Multi-Agent Reinforcement Learning for Independent & Cooperative Behaviour on Highways

A Project Report Submitted
in Partial Fulfillment of the Requirements
for the Degree of

Bachelor of Technology

by

Alisetti Sai Vamsi
(111801002)

INDIAN INSTITUTE
OF TECHNOLOGY
PALAKKAD

COMPUTER SCIENCE AND ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY PALAKKAD

CERTIFICATE

*This is to certify that the work contained in the project entitled "**Multi-Agent Reinforcement Learning for Independent & Cooperative Behaviour on Highways**" is a bonafide work of **Alisetti Sai Vamsi (Roll No. 111801002)**, carried out in the Department of Computer Science and Engineering, Indian Institute of Technology Palakkad under my guidance and that it has not been submitted elsewhere for a degree.*

Dr. Albert Sunny

Assistant/Associate Professor

Department of Computer Science & Engineering

Indian Institute of Technology Palakkad

Acknowledgements

I would like to express my special thanks of gratitude to the ***Ms. Helen Thomas*** *(Ph.D student at IISc working under Prof. Tarun Rambha) who built the simulator in pygame along with me and would like to thank* ***Dr. Tarun Rambha*** *for guiding me through the project. I would like to thank* ***Dr. Albert Sir*** *for agreeing to be the internal guide for this project. I am also thankful to IISc for providing the server for performing my research experiments.*

I would also like to thank ***IIT Palakkad Technology IHub Foundation*** *for offering me the fellowship for this proejct.*

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

In the recent years, autonomous driving has gained appreciable traction in the AI community. Autonomous driving is expected to offer safe & efficient mobility on congested traffic lanes. To develop a rule-based categorical driving policy is of utmost difficulty since the number of scenarios or the traffic patterns that a vehicle can face is exponentially large and hence hand-crafting a rule-based controller for each of these myriad scenarios is not a viable option. A machine learning approach is needed to mitigate these shortcomings of a rule based controller to extend the behavior of the vehicle to unknown situations. Reinforcement Learning (RL) [8] can be used to find optimal control policies that can adapt to unknown traffic patterns.

Over the past decade, RL has produced commendable successes in control tasks such as AlphaGo [9] (defeated the world champion of the game Go), Atari [10], AlphaFold [11] (solved a 50 year old problem of protein folding). This attracted the transportation community in using RL for complex tasks like traffic signal control [12], ramp-metering [13], forward driving, lane changing behavior, etc. Most of these studies, focus primarily on finding the optimal driving policy through a single vehicle that acts as a reinforcement learning agent. But this is seldom the case in real life scenarios where multiple vehicles are being driven on any given road. Multi-Agent Reinforcement Learning (MARL) [5] tackles this problem of finding the optimal strategy in the presence of multiple vehicles acting as reinforcement learning agents.

My research branches out in two different directions. First is to design a novel algorithm for coordinated behavior on highways using MARL. Second is to use coordinated MARL to dissipate traffic congestion waves in ring road setups.

1.2 Organization of The Report

This chapter provides the motivation for the rest of the thesis. Chapter 2 provides a detailed review and background on the current work and explains preliminary concepts needed to comprehend the results. Chapter 3 discusses about agents in highways and presents results on collision avoidance. Chapter 4 discusses about agents in ring roads and presents results on traffic shock wave mitigation. Finally, Chapter 5 concludes with some directions into the future works.

Chapter 2

Review of Prior Works

2.1 Markov Decision Process & Reinforcement Learning

Reinforcement learning is a class of machine learning algorithms that is used for learning sequential decision making behavior. In RL literature we have something called an agent i.e the learner who is trying to accomplish a task and we have an environment with which the agent interacts to achieve a specific goal. Every problem has an environment in which the agent acts and based on this action the environment returns the next state the agent should be in and the reward obtained for the action that the agent chose. The goal of the agent is to choose the actions that maximizes the cumulative reward that it obtains on the long run. Every RL problem includes this behaviour loop as shown in the Fig **2.4**.

Mathematically, RL problems can be formulated as Markov Decision Processes (MDP). MDP is a 5-tuple $< S, A, R, P, \gamma >$ defined as follows:

1. **State Space** (S): A finite set of states. $S \in \mathbb{R}^d$ where d is the size of the feature set that represents the environment.
2. **Actions** (A): A finite set of actions available to agent.

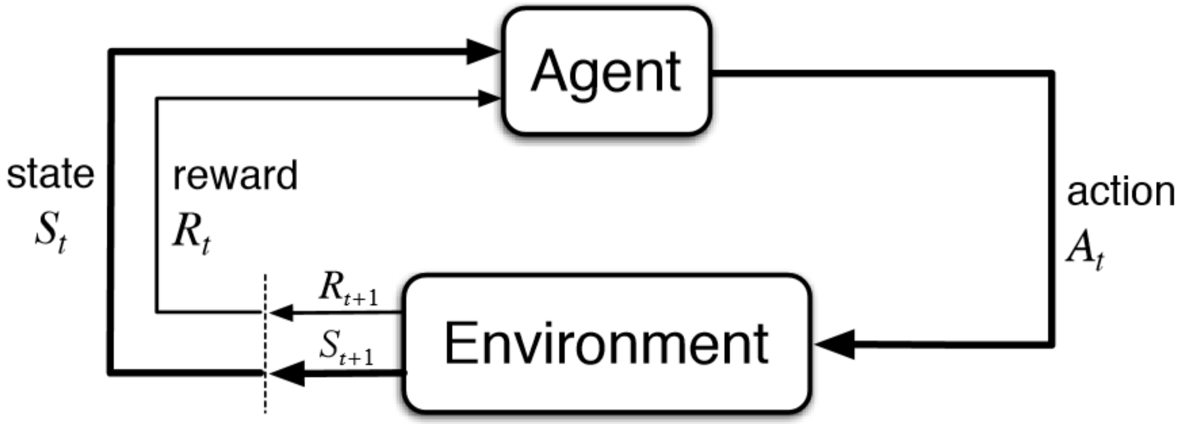

Fig. 2.1: RL Behavior Loop [1]

3. **Transition Probability** (P): An ($|S| \times |S|$) matrix indicating the transition probability from one state to another after performing a particular action. $P(s, a, s') : S \times A \to [0, 1]$.

4. **Reward Function** (R): A function that provides scalar reward to an agent upon performing a specific action. $R : S \times A \to \mathbb{R}$.

5. **Discount Factor** (γ): Parameter indicating how far the agent should look ahead before making a decision. $\gamma \in [0, 1]$. A value of 1 indicates that the agent is looking ahead and basing its actions on future rewards whereas a value of 0 indicates that it is just looking at immediate rewards. Rewards are thus discounted by this factor of γ not only to choose between future and immediate rewards but also to make the problem of infinite time horizon, i.e the policy of the agent does not change with time and because of this the amount of rewards that the agent receives must be moderated i.e it should not blow up.

Three more important terms that need to be introduced are:

1. **Policy** (π): This is equivalent to the agent's brain. The policy of an agent is its behavior. The policy is a map from the states of the environment to the action set

of the agent $\pi : S \rightarrow A$. Therefore if the agent is in some state say s then the agent will take the action a where $a = \pi(s)$.

2. **Value Function** ($V^{\pi}(s)$): Value Function gives a relative measure of how good a state is on the long-term. Value of a state is defined to be the expected return starting from that state. Return (G_t) is defined to be the total discounted reward obtained from time step t.

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... + \gamma^{\infty} R_{\infty} \tag{2.1}$$

$$V^{\pi}(s) = \mathbb{E}_{\pi}[G_t | S_t = s] \tag{2.2}$$

3. **Action Value Function** ($Q^{\pi}(s, a)$): Action Value Function gives the relative measure of how good an action is from a particular state on the long-term. This is defined as the expected return starting from a state and taking a particular action.

$$Q^{\pi}(s, a) = \mathbb{E}_{\pi}[G_t | S_t = s, A_t = a] \tag{2.3}$$

The goal of reinforcement learning is to maximize the expected discounted reward at any given state i.e it needs to obtain the maximum possible value at every state. The optimal policy can be obtained by value iteration and policy iteration algorithms [14]. This requires the model of the network i.e it requires knowledge of transition probabilities. But RL deals with model free problems i.e the RL algorithms try to approximate the transition probabilities of the environment by collecting samples from the environment. Model-free reinforcement algorithms are further classified into value based and policy based methods.

2.2 Q-Learning

Q-Learning is a value-based model-free RL algorithm that finds the Q values (action values) of the states by iteratively collecting samples from the environment and updating the Q value table. The Q value table is a matrix of size $|S| \times |A|$ which stores the corresponding Q values of Q(s,a). The Q Learning update equation is as follows:

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma max_{a'} Q(s',a') - Q(s,a)] \tag{2.4}$$

where, s', a' are the next state, action respectively and α is the stepsize.

Below is the algorithm for Q-Learning using Q value tables.

Algorithm 1: Q-Learning Algorithm

```
1  Initialise $\alpha, \gamma$
2  Q(s,a) $\rightarrow$ 0
3  Sample action a $\sim \pi_\theta$
4  for $Episodes = 1, 2, 3...$ do
5      Randomly select a state $s$
6      while $s \neq s_{goal}$ do
7          $a \sim \pi(s)$
8          $s', r \leftarrow transition(s, a)$
9          $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma max_{a'} Q(s',a') - Q(s,a)]$
10         $s = s'$
```

2.2.1 Deep Q Network (DQN)

A major drawback with using Q value tables is that it needs to store all the Q values in memory. This might seem trifling for problem with small state space but complex problems can see state space in the order of 10^4 to 10^8. Therefore in order to overcome this shortcoming we have moved to the use of neural networks to approximate these Q values instead of storing them.

