow >= 1.1
5. Pytest >= 6.1.2
6. Pyglet >= 1.2.0

Algorithms

Since 2018, there have been a variety of multi-agent RL algorithms in literature and implementing them to our use-case was the main task. The objective here was not to compare and contrast existing algorithms for their pros and cons, but rather to simulate the applicability of such algorithms in the UAV problem setup and to get hand-on implementational

understanding of the algorithms in literature. The following algorithms were chosen to be used for the problem:

1. Multi-Agent Deep Deterministic Policy Gradient [2]
2. Multi-Agent Temporal Difference Deep Deterministic (MATD3) [7]
3. Multi-Agent Soft Actor-Critic (MASAC) [8]
4. Multi-Agent Distributed Distributional Deep Deterministic Policy Gradient (MAD4PG) [9]
5. Monotonic value function factorisation (QMIX) [10]

Name	Recurrent	Continuous	Discrete	Centralised training	Communication	Multi Processing
MADQN	✓	✗	✓	✓	✓	✓
DIAL	✓	✗	✓	✓	✓	✓
MADDPG	✓	✓	✓	✓	✗	✓
MAD4PG	✓	✓	✓	✓	✗	✓
MAPPO	✗	✓	✓	✓	✗	✓
VDN	✗	✗	✓	✓	✗	✓
QMIX	✗	✗	✓	✓	✗	✓

Fig: Overview of functionalities that most MARL algorithms possess [11]

Experiments

As can be observed, all the above algorithms sport 'centralised training' which is the main functionality they possess. Using this to our advantage, we replicate the algorithms given in literature and modify them to implement for our use-case. The most useful GitHub resource was the official repository of OpenAI which provided state-of-the-art reinforcement learning design framework.[13] It supports a variety of scenarios i.e. environments that can be tweaked to a particular use-case.

We used these OpenAI scenarios [13] [19] to model three main reinforcement learning problem that were extended to the UAV application. Implementation of the various papers reviewed in the literature review were chosen from PaperwithCode [20] to get original implementational codes to learn, train, and model our problem on. Online resources were extensively analysed to find good working solutions for MARL problems in general.[21] A brief introduction to each of the problems is as following:

1. **Predator-Prey Movement:** N slower cooperating agents chase the faster adversary agent around a randomly generated environment with L large landmarks impeding the way. The agents are penalised if they collide with each other, while rewarded when they do so with the adversary.

2. **Cooperative Navigation:** In this environment, agents cooperate through physical actions to reach a set of L landmarks. Each agent observes the relative dynamic position

of other agents, along with that of the landmarks, and are rewarded on the basis of their 'closeness' to the landmarks present. To avoid the case that all N agents occupy the same landmark, rewards are calculated for each agent, per landmarks. The agents learn to infer the landmark they must cover, and move there while avoiding other agents.

3. **Physical Deception**: Here, N agents cooperate to reach a single target landmark from a total of N landmarks. The adversary agent competes against the cooperative agents to get to the special target faster. Thus, the cooperating agents, are rewarded also rewarded if they are successful in deceiving the adversary for which target is the special one, along with other distance metrics and collision parameters being used.

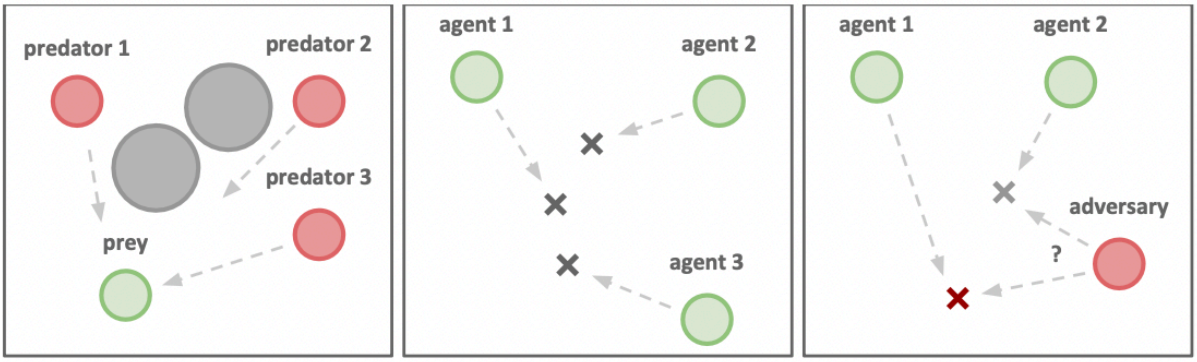


Fig: The scenarios for Predator-Prey, Cooperative Navigation and Physical Deception [19]

