RISK ASSESSMENT OF AUTONOMOUS VEHICLE USING MARKOV DECISION PROCESS

by

Burak Derecik

B.S., Mechanical Engineering, Middle East Technical University, 2015

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Master of Science

Graduate Program in Industrial Engineering

Boğaziçi University

2020

ACKNOWLEDGEMENTS

I came along with great people who helped me accomplish this thesis and motivated me. Therefore, I would like to take the opportunity and express my thanks for their help in the work that has resulted in this thesis.

First of all, I would like to express my sincere gratitude to Prof. Dr Refik Güllü for giving me the opportunity for interesting topic, his patient supervision, helpful guidance and continuous support for this thesis.

I also want to thank all the members of the jury for their interest in my thesis and for taking the time to evaluate it.

Finally, I want to warmly thank my mother, my father, my brother and my friends who pushed me forward and motivated me in both good and bad times. This accomplishment would not have been possible without them.

ABSTRACT

RISK ASSESSMENT OF AUTONOMOUS VEHICLE USING MARKOV DECISION PROCESS

Autonomous decision making of intelligent vehicle is one of the most critical and challenging module due to the fact that traffic of real world is uncertain, complex, continuous and vehicles interact with each other. In this thesis, a decision making based on reinforcement learning algorithms is proposed to represent ego vehicle behaviors interacting with the stochastic behaviors of the environmental vehicles in highway traffic. The presented solver algorithms are formulated as Markov Decision Process (MDP) for autonomous vehicle problems.

Proposed algorithms are implemented in a simulation environment so that they are tested and analyzed with different scenarios. Then, efficiency of different implemented algorithms are compared based on specified criteria. The simulation results of tested scenarios show that ego car is capable of lane change and accelerate or decelerate in order to perform safe driving without any collision with other cars which have uncertain behavior in highway.

ÖZET

TEZ BAŞLIĞI

Akıllı aracın otonom karar vermesi, gerçek dünyanın trafiğinin belirsiz, karmaşık, sürekli olması ve araçların birbiriyle etkileşime girmesi nedeniyle en kritik ve en zorlu modüldür. Bu tezde, karayolu trafiğinde, çevresel araçların stokastik davranışlarıyla etkileşime giren ego araç davranışlarını göstermek için, pekiştirme öğrenme algoritmalarına dayalı bir karar alma önerilmektedir. Sunulan çözücü algoritmalar, otonom araç problemleri için Markov Karar Süreci olarak formüle edilmiştir.

Önerilen algoritmalar, farklı senaryolarla test ve analiz edilmesi için bir simülasyon ortamında uygulanır. Daha sonra, uygulanan farklı algoritmaların verimliliği belirtilen kriterlere göre karşılaştırılır. Test edilen senaryoların simülasyon sonuçları, ego otomobilinin otoyolda belirsiz davranışı olan diğer otomobillerle çarpışmadan güvenli sürüş yapmak için şerit değiştirme ve hızlanma veya yavaşlama yeteneğine sahip olduğunu göstermektedir.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

A	Set of actions
a	Action
$a^{'}$	Next action
G	Expected return
Q_{*}^{π}	Optimal action-value function under policy π
q_{π}	Action-value function under policy π
R	Set of rewards
r	Reward
S	Set of states
s	State
$s^{'}$	Next state
T	Set of conditional transition probabilities
V_{*}^{π}	Optimal state-value function under policy π
v_{π}	State-value function under policy π
α	Learning rate
β	Behavior policy
γ	Discount factor
δ_{t}^{Q}	TD error of Q
δ_{t}^{V}	TD error of V
π	Target policy
π_{*}	Optimal policy
$\pi^{'}$	New policy
ρ	Response time delay
τ	Computational temperature
Ω	Set of observations

LIST OF ACRONYMS/ABBREVIATIONS

ADAS	Advanced Driving Assistance Systems
DP	Dynamic Programming
MC	Monte Carlo
MDP	Markov Decision Process
POMDP	Partially Observable Markov Decision Process
RL	Reinforcement Learning
RSS	Responsibility-Sensitive Safety
TD	Temporal Difference

1. INTRODUCTION

In the global world, according to study "Global status report on road safety 2018" , number of people who die due to road traffic have reached more than 1.3 million every year [1]. The vast majority of these deaths are caused by crash-causing deficiencies such as tiredness, breakdown because of alcohol or drug, distraction, dormancy. Obviously, autonomous vehicles are not affected by these conditions and it can be considered as potential that number of deaths dramatically reduces when crash-causing deficiencies which human drivers have eliminate. Therefore, main objective of autonomous vehicles is to make sustainable transportation safely, comfortably and efficiently. As a result of this objective, autonomous vehicles will potentially give positive impact on the society.