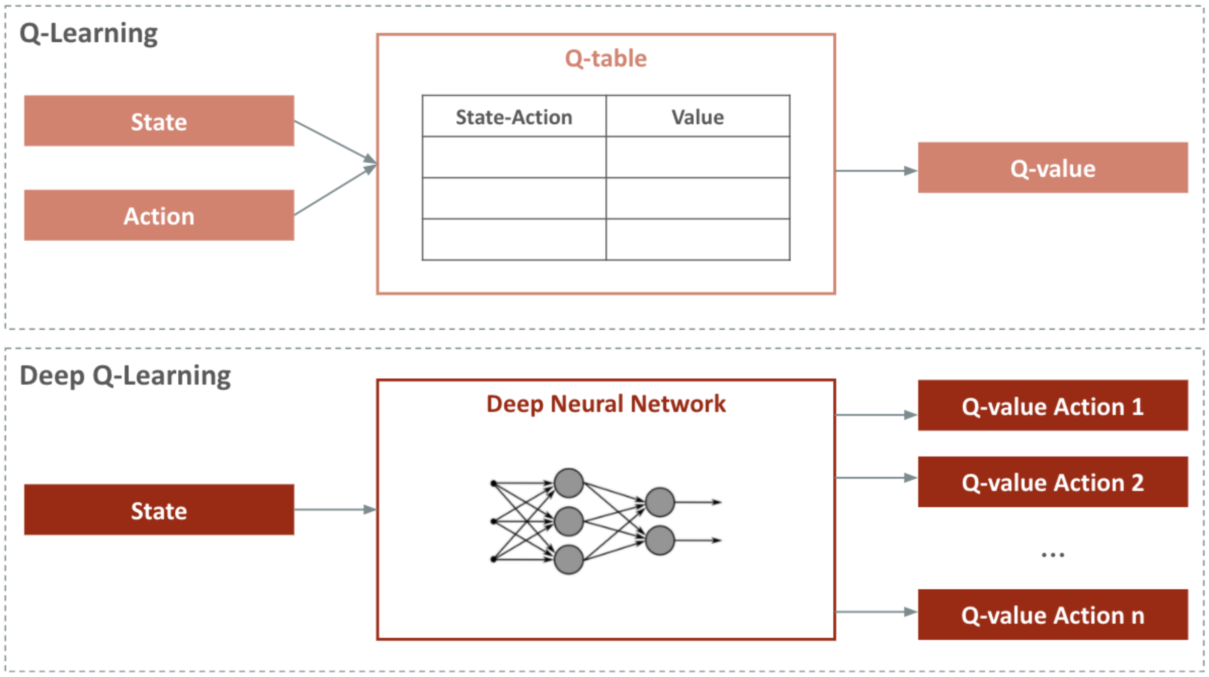

Fig. 2.2: Deep Q Network [2]

2.2.2 Double Deep Q Network

In the vanilla DQN, the Q targets keep changing every iteration, which means that the Q values that we are trying to estimate have moving Q targets. It is better if the Q targets stay constant for a while so that the Q values can catch up to those. This problem is called the maximization bias and can be solved using a second Q network solely for storing the target weights and updating the weights to the online Q network according to a fixed frequency. The following is the update equation for Double DQN:

$$Q_{online}\left(s_t, a_t; \theta_i\right) \leftarrow R_{t+1} + \gamma max_{a'} Q_{target}\left(s_{t+1}, a'; \theta_{i-1}\right) \tag{2.5}$$

Q_{target} provides the estimated Q value for the next iteration using the stored weights and the action to be performed is taken from Q_{online} by taking an argmax on action space. θ_i denotes the weights of the Q network in the i^{th} iteration.

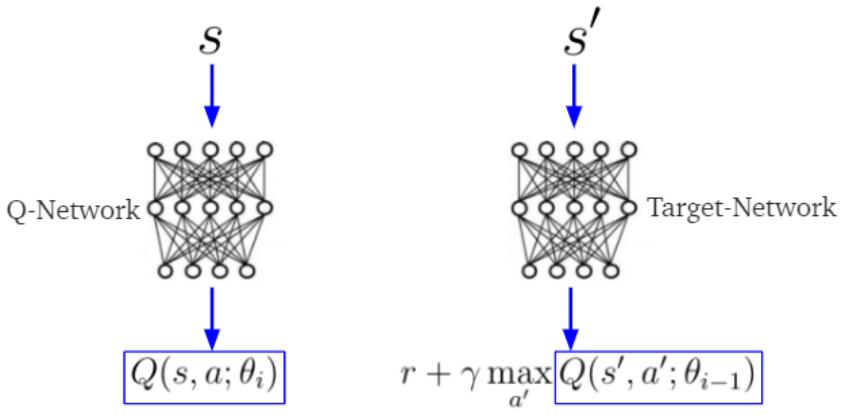

Fig. 2.3: Double Deep Q Network [3]

2.2.3 Dueling Deep Q Network

Dueling DQN splits the Q value into Value Function V(s) and the Advantage Action Function A(s,a). Value Function gives the measure of how rewarding a state is where as the advantage action function gives us the measure of the advantage we get by choosing a particular action. The following is the Q value update equation:

$$Q(s,a) = V(s) + \left(A(s,a) - \frac{1}{|\mathcal{A}|}\sum_{a'} A(s,a')\right) \tag{2.6}$$

where $\mathcal{A}$ denotes the number of actions in the action set.

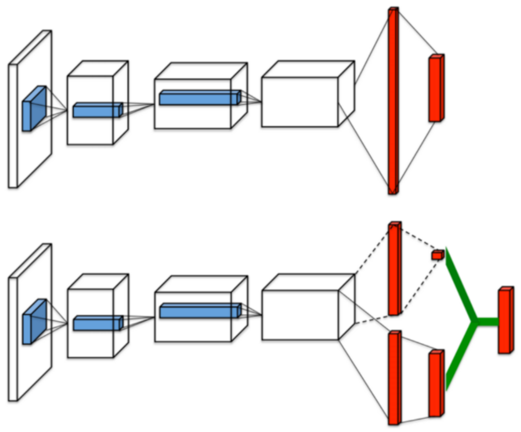

Fig. 2.4: Vanilla DQN (top), Dueling Deep Q Network (bottom) [4]

2.3 Policy Gradient Methods

Actor Critic is an RL algorithm that falls under the wing of Policy Gradient Methods. Policy Gradient Methods outperform value-based methods in Model-Free contexts because they can readily learn stochastic policies and are very efficient in MDPs with continuous state spaces. In policy gradient methods the policy $\pi(a|s)$ is parametrized i.e $(\pi_\theta(a|s))$ where a is action and s is the state.

$$\pi_\theta(a|s) = P[a|s, \theta] \tag{2.7}$$

The goal of policy gradient methods is to obtain the optimal θ given $\pi_\theta(a|s)$.

To evaluate the performance of the policy we have an objective function $J(\theta)$ which essentially give us a measure of how good the policy is compared to other policies based on the parameter θ.

$$J(\theta) = V_{\pi_\theta}(s) \tag{2.8}$$

In the above equation, $V_{\pi_\theta}(s)$ denotes the value obtained upon following the policy π_θ. Now the goal is to find the θ that maximizes $J(\theta)$ which can be done using gradient ascent.

$$\theta = \theta + \alpha \frac{\partial J(\theta)}{\partial \theta} \tag{2.9}$$

where α is the step size. Now according to the policy gradient theorem we have,

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}[\nabla_\theta log(\pi_\theta(a|s))\delta] \tag{2.10}$$

where,

$$\delta = R(s, a) + \gamma Q_w^\pi(s', a') - Q_w^\pi(s, a) \tag{2.11}$$

Using the above equations our policy gradient step becomes,

$$\theta = \theta + \alpha \nabla_\theta log(\pi_\theta(a|s))\delta \tag{2.12}$$

We lose the expectation since we follow an iterative algorithm and will be taking averages instead. The Q values in this equation also needs to be evaluated on the current policy which needs a separate neural network. These Q values gives us a measure of the quality of the policy. In essence the critic updates Q-value function weights, and the actor updates policy parameters θ guided by the Q values from the critic.

Algorithm 2: Q Value Actor Critic

```
1  Initialise $s, \theta, w$
2  Sample action a $\sim \pi_\theta$
3  for $Episodes = 1, 2, 3...$ do
4      while $s \neq s_{goal}$ do
5          $s', r \leftarrow (s, a)$
6          $a' \sim \pi_\theta(.|s')$
7          $\delta \leftarrow r + \gamma Q_w^{\pi_\theta}(s', a')$ - $Q_w^{\pi_\theta}(s, a)$
8          $\theta \leftarrow \theta + \alpha \nabla_\theta log(\pi_\theta(a|s))\delta$
9          $w \leftarrow w + \beta\delta\phi(s, a)$
10         $a \leftarrow a'$
11         $s \leftarrow s'$
```

2.4 Intelligent Driver Model (IDM)

IDM is a car following model that provides the dynamic position and velocity of a vehicle. Consider a single vehicle i and the vehicle in front of it f, and let x_i, x_f denote the position at a time t of the vehicles i, f respectively and v_i, v_f denote the velocity at time t of the vehicles i, f respectively. Let l_i, l_v be the length of the vehicles i, f respectively. Net distance $s_i = x_f - x_i - l_f$ and the approaching rate $\Delta v_i := v_i - v_f$. The dynamics are hence defined by the following ordinary differential equations:

$$\dot{x}_i = \frac{\mathrm{d}x_i}{\mathrm{d}t} = v_i \tag{2.13}$$

$$\dot{v}_i = \frac{\mathrm{d}v_i}{\mathrm{d}t} = a\left(1 - \left(\frac{v_i}{v_0}\right)^{\delta} - \left(\frac{s^*\left(v_i, \Delta v_i\right)}{s_i}\right)^2\right) \tag{2.14}$$

$$s^*\left(v_i, \Delta v_i\right) = s_0 + v_i T + \frac{v_i \Delta v_i}{2\sqrt{ab}} \tag{2.15}$$

The parameters that can be changed here are v_0, s_0, T, a, b which are described as follows:

1. v_0 (*desired velocity*): The velocity to drive in traffic free case.
2. s_0 (*minimum spacing*): This is the desired net distance value i.e the minimum distance to the front vehicle.
3. T (*desired headway*): Minimum time to the front vehicle
4. a (*acceleration*): Maximum acceleration of the vehicle
5. b (*braking deceleration*): Maximum braking deceleration
6. δ (*acceleration exponent*): This is usually set to 4.