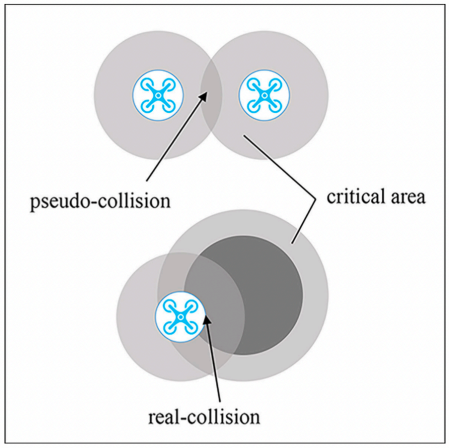


Fig: Critical area, pseudo-collision and real-collision [19]

In all the above scenarios, agents were translated to be UAVs for our problem and the collisions meant heavy penalisation. The threat areas were modelled as locations which these UAVs cannot breach and the landmarks were target locations for these systems. The above scenarios were chosen since they allow cooperative, competitive and mixed environments to be tested for our problem.

Implementational Details

For each scenario, a set of N UAVs were chosen, along with some adversary ones in case of competitive scenarios. These agents are allowed to move 25 steps in each training episode to figure out the environment. One episode is defined as completion of these apriori defined number of steps for each agent. The total number of training episodes was modified as per the computational complexity of the problem, while ensuring that the qualitative analysis could be done based on the results achieved. Generally, the number of episodes was kept to be 20,000 for each scenario. The average returns were copied into a text file to later analyse them with respect to the training episodes.

4.4.2 Decentralised Training and Decentralised Execution with Networked Agents

Next, we moved onto evaluating the other segment of the problem which involved the training to be decentralised as well, using a Consensus policy over the communication in neighbouring agents. For this, IA2C algorithm ConsensusUpdate was chosen for the Traffic Signal Control environment which is discussed in detail later. Cooperative adaptive cruise control (CACC) includes multiple concepts of communication-enabled vehicle following and speed control [32] and are the most applicable scenarios for our setting. These environments have been discussed in detail later.

Communication provides enhanced information so that vehicles can follow their predecessors with higher accuracy, faster response, and shorter gaps; the result would be enhanced traffic flow stability and possibly improved safety. A further distinction is made between CACC, which uses constant-time-gap vehicle following and automated platooning, which uses tightly coupled, constant-clearance, vehicle-following strategies.

This motivates our consensus-based MARL algorithms that leverage the communication network to diffuse the local information, which fosters collaboration among the agents.

Packages

To experimentally implement and simulate the IA2C algorithm discussed with the chosen environments made available on the GitHub repository over [28], the following packages have found to be vital. Apart from these, each algorithmic implementation requires some other requirements which had to be installed in a virtual environment inside Python due to the variety of dependencies that exist for each.

1. Python >= 3.5
2. Tensorflow >= 1.12.0
3. Tensorboard >= 1.15.0
4. SUMO >= 1.1.0

and other additional packages based on the version of above packages installed and the environment being run.

Experimental Setups

The Cooperative Adaptive Cruise Control (CACC) environment as used in [36], is adapted in our work to two scenarios:

- **Catchup**: In this, all UAVs are initially moving at very slow speeds, and they need to speed-up to attain the desired headway and speed targets
- **Slowdown**: In this, all UAVs are initially moving at very high speeds, and they need to slow-down to attain the desired headway and speed targets

Very understandably, the catchup environment is easier for the agents to learn and they would not be subjected to large number of collisions in it, since they are moving at slower speeds initially. Thus, there would be relatively higher rewards received in catchup as compared to slowdown, in which there would be larger negative rewards for the agents due to frequent collisions.

In both the CACC environments, the state of any agent is defined by the headway, velocity and acceleration, more specifically, by a 5-tuple:

- v_state = fractional v_star velocity of that agent, where v_star is the target velocity (defined as 15 m/s in configuration of this environment)
- vdiff_state= difference of velocities of current and its leading agent
- vhdiff_state= difference of velocities of current and the max speed allowed (defined as 30 m/s in configuration of this environment)
- h_state=fractional headway wrt h_star of that agent, where h_star is the target headway (defined as 20 m in configuration of this environment)
- u_state=fractional acceleration wrt to maximum allowed acceleration (defined as 2.5 m/s^2 in configuration of this environment)

Now the observation of any agent is its own state and the state of its neighbour(s).
In this environment, the action refers to 2-tuple (a,b) that represents the human driver behaviour where a is headway gain and b is relative velocity gain. The optimal velocity controller that this environment/agent uses has 4 discrete levels of (a,b): (0,0) (0.5,0) (0,0.5) (0.5,0.5). These 4 discrete actions have been mapped to 0, 1, 2, 3 placeholders.