The first autonomous vehicle was only able to perform lane-following based on camera images collected by environment and it was developed by Carnegie Mellon University's Navlab in 1988. [2] First trip was completed with average speed 63.8 mph between Pittburgh and San Diego by Navlab in United States in 1995. [3] A major milestone in self-driving vehicle was the DARPA Urban (Grand) Challenge, which was performed 3 times between 2004 and 2007. While Autonomous vehicles race at off – road environment in 2004 and 2005, they race at urban area in 2007, which was the first major demonstration that vehicles were able to operate autonomously in an urban environment. [4] After this milestone, research and development in autonomous systems has dramatically increased. In recent years, enhancement of computing power, data processing, mapping and decrease of cost of sensing and computing technology result in increasing of development of autonomous vehicles and make possible to see on road. By reason of software development, many OEMs, startups, and research organizations have focused on building their own autonomous vehicles or tools that helps to build autonomous vehicles. As a result, Advanced Driving Assistance Systems (ADAS) such as parking assistants and adaptive cruise control and intelligent vehicles have been developed to achieve aim of safe road transportation without human drivers.

In general, autonomous driving divides 6 SAE levels which start from 0 to 5. In level 0, human driver performs full-time control in vehicle and there is no autonomy. However, level 5 is full autonomous driving under all roadway and environmental conditions and there is no human interaction. [7] Summary of levels of driving automation for on-road vehicles can be seen at following Figure 1.1

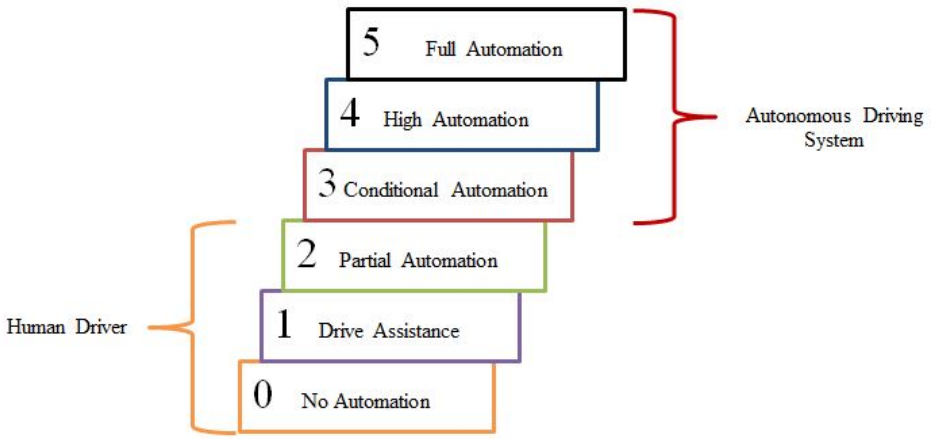

Figure 1.1. Autonomous Vehicle SAE Levels.

Intelligent vehicle system can mainly be divided into following modules: perception such as sensing and map-building, path planning, where the autonomous vehicle plans its future trajectory as a function of time or space [5] [6], decision-making, where appropriate actions from possible action sets are chosen by autonomous vehicle for next time step and dynamic control. [7] [8]. While environment is detected by sensors such as radar, GPS, lidar by means of computer vision techniques including recognition, scene understanding etc. [9], intelligent vehicles make decisions according to detected surrounding data and perform path planning and appropriate driving maneuvers such as lane changing, overtaking [10] [11]. Ultimately, dynamic control conducts actions such as brake, throttle and steering of autonomous vehicles.

Autonomous decision making of intelligent vehicle is one of the most critical and challenging module since real-word traffic is complex and uncertain. Rule based decision making, decision making based on probabilistic model and employment of machine learning algorithm for making decision are three general classes which can be

divided into for previous decision making approaches.