2.5 FollowerStopper Controller

As the name suggests, this model based controller commands a desired velocity U whenever it is within a safe distance from lead vehicle, and it commands a safer lower velocity $v_{cmd} < U$ whenever safety is required. Using the gap $\triangle x$ and the relative velocity $\triangle v = v_{lead} - v_{agent}$, the controller model is defined by the following equations:

$$v_{\text{cmd}} = \begin{cases} 0 & \text{if } \Delta x \leqslant \Delta x_1 \\ v\frac{\Delta x - \Delta x_1}{\Delta x_2 - \Delta x_1} & \text{if } \Delta x_1 < \Delta x \leqslant \Delta x_2 \\ v + (U - v)\frac{\Delta x - \Delta x_2}{\Delta x_3 - \Delta x_2} & \text{if } \Delta x_2 < \Delta x \leqslant \Delta x_3 \\ U & \text{if } \Delta x_3 < \Delta x. \end{cases} \tag{2.16}$$

where $v = min(max(v_{lead}, 0), U)$ and $\triangle x_{1,2,3}$ is defined as follows:

$$\Delta x_k = \Delta x_k^0 + \frac{1}{2d_k}\left(\Delta v_-\right)^2, \quad \text{for } k = 1, 2, 3 \tag{2.17}$$

2.6 Proportional Integral Controller

The PI controller decides upon a command velocity v_{cmd} such that for small gaps the agent should follow the lead vehicle and for large gaps, the agent should catch up to the lead vehicle. The desired velocity is calculated as a temporal average ($U = \frac{1}{m}\sum_{j=1}^{m} v_j^{\text{agent}}$) of the agent's velocity. Generally velocity history length (m) is taken to be $38s$. This desired velocity is translated into a target velocity by the following equation:

$$v_{\text{target}} = U + v_{\text{catch}} \times \min\left(\max\left(\frac{\Delta x - s_l}{s_u - s_l}, 0\right), 1\right) \tag{2.18}$$

where, s_l is the lower gap limit and s_u is the upper gap limit. The final v_{cmd} is calculated using v_{target} and v_{lead} by the following equation:

$$v_{j+1}^{\text{cmd}} = \beta_j\left(\alpha_j v_j^{\text{target}} + (1 - \alpha_j)\, v_j^{\text{lead}}\right) + (1 - \beta_j)\, v_j^{\text{cmd}} \tag{2.19}$$

where α_j, β_j are weights that depend on the gap in front as follows:

$$\beta_j = 1 - \frac{1}{2}\alpha_j \tag{2.20}$$

$$\alpha_j = \min\left(\max\left(\frac{\Delta x - \Delta x^{\mathrm{s}}}{\gamma}, 0\right), 1\right) \tag{2.21}$$

where $\triangle x^s$ is safety distance and the parameter γ controls the rate of transition of the weight α from 0 to 1.

2.7 Multi Agent Reinforcement Learning

An intuitive approach to find optimal strategies for a multi agent system is to train the agents independently, i.e to learn the decentralized policy. Approaches to independent agent learning [15] are prone to instability that arises from the non-stationarity of the environment created by simultaneous learning agents.

The flipside of this switch is to train the agents in a centralized approach with joint action and state spaces. This can handle the non-stationarity problem and also aids in agent co-ordination. Centralized approach bottlenecks in scaling to a higher dimensional state and action space. Recent works have developed a hybrid approach that exploits the co-ordination of centralized training with decentralized execution.

2.7.1 MDP

The MDP for a cooperative multi-agent system can be abstracted by the tuple $\langle S, A, T, r, X, O, n, \gamma\rangle$ where $s \in S$ denotes the state of the environment. Each agent $i \in 1, ..., n$ chooses an action $a^i \in A$ forming a joint action $a \in A$. These actions taken by each agent causes a transition in the environment according to the transition probability matrix $T\left(s' \mid s, \mathbf{u}\right) : S \times \mathbf{U} \times S \rightarrow [0, 1]$. All the agents follow the same joint reward function $r(s, \mathbf{a}) : S \times \mathbf{A} \rightarrow \mathbb{R}$, and the discount factor is as usual $\gamma \in [0, 1)$. Since our agent observations are partial and each agent draws individual observations $x \in X$ following the observation function $O(s, a) : S \times A \rightarrow X$. Each agent maintains a history of actions and observation $\tau^i \in Y \equiv (X \times A)^*$ upon which the the stochastic policy is conditioned

$\pi^i\left(a^i \mid \tau^i\right) : Y \times A \rightarrow [0,1]$.

2.7.2 Distributed Learning

Distributed Learning in reinforcement learning agents implies the use of multiple agents to train a single policy. Multiple agents in the system follow the same policy and reinforce this singular policy through experiences from multiple agents.

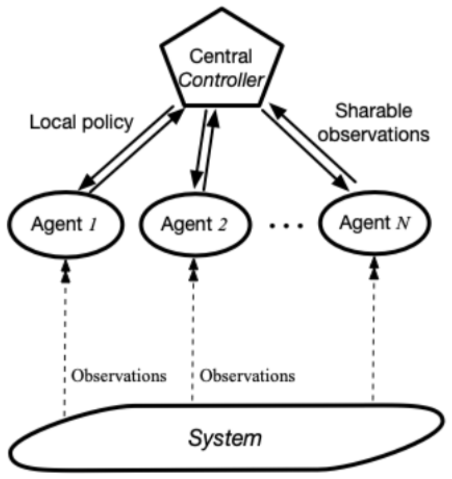

Fig. 2.5: Distributed Learning Setup [5]

2.7.3 Fully Independent Learning

Most commonly applied method in multi-agent reinforcement learning is the independent reinforcement learning where the multi-agent problem is decomposed into a collection of parallel single-agent problems that share the same environment. Non-stationarity is introduced into the environment because of independent learners since from each learner's perspective the environment is dynamic because of other independent learners. This approached does not have convergence guarantees even in the limit of infinite exploration.

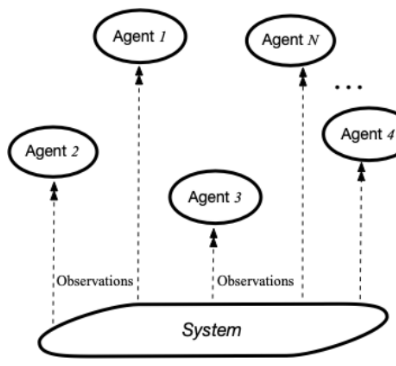

Fig. 2.6: Independent Learning Setup [5]

2.7.4 Centralized Training Decentralized Execution (CTDE)

In CTDE, the learning algorithm has access to local action observation histories because of centralized training but the policy is conditioned only on its own actions because of the decentralized execution. The following are the most promising models that are implemented with CTDE.

2.7.5 Value Decomposition Networks (VDN)

Value Decomposition Networks [6] aim to learn the joint-action value function $Q_{tot}(\boldsymbol{\tau}, \mathbf{a})$ where a is a joint action and $\tau \in \mathbf{Y} \equiv \mathcal{Y}^n$ is the joint action-observation history. VDN's claim is that it can decompose the joint action value function into individual value functions that condition on individual action-observation histories.

$$Q_{tot}(\boldsymbol{\tau}, \mathbf{a}) = \sum_{j=1}^{n} Q_j \left(\tau^j, a^j; \theta^j\right) \tag{2.22}$$

2.7.6 QMIX

QMIX [7], like VDN is a CTDE method but can represent a much richer class of action-value functions. QMIX represents the Q_{tot} by an architecture that consists of agent networks, mixing network and a set of hypernetworks [16]. The architecture contains one network

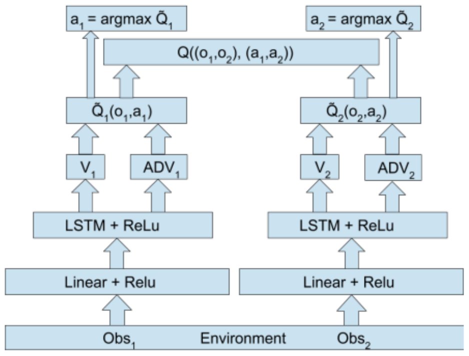

Fig. 2.7: Value Decomposition Architecture for multi-agent system (2 agents) [6]

(DRQN [17]) per agent to represent its value function which receives the current observation and the last action as input at each time step. Mixing network is a feed-forward neural network that takes the agent network outputs and mixes them producing the values of Q_{tot}. The weights of mixing network are produced separately by hypernetworks.

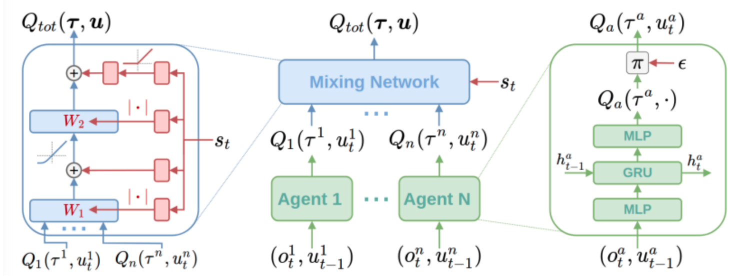
$Q_{tot}(\boldsymbol{\tau}, \boldsymbol{u})$
W_2
$|\cdot|$
W_1
s_t
$Q_1(\tau^1, u_t^1)$ $Q_n(\tau^n, u_t^n)$
Mixing Network
$Q_1(\tau^1, u_t^1)$ $Q_n(\tau^n, u_t^n)$
Agent 1
Agent N
(o_t^1, u_{t-1}^1) (o_t^n, u_{t-1}^n)
$Q_a(\tau^a, u_t^a)$
π
ϵ
$Q_a(\tau^a, \cdot)$
MLP
GRU
MLP
h_{t-1}^a
h_t^a
(o_t^a, u_{t-1}^a)

Fig. 2.8: QMIX Architecture [7]

2.8 Conclusion

This chapter provides details about the preliminaries needed to understand the rest of the thesis. It describes the algorithms that are used and the basic introduction of reinforcement learning and markov decision processes. In the next chapter we discuss the setup of agents in highway and the training results.