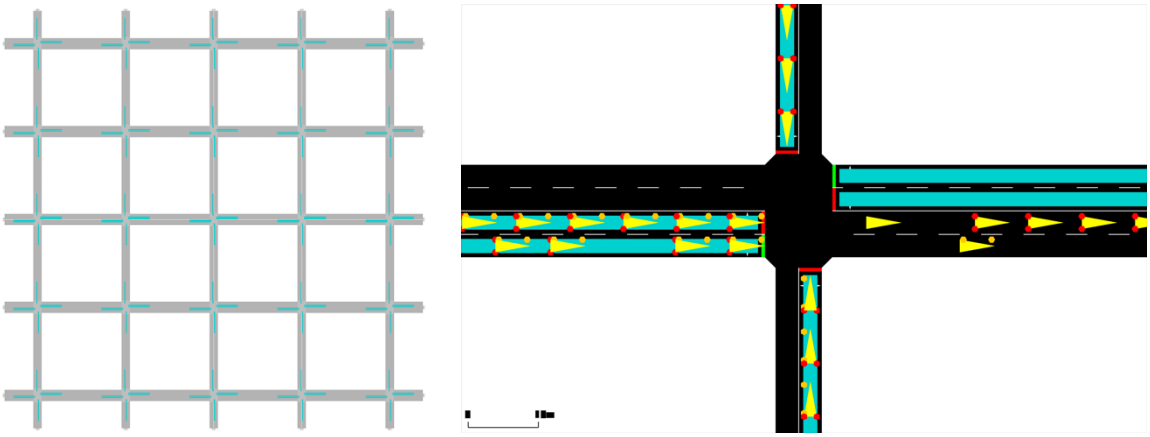


Fig: Environment for Traffic Signal Network along with the exploded view of one of the intersections in the complete grid [28]

The policy of each agent is the LSTM network that outputs a 4-dim array of probabilities. So for each agent we have a 4-dim array from which that agent selects the current action by randomly sampling from the policy according to the probabilities using the np.random.choice python function.

The neighbours in this environment are defined by putting some condition on the relative distance between the agents. And the neighbouring information is stored in a binary grid of size n_agent x n_agent.

During training, all the agents choose action by randomly sampling action according to probabilities outputted by LSTM network, while during evaluation/testing, each agent chooses the greedy action by taking argmax over the array of probabilities output by LSTM network.

Implementational Details

For each scenario, a set of 4 agents representing UAVs were chosen and were allowed to move in horizontal and vertical lanes in a larger grid as shown in the figure below. These agents are allowed to interact for 60 seconds in the environment and that counts as an end of an episode. A huge penalty is levied upon the collisions amongst agents and certain reward is given based on the velocity that the individual agent is able to continue moving with. A constant number of 500k episodes was chosen for each of the scenarios to be run based on the computational power that was available, while ensuring that insights and analysis could be done based on the results. The average episodic returns after every 600 episodes was transferred to a .CSV file and analysed visually for the trained number of episodes.

Chapter 5: Results & Discussion

5.1 Centralised Training and Decentralised Execution

Due to the high computational requirements, even in case of optimised algorithms with the deep learning architecture, high performance computing was necessary for complete execution of codes. As depicted in [19], the time taken for training a cluster of 3 UAVs for cooperative navigation problem was around 1.5 hours. This was further pushed due to lack of heavy computational resources by us while implementation.

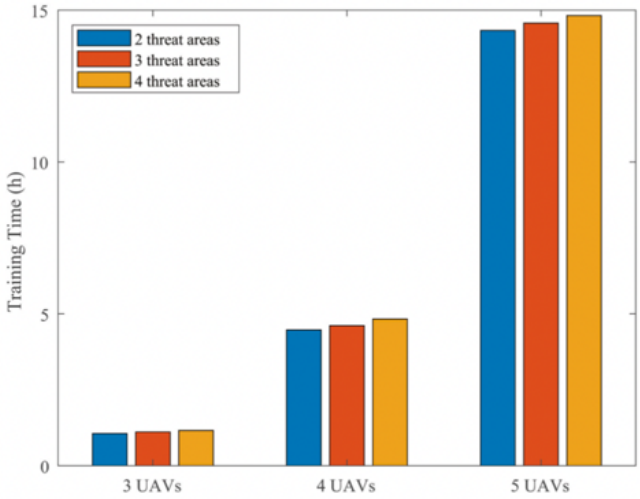


Fig: Comparison of time for training to convergence with Number of agents as 3,4,5 [19]

Also, as suggested by [2], MADDPG came across as one of the best algorithms for cooperative environments, with quick learnability and better overall episodic rewards. This shall also be visible from the results achieved from our simulation.