Rule - based decision making is the first class of decision making approaches. It is also called as expert systems [12], knowledge – based system or fuzzy. It was proposed by [13] for intelligent vehicles on highway. In this paper, controlled dynamic system's state and its dynamic environment' evolution is predicted by worst-case decision making method in order to overcome uncertainties in decision making. In papers [12] and [14] written by Researcher from Carnegie Mellon University, prediction- and cost function based algorithm (PCB) is proposed to perform robust highway driving for autonomous vehicles. Both proposed algorithms were tested and verified in DARPA Urban Challenge 2007. It was seen that autonomous vehicle performance is enchanced in performing distance keeping, lane selecting and merging on freeways. However, main disadvantage of rule-based methods is that algorithms are tuned according to predefined parameters for specific environment. Although this method has easy implementation, it has not enough capability to properly work when agents face with unforeseen situations due to the fact that only prescribed decisions are allowed and it needs a lot of prior knowledge. Therefore, rule-based methods are not suitable for real world applications such as different traffic situations.

Using probabilistic models to overcome uncertainties is the second class of decision making approaches. For lane change decisions, [10] proposed a probabilistic online decision making for self-driving. [15] and [16] proposed decision making approaches using Bayesian networks to deal with uncertainties starting from perception. However; it is difficult to implement Bayesian networks for complex and dynamic environments. At this point, agent may be in stochastic environment. It means that environment may be observed by agent as partially, and it can be modelled as Partially Observable Markov Decision Process. (POMDP). Application of autonomous driving for POMDP [17] is offered in different domain such as merging [18], urban [19] and lane change [13].

Third and last technique is to use idea from artificial intelligence (AI) to establish decision functions based on observation data. It includes supervised learning [20],

reinforcement learning [21] and deep learning [22]. In supervised learning, policies for decision making can be performed according to labeled data. However, collection of enough labeled data based on human experiences is a hard and challenging task. Reinforcement learning is used to solve sequential autonomous vehicle decision making problems that involve interacting with the environment. In RL, agents learn how to act or make decisions based on past experiences. It can be applied to any type of sequential decision making problem such as autonomous vehicle decision making. Optimal or near optimal policy using function approximation is obtained using RL algorithms without model information. Advantage of this method is that when sufficient informative data is being well trained, agent can deal with even unforeseen situations as in real world. However, learning efficiency of agent can be low. In addition to learning efficiency, there are many ways to approximate function [23], [24] and studies are still ongoing. Tabular reinforcement learning which is called Q – learning was proposed as multi-goal decisions in [25] and [26] or overtaking and lane changing. Zheng [27] proposes single objective decision making method for self-driving vehicles based on least-squares policy iteration and simple traffic simulation with single decision objective. The last artificial intelligence (AI) technique for decision making is deep reinforcement learning [28]. Deep reinforcement learning is the outcome of applying reinforcement learning using deep neural networks. This technique is also efficient training of arbitrary policies for specific goal. Tactical decision using DRL is being performed by [29] for making Lane change in highway driving. Proposed approach was able to outperform a simple rule-based approach and a human driver with different metrics and much safer policy. Main problem of deep learning approaches is that system is basically a blackbox. It is a serious problem for development of self driving cars since it is not comprehensible that how underlying network properly works. Instead, automotive companies prefer transparency of system design and interaction of subsystems with each other.

In this thesis, in order to capture dynamics of complex and real traffic environment such as accelerating or decelerating during driving to ensure safety or comfort criteria, making maneuver for lane change to avoid crash or traffic congestion caused by too many vehicles in long-term, or reaching a goal location like an exit in high-

way, a tactical to strategic decision making based on reinforcement learning algorithms modeled as Markov Decision Process (MDP) will be proposed to represent ego vehicle behaviors interacting with the stochastic behaviors of the environmental vehicles in highway traffic. Contribution in this thesis focuses on developing RL so that while agent that interacts with the stochastic behaviors of the environmental vehicles can make safe lane changes, it can also robustly decide acceleration/deceleration in a dynamic highway. Realistic and dynamic environment is simulated in order to train many different scenarios. The rest of the thesis is organized as follows: Chapter 2 introduces concept and background of RL algorithms as well as MDP. Chapter 3 covers mathematical background of proposed algorithms such as Q-learning and Sarsa. Chapter 4 mentions formulation of autonomous vehicle problem including reward function, terminal state and collision detection. Chapter 5 implements the proposed RL algorithms in a simulation environment, analysis and results of test scenarios, and comparison of algorithms. Finally, Chapter 6 summarizes the conclusion and future work of this study.

2. BACKGROUND – CONCEPTS

2.1. Markov Decision Process (MDP)

Markov property is an important concept in order to understand MDP. Markov property states that conditional probability of distribution at next step depends only on the present states and does not depend on past history.