Chapter 3

Agent in Highway

3.1 MDP Formulation

3.1.1 About the simulator

The simulator is built using PyGame library [18] and a snippet of the simulator is shown in Fig 3.2. The simulator consists of two lanes and the agent is in red color. All the other vehicles in the simulation follow the IDM model and don't swich lanes and are generated according to a random seed. The simulation runs at 10 FPS.

3.1.2 Environment Setting

The following are the IDM parameters used to create the traffic simulation:

Light Traffic Parameters	
v_0	16.67 m/s
s_0	2m
T	1.5s
a	1.2 m/s
b	3 m/s
δ	4

Table 3.1: Light IDM Parameters

Medium Traffic Parameters	
v_0	16.67 m/s
s_0	2m
T	1.5s
a	1.2 m/s
b	3 m/s
σ_{v_0}	3.75m/s
σ_{s_0}	0.2m
δ	4

Table 3.2: Medium IDM Parameters

Dense Traffic Parameters	
v_0	16.67 m/s
s_0	2m
T	1.5s
a	1.2 m/s
b	3 m/s
σ_{v_0}	5.75m/s
σ_{s_0}	0.4m
δ	4

Table 3.3: Dense IDM Parameters

3.1.3 MDP

The MDP for the problem is as follows:

1. **State Space**: A state in this environment is represented from the perspective of the agent. The feature set has a size of $d = 9$ i.e $S \in \mathbb{R}^9$. Therefore the feature of a state represented as $\phi(s) = \{lane, velocity, GF, GB, collision, OGF, OGB, OFV, OBV\}$. See Fig 3.2.

   - **lane**: This feature indicates the lane number in which the agent is currently in, lane $\in \{1, 2\}$.
   - **velocity**: This feature indicates the velocity of the agent, velocity $\in [0, 22]$ and velocity $\in Z^+$.
   - **GF**: Gap Front is the distance to the front vehicle.

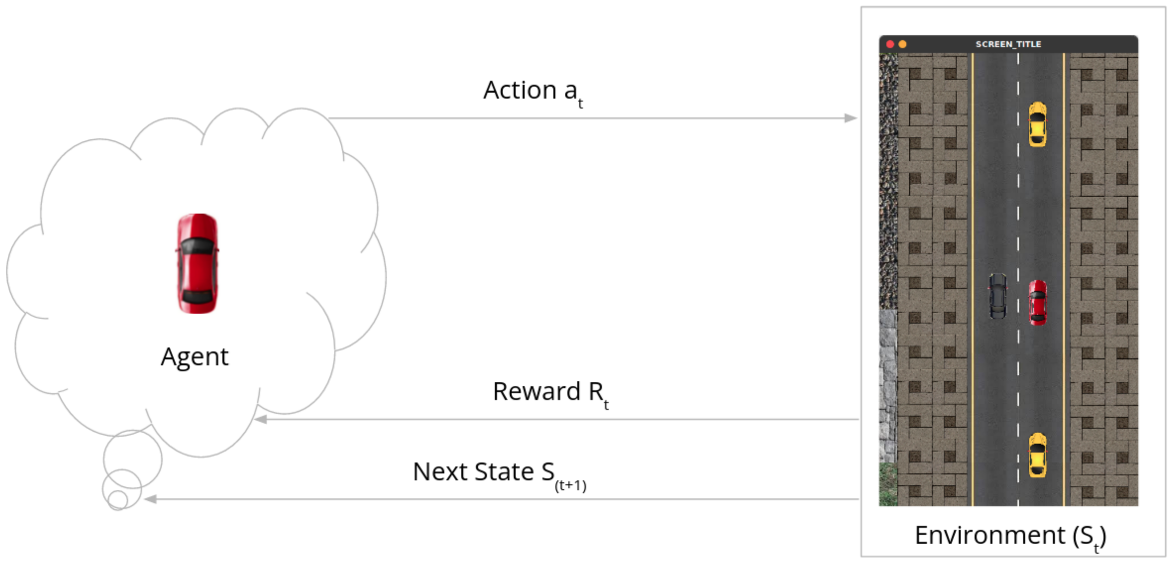

Fig. 3.1: RL Loop

- **GB**: Gap Back is the distance to the vehicle behind.
- **collision**: Collision flag indicates whether the agent collided with the environment vehicles or not, collision: O $\in \{0, 1\}$.
- **OGF**: Other lane gap front indicates the distance to the front vehicle on the lane in which agent is not present.
- **OGB**: Other lane gap back indicates the distance to the back vehicle on the lane in which agent is not present.
- **OFV**: Other lane front vehicle velocity indicates the velocity of the vehicle in front of the agent on the lane in which the agent is not present.
- **OBV**: Other lane back vehicle velocity indicates the velocity of the vehicle in back of the agent on the lane in which the agent is not present.

2. **Actions**: {Accelerate, Decelerate, Shift lane, Do Nothing}

3. **Transition Probability**: Unknown, hence it is a model free RL problem.

4. **Reward Function**: Linear function of states R(s) where $s \in S$. $R(s) = \theta^T.\phi(s)$.

$\theta \in \mathbb{R}^9$ are the reward weights.

5. **Discount Factor**: $\gamma = 0.99$

The task that we are trying to solve is for the agent to move past all the environment vehicles without colliding.

3.2 Single Agent Setup

The pygame simulator has been integrated with the stable baselines 2 library [19]. Stable baselines offers a plethora of RL algorithms out of which the DQN implementation is used to train the single agent. Stable baselines 2 uses tensorflow 1.15 as the backend in building the neural networks.

3.2.1 Training DQN Agent

The following results are trained on medium traffic environment setting since it can easily generalize to light and dense traffic scenarios. For every collision the episode is ended with a negative reward signal. The following are the hyperparameters used to train the DQN agent:

HyperParameters	
Total Training TimeSteps	100000
Total Number of Episodes	150 (600 timesteps per episode)
Deep Q Network Size	(256, 256, 256)
Replay Buffer Size	100000
Discount Factor	0.99
Batch Size	128
Learning Rate	0.0001
Training Frequency	4
Target Network Update Frequency	1000
Agent Max Velocity	22 m/s
Reward Weights (θ)	[0, 0.2, 0, 0, -400, 0, 0, 0, 0]

Table 3.4: DQN Hyperparameters

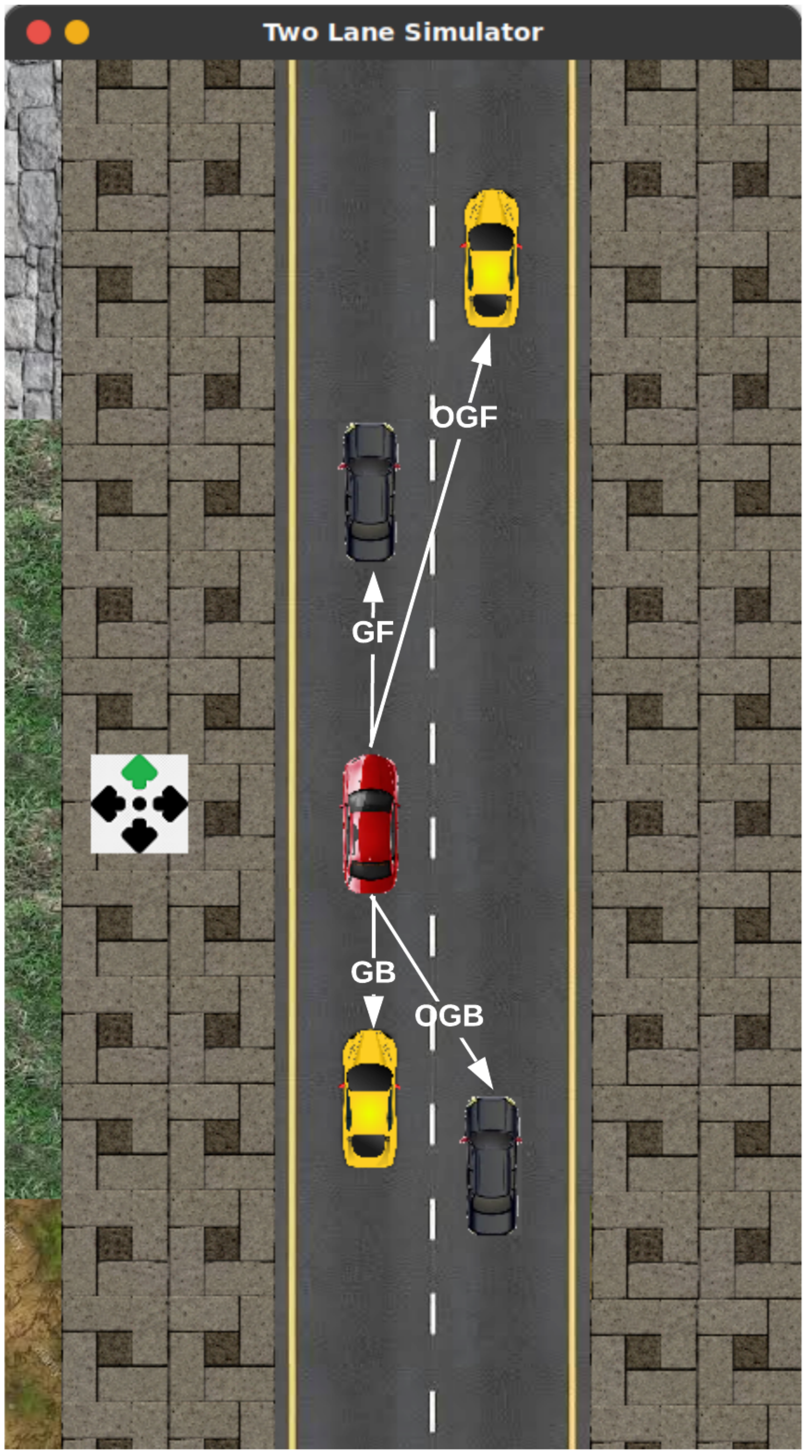

Fig. 3.2: (a) GF: Gap front (b) GB: Gap back (c) OGF: Other Lane front vehicle distance (d) OGB: Other Lane back vehicle distance

The reward weights indicate that the reward function is as follows:

$$R(s) = \theta^T.\phi(s) = \begin{bmatrix} 0 \\ 0.2 \\ 0 \\ 0 \\ -400 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} .\phi(s) = 0.2 \times velocity + (-400) \times collision \tag{3.1}$$

Results

The following are the average reward plot on all the types of DQN on medium traffic scenario. It is quite evident that Dueling DQN performs better than Double DQN and Vanilla DQN through the average reward plot and the collision plot.