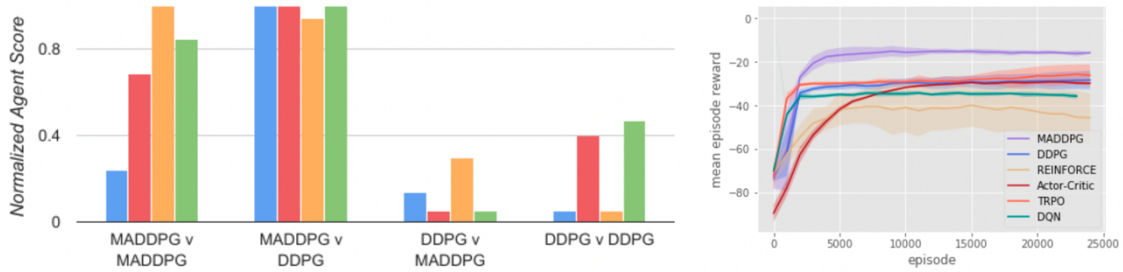


Fig: (a) Comparing results with cooperative agents using MADDPG & adversary using DDPG [2] (b) Comparison of multiple algorithms for the Cooperative Navigation [2]

Going forward with choosing MADDPG algorithm as the basic algorithm, we choose to simulate 3 scenarios related to our problem. These, along with their implementational details, results and simulations, have been discussed below.

1. **Cooperative Navigation for 3 UAVs with 3 Target Locations:**
   The cooperative scenario was implemented and trained for 20000 episodes, which took more than 2 hours to simulate. We used MADDPG algorithm for each of the cooperative agents and utilised 128 layers in the neural layer. The simulation was rendered to gain more accurate understanding of how agents are interacting with the environment and

the gradual process of increase in net reward as the number of episodes pass by. The agents go closer to their respective landmarks as the number of training episodes passes increase and the average episodic return increases. This can also be seen from the following simulation image.

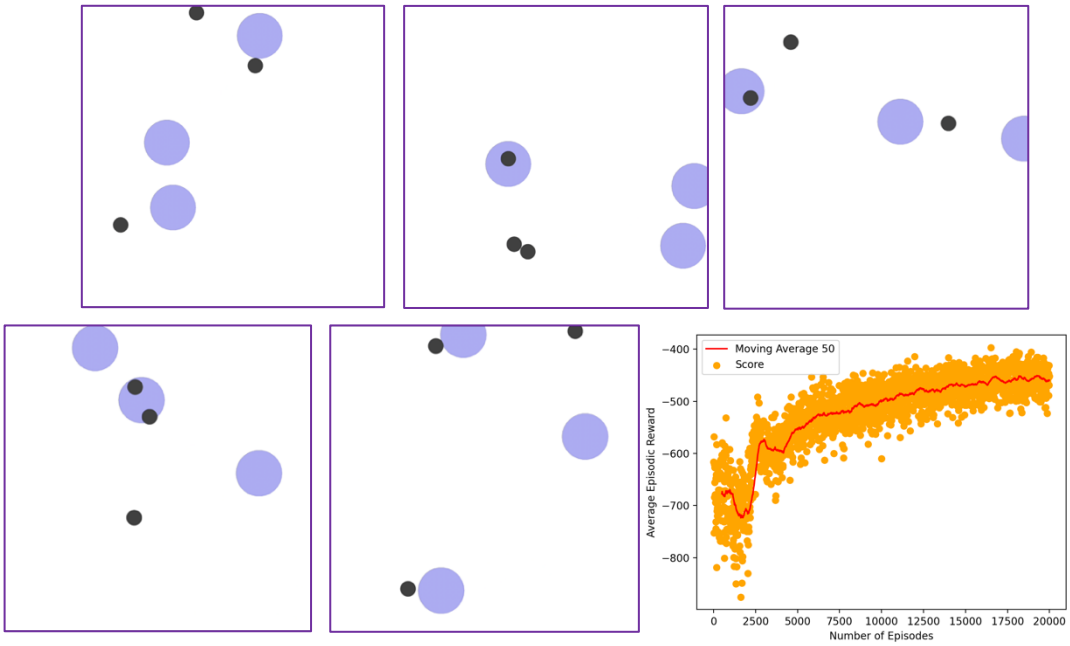


Fig: Simulation for Cooperative Navigation environment for distinct episodes

2. **Predator-Prey Environment with 3 collaborative UAVs, 1 adversary UAV and 2 non-breachable landmarks**
   This is a mixed environment with the cooperative agents learning with the MADDPG algorithm, while the adversary trained on DDPG algorithm. The environment was run for 10,000 episodes and the average episodic rewards was stored to get an overall view of the training being implemented under the hood. The number of units in the neural layer were chosen to be 128.

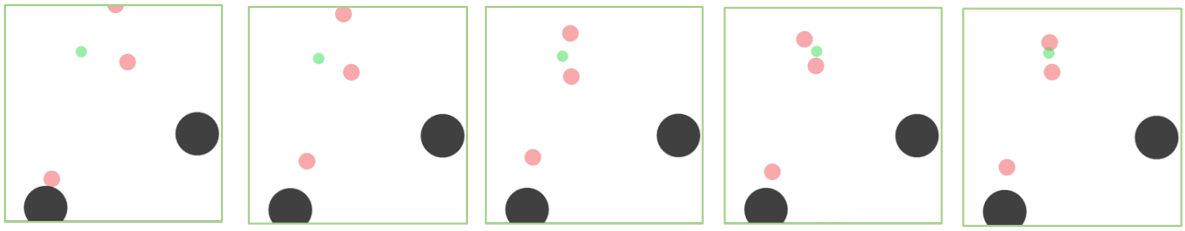


Fig: Training on Predator-Prey environment for 10000 episodes for a particular episode depicting how the cooperative agents learn to attack the prey without touching landmarks

As can be seen in the above simulation images, the three agents keep on closing onto the prey-agent, while avoiding landmarks which are not to be touched. Also, the average episodic return (score) keep increasing, with oscillatory behaviour as the number of episodes increase. This indicates how the agent is learning about the dynamic environment.