MDP is a tuple which has the following elements:

- Set of states is S
- Set of actions is A
- $\mathbb{P}\{S_{t+1} = s'_t | S_t = s_t, A_t = a_t\}$ is state transition probability
- γ is a discount factor $\in [0, 1)$
- $R : S \times A \to \mathbb{R}$ is reward function.

MDP can be described as follows: agent which is in state s_t takes and action $a_t \in A$ at time step t. Agent receives a reward as a result of selected action. According to transition probability $P(s'|s, a)$ transition from current state s_t to next state s_{t+1} can be performed. Then, once agent is in state s_{t+1}, it takes an action a_{t+1} based on the current state s_{t+1} and receives a reward which is $r(s', a') = \mathbb{E}\{R_{t+2}|S_{t+1} = s', A_{t+1} = a'\}$. Agent moves the new state s_{t+2}. Iterative interaction repeats until predetermined time horizon T is reached. Agent takes an action according to policy which is called $\pi : S \to A$ and it follows states $s_t, s_{t+1}, s_{t+2}, \cdots$ and receives discounted rewards as following:

$$r(s_t, a_t) + \gamma r(s_{t+1}, a_{t+1}) + \gamma^2 r(s_{t+1}, a_{t+1}) + \cdots \tag{2.1}$$

Importance of rewards which is taken by agent according to action in followed states can be specified using discount factor γ. If discount factor is 1, every reward in each

step has equal importance. If it is less than 1, importance of future rewards decreases. It is important that discount factor must be between 0 and 1. Sum of the rewards that agent takes according to selected action and policy is called returns. The goal of this iteration is to reach an optimal policy which maximizes returns. The agent - the environment relation can be seen in Figure 2.1.

Figure 2.1. Agent - Environment Relation.

2.2. Value Functions and Policies

Value functions and policies are two important concepts in reinforcement learning. Value functions consist of two functions: state-value functions and action value functions. Reinforcement learning algorithms are trying to approximate expected future rewards which agent is receiving as a result of action in certain state. In order to perform this, value functions and policies should be optimized using RL algorithm. According to policy π which is the mapping formed by agent in each state $s \in S$ under each action $a \in A$ and the probability $\pi(a, s)$ of taking action a, value functions are designed. Aim of the agent is to maximise the expected return which can be defined as G_t. It is a specific function of sequences of reward $r_{t+1}, r_{t+2}, r_{t+3}, \cdots$. The return G_t is the sum of rewards and it can be defined as:

$$G_t = r_{t+1} + r_{t+2} + \cdots + r_T \tag{2.2}$$

The state-value function under policy π is defined as following :

$$v_\pi \doteq \mathbb{E}\left[G_t|S_t = s\right] = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s\right] \tag{2.3}$$

and the action-value function under policy π is equal to the expected return when agent is in state s and it is taking action a under policy π. It can be defined as:

$$q_\pi(s,a) \doteq \mathbb{E}_\pi\left[G_t|S_t = s, A_t = a\right] = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s, A_t = a\right] \tag{2.4}$$

The relationship between the value of current state s and its successive state s_{t+1} can be expressed based on Bellman's Equation since value functions described Equation 2.3 and Equation 2.4 satisfy the recursive properties.

$$\begin{aligned} v_\pi &\doteq \mathbb{E}\left[G_t|S_t = s\right] \\ &= \mathbb{E}_\pi\left[R_{t+1} + \gamma G_{t+1}|S_t = s\right] \\ &= \sum_a \pi(a|s) \sum_{s'} \sum_r p\left(s', r|s, a\right)\left[r + \gamma \mathbb{E}_\pi\left[G_{t+1}|S_{t+1} = s'\right]\right] \\ &= \sum_a \pi(a|s) \sum_{s',r} p\left(s', r|s, a\right)\left[r + \gamma_\pi(s')\right] \end{aligned} \tag{2.5}$$

Bellman's Equation which is a mathematical optimization method known in dynamic programming gives the recursive relationship between value of current state and value of next (successor) states. Equation represents that sum over all possible actions a represents the sum between the obtained reward value and the discounted expected value of the return for the next state.

2.2.1. Optimal policy

In the reinforcement learning problem, aim is to find the optimal policy π_* which maximize the long term rewards. Optimal policy π_* is the policy that achieves optimal

value function and it is at least as good as among all other policies.