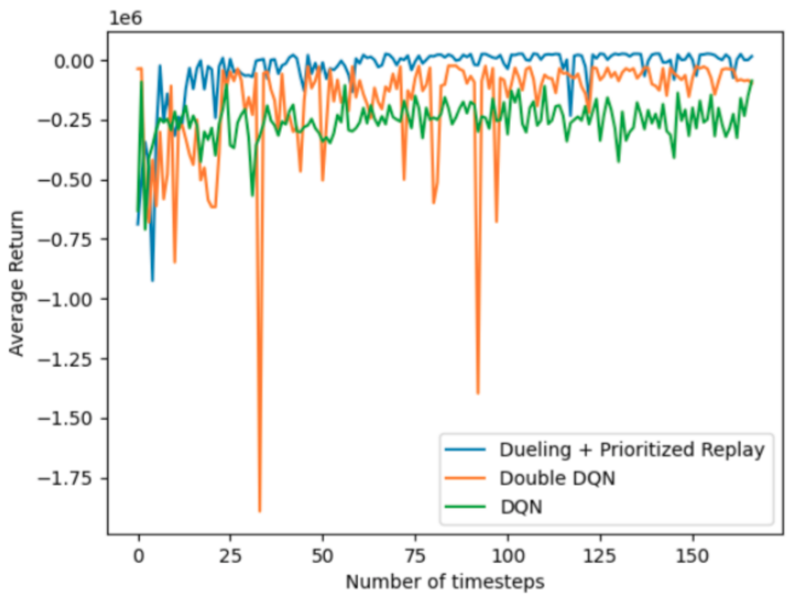

Fig. 3.3: Average Reward Plot

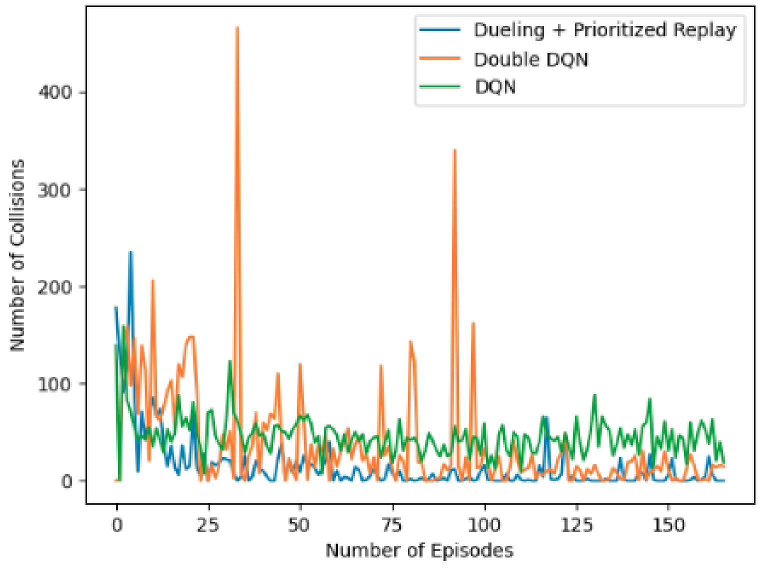

Fig. 3.4: Number of Collisions Plot

3.2.2 Simulations

The simulations can be found in the following drive link. The code is in a private repository, so please contact me so that I can provide access. Git Link

3.2.3 Conclusion

In this chapter we provide the MDP framework for the highway RL problem and provide a brief detail about the simulator used and the environment settings applied. We also discuss the results from the DQN agent on this environment and provide average reward plots of different types of DQN and collision rate. We can see from the simulation video that it is able to avoid collision in most cases.

Chapter 4

Agent in Ring Roads

4.1 MDP Formulation

The MDP for the problem is as follows:

1. **State Space**: A state in this environment is represented from the perspective of the agent. Although the full state information is available to the learning agent, it is practical to provide partial observations that the learning agent can physically sense. Therefore the feature of a state is represented as $\phi(s) = \{\ \frac{V_i}{V_0},\ \frac{\dot{X}_i}{V_0},\ \frac{X_i}{L_{track}}\}$, where i is the index of the learning agent, V_i corresponds to the velocity of the learning agent, V_0 corresponds to the max velocity of any vehicle in the environment, L_{track} corresponds to the track length, and $\dot{X}_i = V_{i-1} - V_i$.

2. **Actions**: {Accelerate, Decelerate}

3. **Transition Probability**: Unknown, hence it is a model free RL problem.

4. **Reward Function**: Linear function of states R(s) where $s \in S$.

5. **Discount Factor**: $\gamma = 0.99$

The task that we are trying to solve is to develop an agent behaviour that can reduce the traffic congestion waves in ring road systems.

4.2 Experiment Setup

The experiment contains 22 vehicles of which there are multiple autonomous vehicles along with human driven vehicles. The autonomous vehicles are governed by certain model based and learned control laws whereas the human driven vehicles are governed by IDM dynamics with an additional Gaussian acceleration noise of $\mathcal{N}(0, 0.2)$. The learning agents receive partial observations which restricts the observations to the information that can be directly sensed. All vehicles are equidistantly placed around the circular track before beginning the simulation with an initial velocity of $0m/s$.

4.2.1 Training Workflow

Each training episode is of $300s$ with a warmup period of $75s$. The warmup period overrides the control law of the autonomous vehicle with an IDM model. This allows for randomization of the initial state for the formation of stop-and-go-waves. In order to prevent overfitting to a singular track length, the lane density is randomly chosen from a range of traffic densities by uniformly sampling the total number of vehicles. In order for the learning agent to prevent crashes, certain fail-safe mechanisms have been implemented that introduces accelerations bounds.

4.2.2 Evaluation Workflow

Each evaluation episode is of $600s$ where for the first $300s$, the control laws of all the learning agents are overridden by IDM control to produce stop-and-go waves, and then the control is switched to the corresponding controller until the end of evaluation.

4.2.3 Reward Function

The reward function takes into account two important factors that aid in reducing the traffic waves, i.e the average velocity of the system and the control cost of the action. The rewarding nature of the two metrics is to positively reward increase in average velocity and penalize high accelerations. Therefore the reward function is as follows:

$$r(s,a) = \frac{1}{n}\sum_i v_i - \alpha\frac{1}{m}\sum_{j\in[m]} |a_j| \tag{4.1}$$

where $\alpha = 0.1$.

4.3 Control Laws

4.3.1 Model - Based Controller

Intelligent Driving Model

v_0	s_0	T	a	b	δ	noise
30 m/s	2m	1s	1 m/s	1.5 m/s	4	$\mathcal{N}(0, 0.2)$

Table 4.1: IDM Parameters

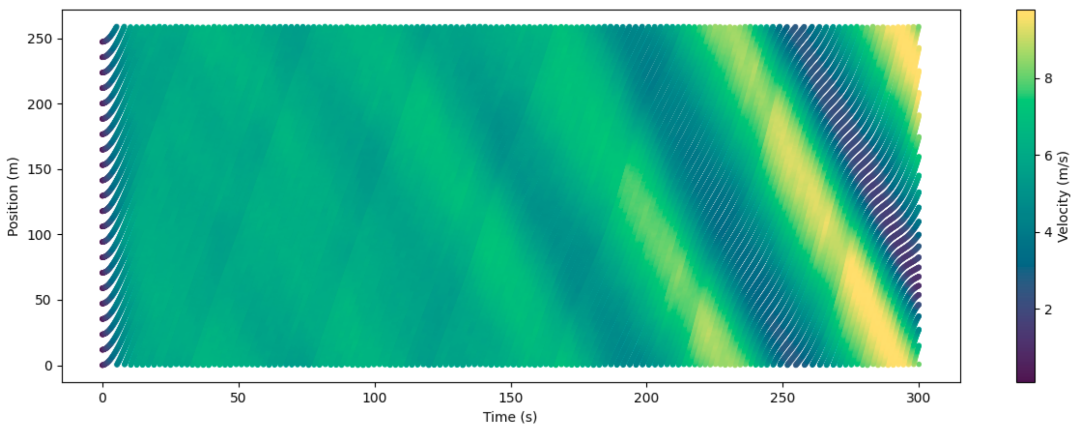

Fig. 4.1: Space Time Diagram of IDM control

The evaluation is run for 300s with all the vehicles in the ring road system following IDM model with the parameters given in 4.1. The space-time and velocity profile plot shows the occurrence of stop-and-go waves.