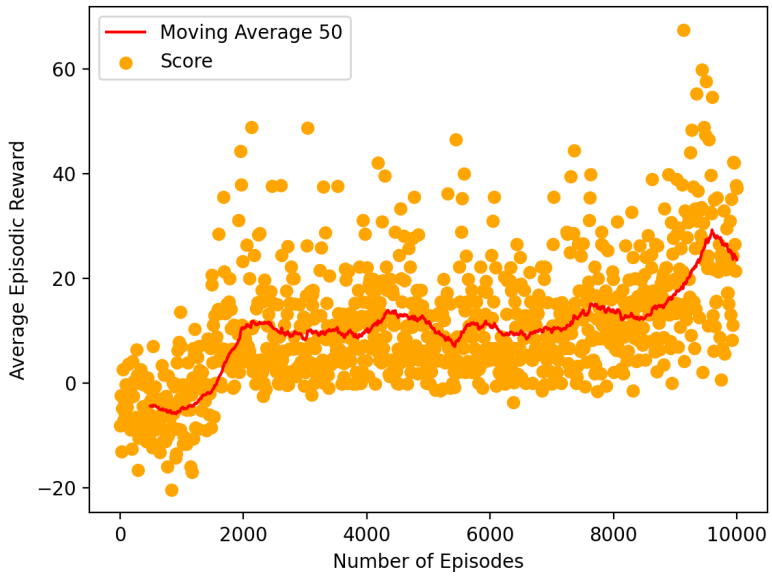


Fig: Training on Predator Prey environment for 10000 episodes with average reward representation over a training episodes

3. **Physical Deception with 2 cooperative agents and 1 adversary agent:**
   This is also a mixed environment with MADDPG algorithm for all the agents present in the system. The environment was run for 10,000 episodes and the average episodic rewards was stored to get an overall view of the training being implemented under the hood. Since the process was extremely heavy on computation, the experiment was curtailed when the rewards reached the near-optimal value. The agents were penalised for colliding with each other while rewarded based on the proximity to the landmarks. The number of units in the neural layer were chosen to be 128. It is observed that the learnability is quick due to the application of MADDPG and it gets to a positive reward within a span of 2000 episodes, and stays positive generally for all the next episodes.

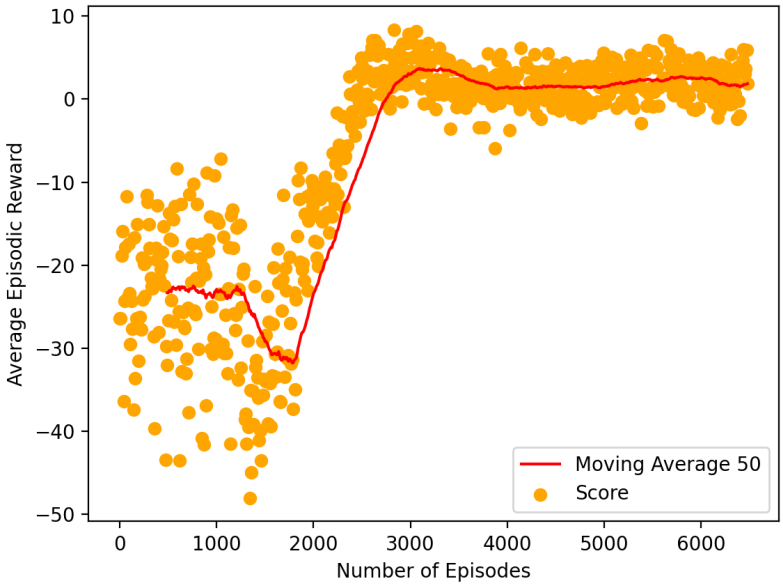


Fig: Training on Physical Deception environment for 6500 episodes using MADDPG

5.2 Decentralised Training and Decentralised Execution with Networked Agents

Having discussed the environments and their associated parameters that define the RL model like actions, policy, observations, rewards, etc., we moved onto simulating these scenarios applicable for our purpose. The plots and visual insights for both the scenarios are as listed below:

1. **CACC Slow-down Scenario:**
   The average episodic reward (AER) decreases initially due to the nature of the environment for all the cases. Specific insights are as follows:

   A. Without noise: AER is on the path to increase as it oscillates yet moves towards a better AER as episodes go by. This indicates that the agents are learning, though slowly due to independent evaluation and communication only amongst neighbouring agents.