- $\pi_* \geq \pi$,where $\forall \pi$

It is possible to find more than 1 optimal policy but all optimal policies achieve the optimal value function and optimal action-value function.

- $V_*^{\pi}(s) = V_*(s)$,where $\forall s$
- $Q_*^{\pi}(s,a) = Q_*(s,a)$,where $\forall s, a$

This means that policy π is better than π_* if and only if corresponding value $V^{\pi}(s)$ is greater or equal to value $V_*^{\pi}(s)$ for all $s \in S$. Therefore, following two Equations 2.6 and 2.7 give unique solutions as a result of Bellman's Optimality Equation. Optimal policy π_* and corresponding state-value function can be described as optimal state-value function seen as following Equation 2.6 for all $s \in S$.

$$v_*(s) \doteq \max_{\pi} v_{\pi}(s) \tag{2.6}$$

Similarly, optimal action-value function can be described for all $s \in S$ and $a \in A$ as following Equation 2.7:

$$q_*(s,a) \doteq \max_{\pi} v_{\pi}(s,a) \tag{2.7}$$

"Expected sum of rewards called return for best action in state must equal to value of this state under an optimal policy". This statement provided by Bellman's Optimality Equation as follow for v_*:

$$\begin{aligned} v_*(s) &= \max_{a \in A} q_*(s,a) \\ &= \max_{a \in A} \sum_{s',r} p(s',r|s,a) \left[r + \gamma v_*(s') \right] \end{aligned} \tag{2.8}$$

Similarly, Bellman's Optimality Equation for action-value function can be defined as:

$$q_*(s,a) = \sum_{s',r} p(s',r|s,a) \left[r + \gamma \max_{a' \in A} q_*(s',a') \right] \tag{2.9}$$

A greedy optimal policy from the optimal Value function in Equations 2.10 and 2.11.

$$\pi_*(s) = \underset{a \in A}{\operatorname{argmax}}\, q_*(s,a) \tag{2.10}$$

$$\pi_*(s) = \underset{a \in A}{\operatorname{argmax}} \sum_{s',r} p(s',r|s,a) \left[r + \gamma v_*(s') \right] \tag{2.11}$$

Argument maximum is defined as the fact that the values of a function in a specified domain give maximum outputs. In optimal value function, action is the out of $\pi_*(s)$ function. However, argmax function might give more than 1 output. It means that the optimality can be achieved with different actions taken by agents. There are many strategies to specify the action to be taken. The simplest one is to choose first optimal action in the set of optimal actions. It is important to note that it is possible to chose optimal action from Equation 2.10 without using $p(s',r|s,a)$. It means that optimal action can be specified without using dynamics and model of environment. It is called as "Q-learning" which is one of the most popular reinforcement learning techniques of model free value iteration algorithm. [30]

2.3. Partially Observable Markov Decision Process (POMDP)

Markov Decision Process assumes that states are known perfectly by agents. However, in reality, it might not be generally possible that agent can directly gain perfect knowledge about states whereas MDP meets opposite assumption. Instead of this, agent can observe real states at each discrete point in time and it forms belief about system's current states. For example, there are many sensors such as camera, lidar, GPS, radar which are used in autonomous vehicle applications. However, it is

well known that these sensors are always noisy and inaccurate. Also, according to agent cars position and environment like cars behind corners, they can only provide limited/partial information about world when agent cars try to perceive environment via these sensors. A discrete time framework of Partially Observable Markov Decision Process(POMDP) can be described by 7 tuple (S , A , T , R , Ω , O , γ) where

- S is the set of state space as $\{s_1, s_2, s_3, ..., s_n\}\, for\, s \in S$
- A is the set of actions as $\{a_1, a_2, a_3, ..., a_m\}\, for\, a \in A$
- T is the set of conditional transition probabilities as $T(s'|s, a)$ for the state transition $s \to s'$
- R is the reward function as $S \times A \to \mathbb{R}$
- Ω is the set of observations as $\Omega = \{o_1, o_2, o_3, ..., o_k\}\, for\, o \in \Omega$
- O is the set of conditional observation probabilities as $O(o|s', a)$
- γ is a discount factor $\in [0, 1)$

When the agent is in some state $s \in S$ at each time period, and it chooses an action $a \in A$, this leads to transition of state with probability $T(s'|s, a)$ from s to $s' \in S$. Simultaneously, observation $o \in \Omega$ with probability $O(o|s', a)$ is taken by agent based on the environment where agent goes in. Eventually, agent takes reward $R(s, a)$. Agent repeats this process and goal is to maximize expected future discount reward (return).