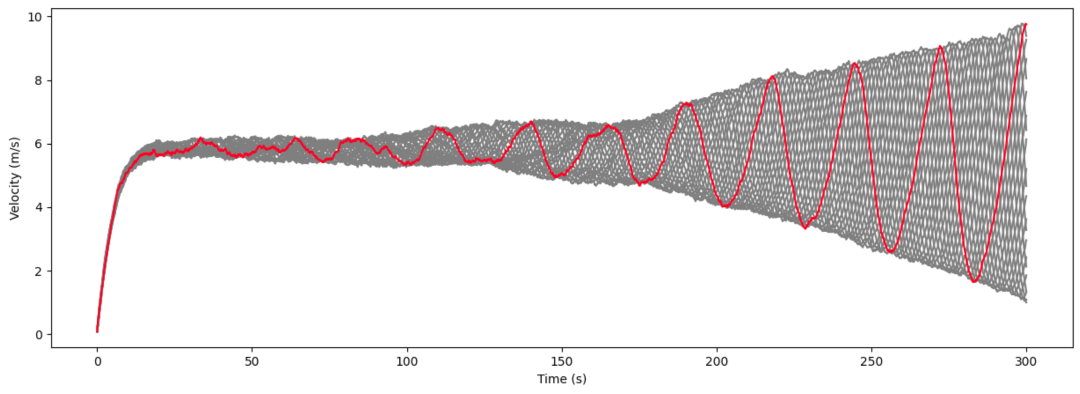

Fig. 4.2: Velocity profile of all vehicles, (red) agent vehicle, (gray) environment vehicles

FollowerStopper

d_1	d_2	d_3	Δv	Δx_1^0	Δx_2^0	Δx_3^0	U
$1.5m/s^2$	$1.0m/s^2$	$0.5m/s^2$	$-3m/s$	$4.5m$	$5.25m$	$6.0m$	$4.15m/s$

Table 4.2: FollowerStopper Parameters

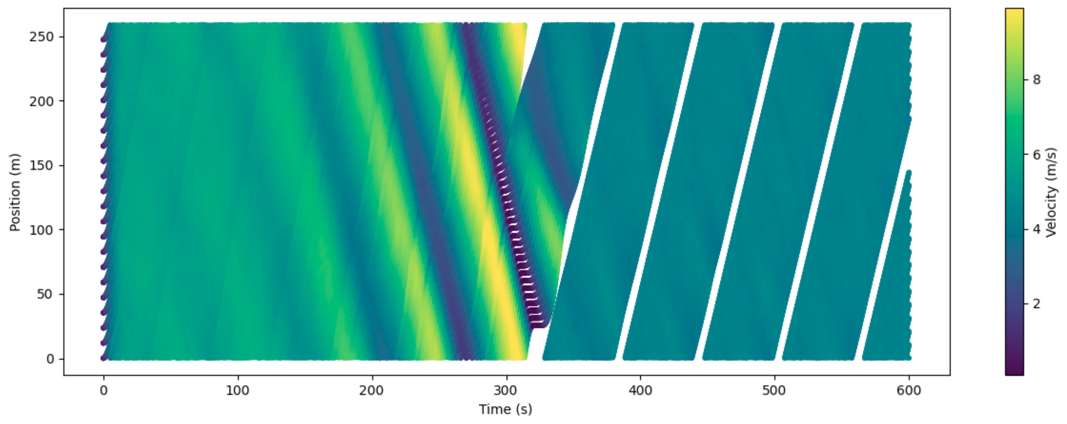

Fig. 4.3: Space Time Diagram of the simulation

The evaluation is run for 600s where the agent's control law is overridden by IDM for the first 300s to create stop-and-go waves and for the next 300s the followerstopper controller law is applied on the agent with the parameters from Table 4.2. The velocity of the agent tends to converge to the velocity $v_{agent} = 4.15m/s$.

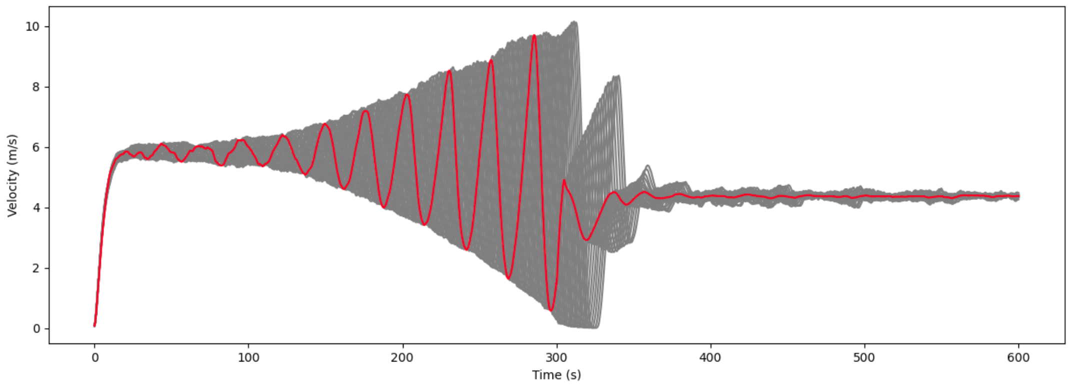

Fig. 4.4: Velocity profile of all vehicles, (red) agent vehicle, (gray) environment vehicles

Proportional Integral (PI)

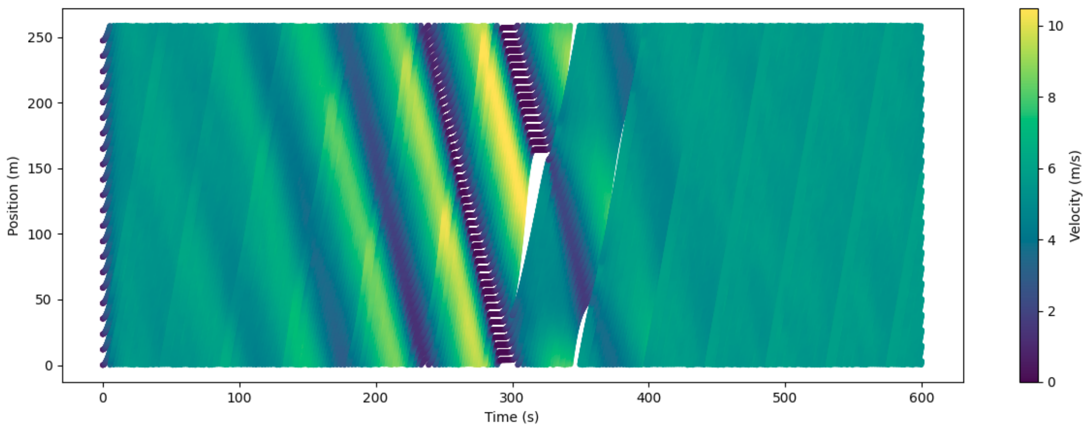

Fig. 4.5: Space Time Diagram of the simulation

δ	s_l	s_u	v_{catch}
$2m$	$7m$	$30m$	$1m/s$

Table 4.3: PI Saturation Parameters

The evaluation is run for $600s$ where the agent's control law is overridden by IDM for the first $300s$ to create stop-and-go waves and for the next $300s$ the PI Saturation controller law is applied on the agent with the parameters from Table 4.3. The velocity of the agent tends to converge to the velocity $v_{agent} = 4.85m/s$.

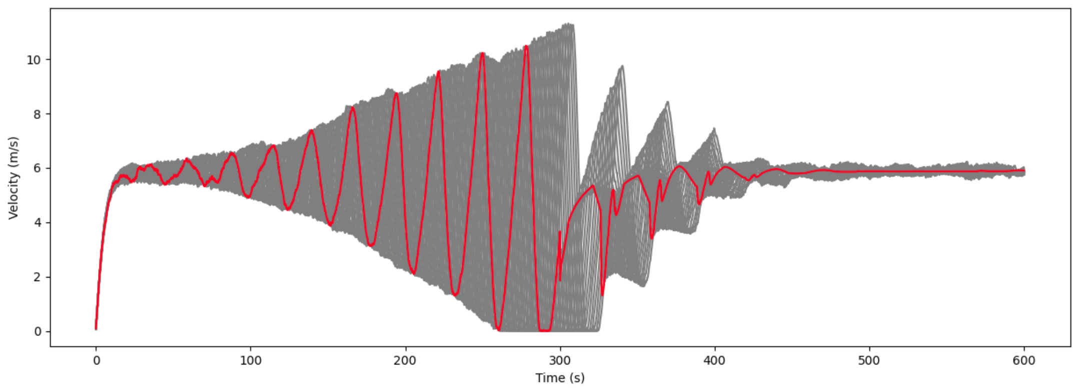

Fig. 4.6: Velocity profile of all vehicles, (red) agent vehicle, (gray) environment vehicles

4.4 Single Agent Learned Controllers

4.4.1 DQN Agent: Value Based Agent

The following are the hyperparameters used to train the DQN agent:

HyperParameters	
Total Training Timesteps	500k
Horizon	300s
Warmup Period	75s
Deep Q Network Size	(128, 128, 128)
Replay Buffer Size	1M
Discount Factor	0.99
Batch Size	32
Learning Rate	0.0001
Target Network Update Frequency	50
Action Set	$\{-1, 1\} m/s^2$
Agent Max Velocity	30 m/s

Table 4.4: DQN Hyperparameters

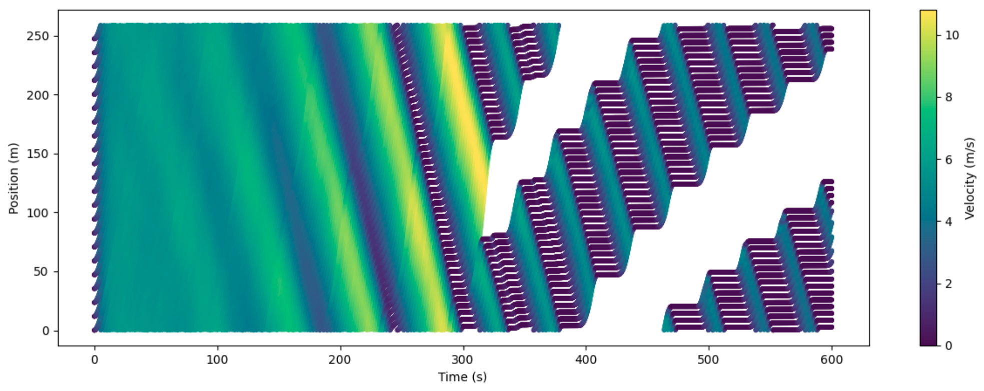

Fig. 4.7: Space Time Diagram with DQN control

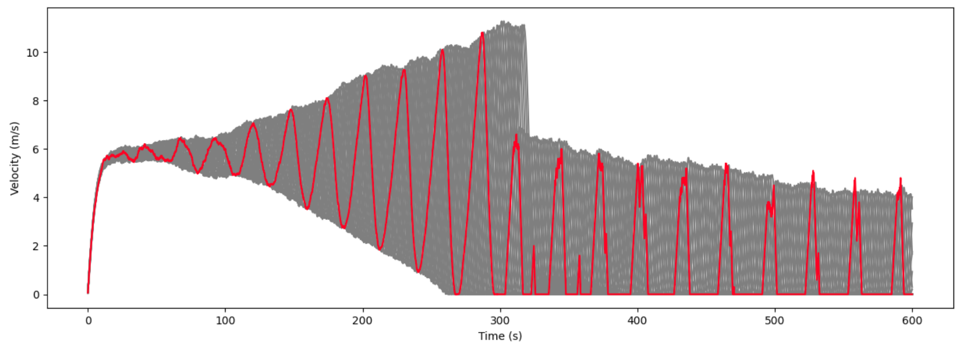

Fig. 4.8: Velocity profile of all vehicles, (red) agent vehicle, (gray) environment vehicles