   B. With light noise: The version with light noise performs better in terms of AER, though the rate of increase in the performance is lesser than the previous case. With more computational power, further insights could be drawn on the convergence of these results as a continuation to the project.

   C. With heavy noise: The model surprisingly performs even better almost throughout all the episodes. Possibly, the noise makes the agents to realise that their neighbours are closer than they actually are and hence the change in observations leads to better learnability in terms of better AER as episodes go by.

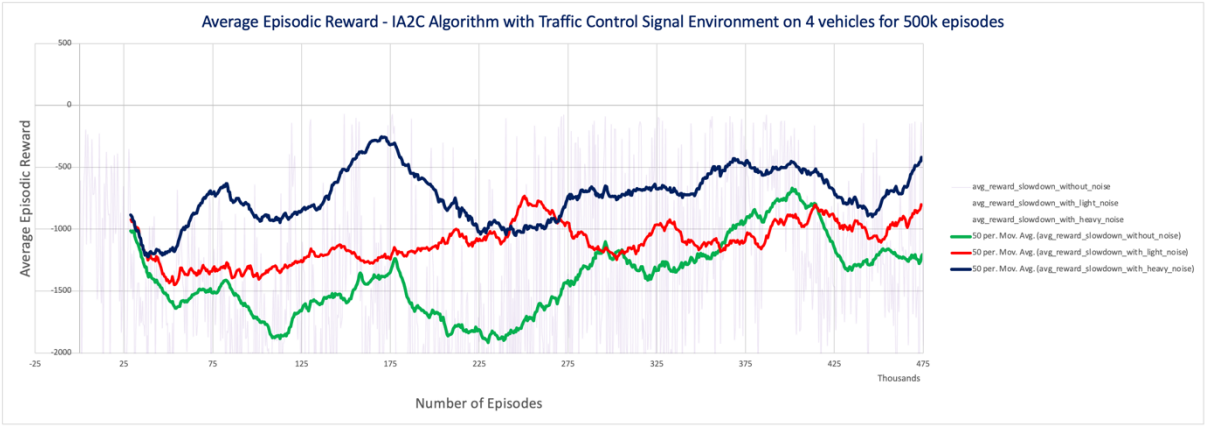


Fig: Training on CACC Slowdown scenario for 500k episodes using IA2C Algorithm with varying amounts of noise (light and heavy)

This further motivates that adding noise in the observations turns out to be a decent strategy in case of quicker and a better learnability scenario! Since the agents are fast initially in all the episodes, this noise perhaps makes them more vulnerable to collide and hence to avoid that, they take different actions that makes the AER better with episodes. Adding noise fulfils the objectives that were presented by the need of differential policy in the previous sections of the report and the representations support the presence of noise for the chosen environments under the Fully Decentralized MARL problem with networked agents.

2. **CACC Catch-up Scenario:**
   Since the agents are moving slowly in the initial part of their movement in the environment, the AER stands out to be much more positive relative to the other environment. Also, since the probability of collision is much lesser in this case, the fluctuations are lesser vulnerable to noise. Specific insights are as follows:

   A. Without Noise: The AER change is nearly steady in the first 500k episodes. It keeps oscillating up and down and the moving average shows a relatively steadier journey with the number of episodes.

   B. With light noise: The oscillations start to widen, but as more and more episodes pass, the vehicles start to learn better with slight noise than in the case when observations are sent through without noise. This is probably because of agents getting more vigilant about other vehicles for collision.

   C. With Heavy noise: In the initial thousands of episodes, the movement in the AER is more or less constant but near the end, it start to show better learnability!

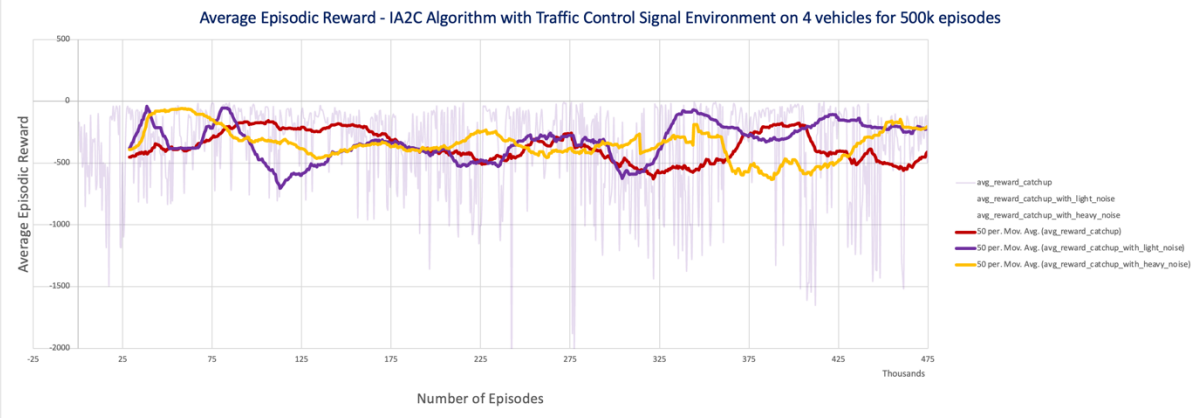


Fig: Training on CACC Catchup scenario for 500k episodes using IA2C Algorithm with varying amounts of noise (light and heavy)