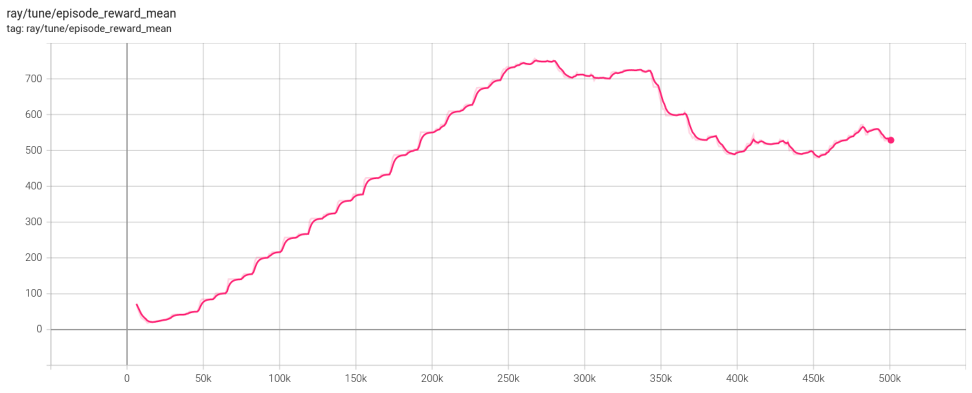

Fig. 4.9: Average Return Plot, Y-axis: Average Return, X-axis: Timesteps

4.4.2 PPO Agent: Policy Based Agent

The following are the hyperparameters for training the PPO agent:

HyperParameters	
Total Training TimeSteps	1M
Horizon	300s
Warmup Period	75s
Policy, Value MLP	(256, 256, 256)
Discount Factor	0.99
Batch Size	32
Learning Rate	0.0001
GAE Parameter	0.97
KL Target	0.02
Action Set	$[-1, 1]m/s^2$
Agent Max Velocity	$30m/s$

Table 4.5: PPO Hyperparameters

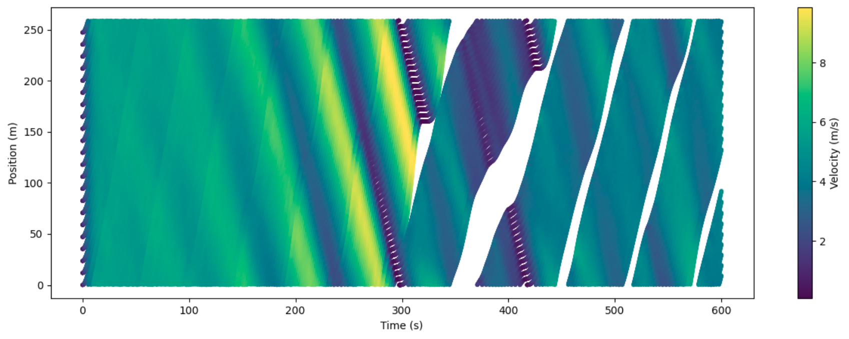

Fig. 4.10: Space Time Diagram with PPO Control

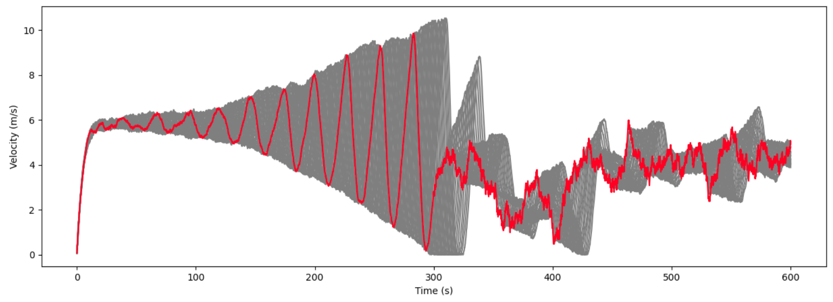

Fig. 4.11: Velocity profile of all vehicles, (red) agent vehicle, (gray) environment vehicles

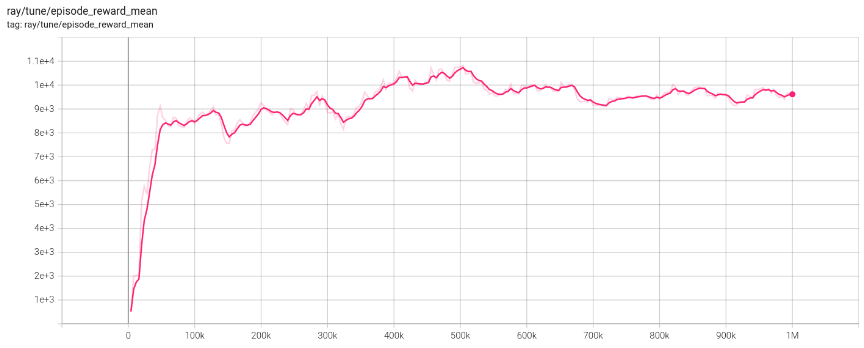

Fig. 4.12: Average Return Plot, Y-axis: Average Return, X-axis: Timesteps

4.5 Multi Agent Learned Controllers

4.5.1 Distributed Learning

HyperParameters	
Total Training TimeSteps	1M
Horizon	300s
Warmup Period	75s
Policy, Value MLP	(512, 512, 512)
Agent Type	PPO
Discount Factor	0.99
Batch Size	32
Learning Rate	1e-6
GAE Parameter	0.97
KL Target	0.02
Action Set	$[-1, 1]m/s^2$
Agent Max Velocity	$30m/s$
No. of Agents	3

Table 4.6: Distributed Learning Parameters

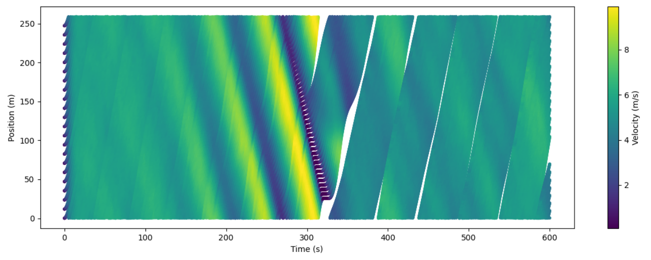

Fig. 4.13: Space Time Diagram with Distributed Learning

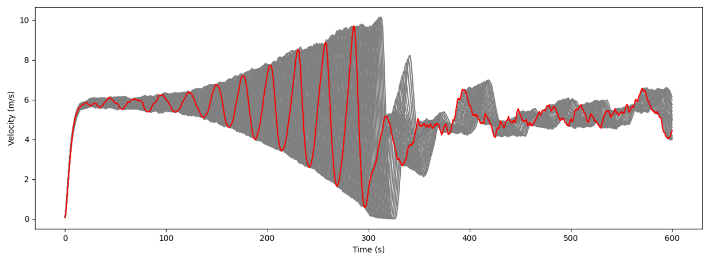

Fig. 4.14: Velocity profile of all vehicles, (red) agent vehicle, (gray) environment vehicles

4.5.2 Fully Independent Learning

HyperParameters	
Total Training TimeSteps	1M
Horizon	300s
Warmup Period	75s
Policy, Value MLP	(512, 512, 512)
Agent Type	PPO
Discount Factor	0.99
Batch Size	32
Learning Rate	1e-6
GAE Parameter	0.97
KL Target	0.02
Action Set	$[-1, 1]m/s^2$
Agent Max Velocity	$30m/s$
No. of Agents	3

Table 4.7: Fully Independent Learning Parameters

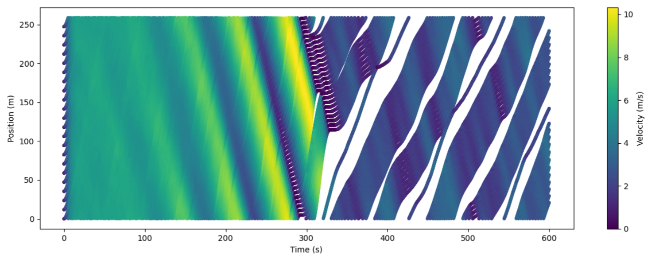

Fig. 4.15: Space Time Diagram with Fully Independent Learning

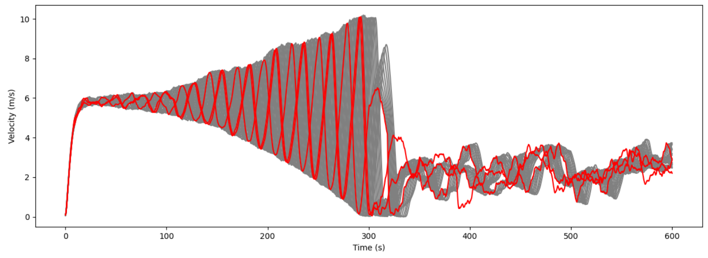

Fig. 4.16: Velocity profile of all vehicles, (red) agent vehicle, (gray) environment vehicles

4.5.3 Centralized Training Decentralized Execution

4.5.4 VDN

HyperParameters	
Total Training TimeSteps	1M
Horizon	300s
Warmup Period	75s
Discount Factor	0.99
Batch Size	32
Learning Rate	1e-6
LSTM Cell Size	32
Max Sequence Length	10
GAE Parameter	0.97
KL Target	0.02
Action Set	$[-1, 1]m/s^2$
Agent Max Velocity	$30m/s$
No. of Agents	3
Mixer	VDN

Table 4.8: VDN Parameters

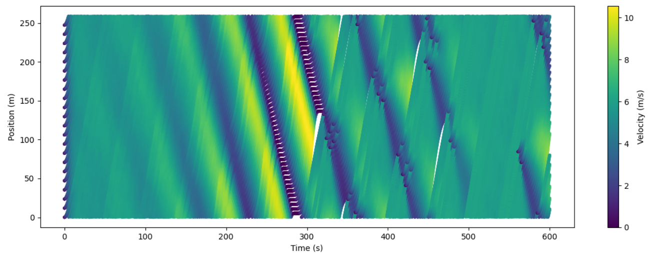

Fig. 4.17: Space Time Diagram with VDN Control

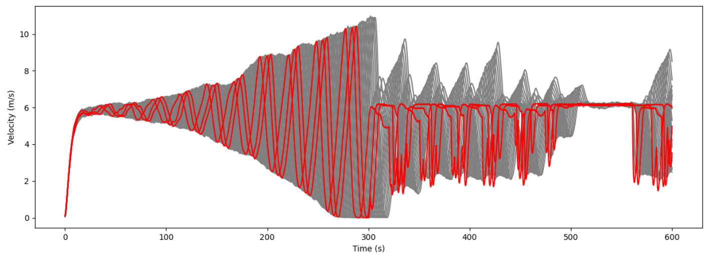

Fig. 4.18: Velocity profile of all vehicles, (red) agent vehicle, (gray) environment vehicles

4.5.5 QMIX

HyperParameters	
Total Training TimeSteps	1M
Horizon	300s
Warmup Period	75s
Discount Factor	0.99
Batch Size	32
Learning Rate	1e-6
LSTM Cell Size	32
Max Sequence Length	10
GAE Parameter	0.97
KL Target	0.02
Action Set	$[-1,1]m/s^2$
Agent Max Velocity	$30m/s$
No. of Agents	3
Mixer	QMIX

Table 4.9: QMIX Parameters

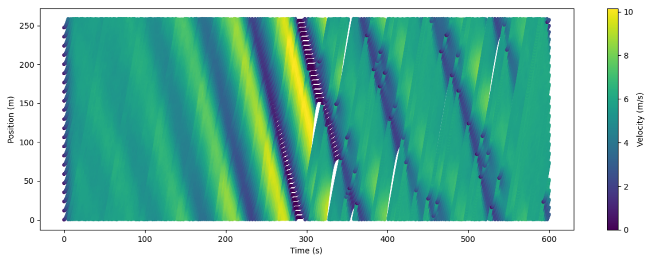

Fig. 4.19: Space Time Diagram with QMIX Control

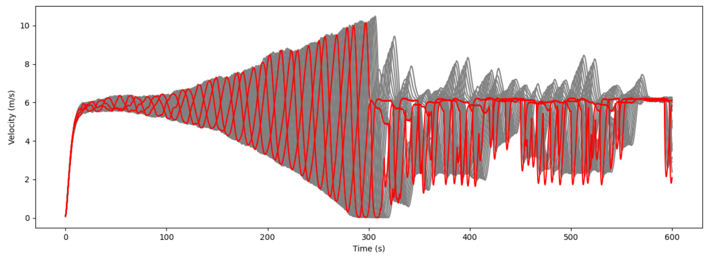

Fig. 4.20: Velocity profile of all vehicles, (red) agent vehicle, (gray) environment vehicles

4.6 Simulations

Simulations are stored in the following drive link. The code is in a private repository, so please contact me so that I can provide access. Git Link. The experiments were conducted on a server using NVIDIA Tesla V100-PCIE GPU and Intel(R) Xeon(R) Gold 6154 CPU @ 3.00GHz.

4.7 Conclusion

In this chapter we establish the training and evaluation workflows for an RL agent and explore the DQN and PPO agents which are value and policy based RL algorithms respectively. Along with these we have seen some model based control laws like IDM, followerstopper and PI saturation controller that act like a baseline. We have also explored the multi-agent system with different strategies of training implemented. Finally we can compare the results from single-agent and multi-agent systems.

Agent	Velocity (m/s)
FollowerStopper	4.15
PI Saturation	4.85
PPO Single Agent	5.2
PPO Distributed Learning	5.1
PPO Fully Independent Learning	2.4
VDN	5.8
QMIX	6

Table 4.10: Saturated Velocity

4.8 About the simulator

The simulator is built using PyGame library [18] and a snippet of the simulator is shown in Fig 4.21. The pygame simulator has been integrated with the Ray RLlib library [20]. RLlib is an open-source reinforcement learning (RL) framework that supports production-level, massively distributed reinforcement learning workloads while preserving consistent and easy APIs for a wide range of industry applications. The red color car indicates agent and the yellow cars indicate environment vehicles. The vehicles move in a clock wise direction and try to simulate a ring road setting. The movement of these vehicles are configured in such a way that it produces traffic congestion waves. The environment details are briefed in the Table 4.6.

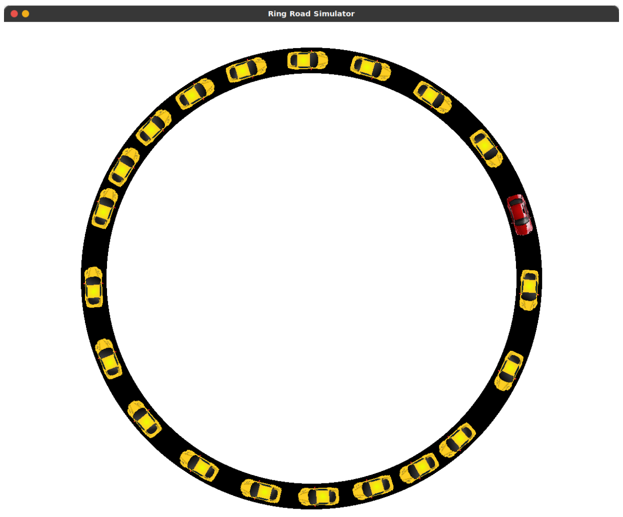

Fig. 4.21: Snapshot of the Simulator

Simulation Settings	
L_{track}	260m
$L_{vehicle}$	5m
Pixel Factor	10 pixels per m
Frame rate	10 FPS
Total number of vehicles	22

Table 4.11: Simulation Setting

Chapter 5

Conclusion and Future Work

We have successfully integrated the QMIX and VDN multi-agent algorithms into the ring road system and have extracted results which are as expected. This is the first ever work that combined CTDE algorithms with closed traffic systems like ring road. Moreover we have also implemented the two extremes of multi-agent reinforcement learning i.e independent agent learning and distributed learning. Overall the ring road scenario has been explored in the decentralized and centralized multi-agent setting and the results are promising.

Moving forward we expect to do the same with the highway scenario by integrating it with the Ray RLlib library and then applying CTDE algorithms for improved co-ordination within the multi-agent system and also to improve collision avoidance as an intelligent system.

References

[1] S. Bhatt, "Reinforcement Learning 101," https://towardsdatascience.com/reinforcement-learning-101-e24b50e1d292, 2008, [Online Website].

[2] L. Falk, "Deep Q Network: Combining Deep Reinforcement Learning," https://towardsdatascience.com/deep-q-network-combining-deep-reinforcement-learning-a5616bcfc207, 2021, [Online Website].

[3] A. Oppermann, "Deep (double) q-learning," Aug 2020. [Online]. Available: https://towardsdatascience.com/deep-double-q-learning-7fca410b193a

[4] Z. Wang, T. Schaul, M. Hessel, H. Hasselt, M. Lanctot, and N. Freitas, "Dueling network architectures for deep reinforcement learning," in *International conference on machine learning*. PMLR, 2016, pp. 1995–2003.

[5] K. Zhang and Z. Yang, "Tamer bacs ar. 2019. multi-agent reinforcement learning: A selective overview of theories and algorithms," *arXiv preprint arXiv:1911.10635*, 2019.

[6] P. Sunehag, G. Lever, A. Gruslys, W. M. Czarnecki, V. Zambaldi, M. Jaderberg, M. Lanctot, N. Sonnerat, J. Z. Leibo, K. Tuyls *et al.*, "Value-decomposition networks for cooperative multi-agent learning," *arXiv preprint arXiv:1706.05296*, 2017.

[7] T. Rashid, M. Samvelyan, C. Schroeder, G. Farquhar, J. Foerster, and S. Whiteson, "Qmix: Monotonic value function factorisation for deep multi-agent reinforcement

learning," in *International Conference on Machine Learning.* PMLR, 2018, pp. 4295–4304.

[8] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction.* MIT press, 2018.

[9] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton *et al.*, "Mastering the game of go without human knowledge," *nature*, vol. 550, no. 7676, pp. 354–359, 2017.

[10] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[11] J. Jumper, R. Evans, A. Pritzel, T. Green, M. Figurnov, O. Ronneberger, K. Tunyasuvunakool, R. Bates, A. Žídek, A. Potapenko *et al.*, "Highly accurate protein structure prediction with alphafold," *Nature*, vol. 596, no. 7873, pp. 583–589, 2021.

[12] Z. Li, C. Xu, and G. Zhang, "A deep reinforcement learning approach for traffic signal control optimization," *arXiv preprint arXiv:2107.06115*, 2021.

[13] A. Fares and W. Gomaa, "Freeway ramp-metering control based on reinforcement learning," in *11th IEEE International Conference on Control Automation (ICCA)*, 2014, pp. 1226–1231.

[14] D. Bertsekas, *Dynamic programming and optimal control: Volume I.* Athena scientific, 2012, vol. 1.

[15] M. Tan, "Multi-agent reinforcement learning: Independent vs. cooperative agents," in *Proceedings of the tenth international conference on machine learning*, 1993, pp. 330–337.

[16] D. Ha, A. Dai, and Q. V. Le, "Hypernetworks," *arXiv preprint arXiv:1609.09106*, 2016.

[17] M. Hausknecht and P. Stone, "Deep recurrent q-learning for partially observable mdps," in *2015 aaai fall symposium series*, 2015.

[18] P. Shinners, "Pygame," http://pygame.org/, 2011.

[19] A. Hill, A. Raffin, M. Ernestus, A. Gleave, A. Kanervisto, R. Traore, P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, and Y. Wu, "Stable baselines," https://github.com/hill-a/stable-baselines, 2018.

[20] E. Liang, R. Liaw, P. Moritz, R. Nishihara, R. Fox, K. Goldberg, J. E. Gonzalez, M. I. Jordan, and I. Stoica, "Rllib: Abstractions for distributed reinforcement learning. arxiv e-prints, page," *arXiv preprint arXiv:1712.09381*, 2017.