A combined visualisation for all the plots is shown below to indicate the higher average episodic reward for the catchup scenarios where initial collisions are significantly reduced.

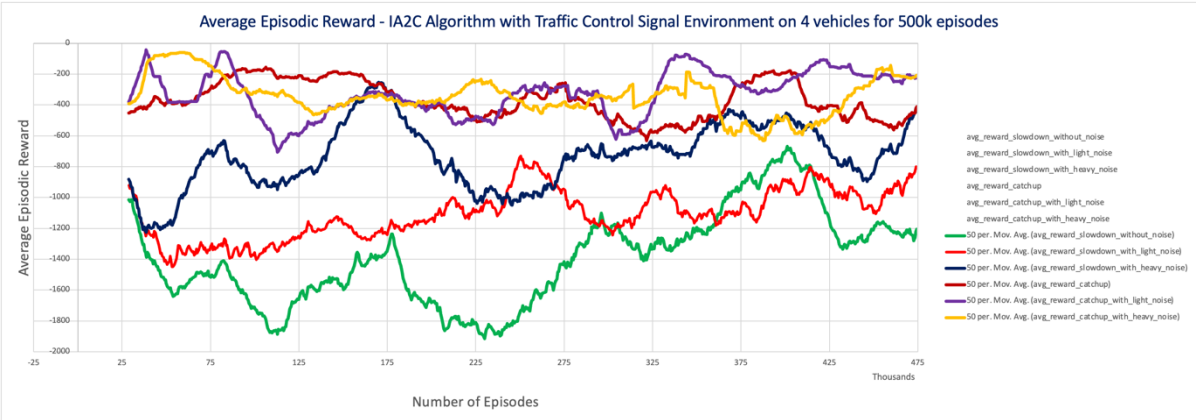


Fig: Combined Visualisation for Training on CACC Slowdown and Catchup scenario for 500k episodes using IA2C Algorithm with varying amounts of noise (light and heavy)

Both the experiments indicate that using noise to facilitate the idea of differential privacy turns out to give better Average Episodic Rewards from the agents and thus, adding a noise can boost the learnability of the agents especially in the case of independent evaluation using IA2C Consensus Algorithm.

Communication provides enhanced information so that vehicles can follow their predecessors with higher accuracy, faster response, and shorter gaps; the result would be enhanced traffic flow stability and possibly improved safety. The addition of noise seem to give better learnability atleast in the first 0.5mn episodes for independent scenarios. This motivates our consensus-based MARL algorithms that leverage the communication network to diffuse the local information, which fosters collaboration among the agents, along with strengthening further research that can be done on the effect of noise on such environments and scenarios.

Conclusion & Future Work

The simulations of the experiments were done, and the training curve was obtained as expected, with the scores increasing. Seeing the significantly large training time even when Deep Neural Networks were being used as function approximators, it can be firmly deduced that such real-world problems with multiple agents are not at all well-suited to be carried out with conventional RL algorithms with lookup tables.

Further, using a MADRL algorithm such as MADDPG, we need to develop a multi-UAV RL system having an actor network which will determine what any UAV would do, and a critic network for policy improvement. As an extension to it, modifying the algorithm to incorporate constraints relevant to UAVs and exploring if it could be made decentralized remains to be done. These directions would be explored with work to be done towards a novel algorithmic technique creation for Cooperative Multi-Agent Reinforcement Learning for UAVs.

The realistic noise addition was performed to the observations' of the agents at three different levels: (1) no noise, exact communication, (2) little (feeble) noise, and (3) heavy noise. We observed the effects of noise addition and obtained that the addition of noise not only made the environment mimic the practical scenario a bit more closely, infact this little addition helped the learnability of the algorithm in some scenarios wrt the same in an environment without any noise.

Further work would involve implementing "delayed-IA2C" algorithm where we would see how the IA2C algorithm performs if we also have a delay in the communication between agents.

References

[1] Sutton, Richard S., and Andrew G. Barto. Reinforcement learning: An introduction. MIT press, 2018.
[2] Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments
[3] Contrasting Centralized and Decentralized Critics in Multi-Agent Reinforcement Learning
[4] Achiam, Joshua, et al. "Constrained policy optimization." International Conference on Machine Learning.
[5] Z. Wang, Y. Zhang, C. Yin and Z. Huang, "Multi-agent Deep Reinforcement Learning based on Maximum Entropy," 2021 IEEE 4th Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC), 2021, pp. 1402-1406, doi: 10.110
[6] Lyapunov-Based Reinforcement Learning for Decentralized Multi-Agent Control; Qingrui Zhang, Hao Dong, and Wei Pan; Sept 2020
[7] Reducing Overestimation Bias in Multi-Agent Domains Using Double Centralized Critics; Johannes Ackermann, Volker Gabler, Takayuki Osa, Masashi Sugiyama; Dec 2019
[8] Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, Sergey Levine; Jan 2018
[9] Distributed Distributional Deterministic Policy Gradients; Gabriel Barth-Maron, Matthew W. Hoffman, David Budden, Will Dabney, Dan Horgan, Dhruva TB, Alistair Muldal, Nicolas Heess, Timothy Lillicrap; April
[10] QMIX: Monotonic Value Function Factorisation for Deep Multi-Agent Reinforcement Learning; Tabish Rashid, Mikayel Samvelyan, Christian S.Witt, Gregory Farquhar, Jakob Foerster, Shimon Whiteson; ICML 2018
[11] A library of multi-agent reinforcement learning components and systems
[12] Multi-agent reinforcement learning: An overview;
L. Bus ̧oniu, R. Babus ̆ka, and B. De Schutter; 2010
[13] Multi-Agent Particle Environments
[14] Multi-Agent Reinforcement Learning: A Review of Challenges and Applications; Lorenzo Canese †, Gian Carlo Cardarilli, Luca Di Nunzio †, Rocco Fazzolari, Daniele Giardino †, Marco Re † and Sergio Spanò; May 21
[15] Autonomous UAV Navigation Using Reinforcement Learning; Huy X. Pham, Hung M. La, David Feil-Seifer, Luan V. Nguyen; Jan 2018
[16] Application of reinforcement learning in UAV cluster task scheduling; Jun Yang a, Xinghui You a,∗, Gaoxiang Wu a,∗, Mohammad Mehedi Hassan b, Ahmad Almogren b, Joze Guna c; Jan 2019
[17] Reinforcement Learning for UAV Attitude Control; WILLIAM KOCH, RENATO MANCUSO, RICHARD WEST, and AZER BESTAVROS; 2019
[18] Cooperative and Distributed Reinforcement Learning of Drones for Field Coverage; Huy Xuan Pham, Hung Manh La, David Feil-Seifer, and Ara Nefian; Sept 2018
[19] Joint Optimization of Multi-UAV Target Assignment and Path Planning Based on Multi-Agent Reinforcement Learning; HAN QIE, DIANXI SHI, TIANLONG SHEN, XINHAI XU, YUAN LI AND LIUJING WANG; October 2019
[20] Papers with Code
[21] TensorFlow 2 Implementation of Multi-Agent Reinforcement Learning Approaches
[22] Watkins, C.J.C.H., Dayan, P. Q-learning. Mach Learn 8, 279–292 (1992).
https://doi.org/10.1007/BF00992698
[23] Mnih, Volodymyr, et al. "Playing atari with deep reinforcement learning." arXiv preprint arXiv:1312.5602
[24] A2C / A3C (Asynchronous Advantage Actor-Critic): Mnih et al, 2016
[25] DDPG (Deep Deterministic Policy Gradient): Lillicrap et al, 2015
[26] Rashid, Tabish, et al. "Qmix: Monotonic value function factorisation for deep multi-agent reinforcement learning." International Conference on Machine Learning. PMLR, *2018.*
[27] Consensus Algorithm Blockchain: https://toshitimes.com/why-consensus-algorithms-matter-for-developers/
[28] Networked Multi Agent Reinforcement Learning (NMARL): https://github.com/cts198859/deeprl_network
[29] PolicyInferring: Lowe, Ryan, et al. "Multi-agent actor-critic for mixed cooperative-competitive environments." Advances in Neural Information Processing Systems, 2017.
[30] FingerPrint: Foerster, Jakob, et al. "Stabilising experience replay for deep multi-agent reinforcement learning." arXiv preprint arXiv:1702.08887, 2017.
[31] ConsensusUpdate: Zhang, Kaiqing, et al. "Fully decentralized multi-agent reinforcement learning with networked agents." arXiv preprint arXiv:1802.08757, 2018.
[32] Cooperative Adaptive Cruise Control: Definitions and Operating Concepts:
https://journals.sagepub.com/doi/10.3141/2489-17
[33] https://medium.com/secure-and-private-ai-writing-challenge/differential-privacy-e5c7b933ef9e
[34] https://en.wikipedia.org/wiki/Laplace_distribution
[35] Multi-Agent Deep Reinforcement Learning for Large-scale Traffic Signal Control
[36] MULTI-AGENT REINFORCEMENT LEARNING FOR NETWORKED SYSTEM CONTROL