2.4. Function Approximation

There are two types of approximations in RL problem: Linear and non-linear approximation. In the discrete space MDP, value function is represented by lookup table. Every state s or state-action pair (s, a) has corresponding $V(s)$ or $Q(s, a)$ value, respectively, and it is stored in lookup table. However, these values became too large to store in the table if space is continuous or discrete which has too large data sets. Thus, curse of dimensionality problems occur. It means that it is impossible or incredibly difficult to store q-values of every state-action pair if the state space is very large or

continuous. In order to solve RL problems, training data should keep each visited value individually in array, and due to curse of dimensionality problem, enough training data can not be stored.

In order to deal with high-dimensional state and action spaces in very large data sets, one common solution in MDP is to use value function approximations as generalization of state features instead of using each value. Function approximation can also be used to estimate policies and real Q values can be approximated by raising approximation function $\hat{Q}(s, a)$. The parameters of both policy and value approximation functions can be learned from experience and these can be described as $\theta_p = (s_0, a_0, r_0, s_1, a_1, r_1, ...)$ or $\theta_v = (S_t, A_t, G_t)$, called trajectories or rollouts.

It is important to note that a substantial drawback of function approximation is convergence. While it has been proven that [31] convergence can be reached empirically in Q learning algorithm with linear function approximations, function approximations are never theoretically guaranteed that convergent result is obtained.

2.5. Exploration and Exploitation

Exploration and exploitation are the fundamental concepts in reinforcement learning. Basically, agent searches whole space and gathers more information according to specified policy in order to use it to give better decisions in the future. It is called exploration. Then, agent takes actions according to gained information from exploration process. It is called exploitation. These two concepts make the learning process of agent possible. However, there is a trade-off between the two concepts. Due to the fact that agent explores a limited set of environment, it can only use explored information which is also limited. However, if unexplored information is available, agent might not take optimal action. It means that, there can be unexplored states which have higher unknown reward than known reward at current state. However; Q-values can converge to optimum Q by 2 conditions, which has been proved by [32]

- Every state-action pairs must be visited infinitely often. Actions are chosen randomly with a uniform distribution over the action space. (exploration)
- However, return should be maximized according to decided policy (exploitation)

There are many of exploration methods, and mostly used ones are summarized below. Note that, in this thesis, ϵ-greedy will be used due to efficiency and easy implementation.

ϵ-greedy policy: It can be defined as while greedy action is chosen by probability 1-ϵ at each time-step, the other actions are chosen with probability ϵ. In the long term, we would like to be confident that exploration is enough to learn information and Q-values can be used reliably in the future. Then, we should start to exploit this information. It can be achievable if ϵ can be decreased via decaying rate over time. Theoretical information can be found in [21]

$$a^*{}_t = \underset{a}{\operatorname{argmax}}\, Q_t(a) \tag{2.12}$$

$$a_t = \begin{cases} a^*{}_t & with\, probability\, 1-\epsilon \\ random\, action & with\, probability\, \epsilon \end{cases} \tag{2.13}$$

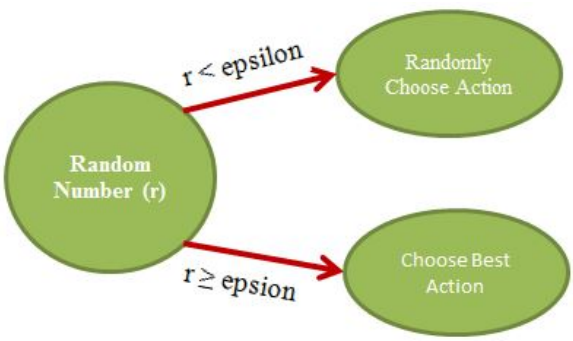

Figure 2.2. Action Selection Method.

Softmax or Boltzmann Method: Action a can be chosen at time step t with probability

$$p(a_t) = \frac{e^{Q_t(a)/\tau}}{\sum_{a'} e^{Q_t(a')/\tau}} \tag{2.14}$$

Where τ is the computational temperature, and when decreasing of this value result in decrease of exploration. Method performance depends on the action situation. When the actions are close to each other, method does not give good result. [33]

According to [21], it is not clear that ϵ-greedy action or softmax action gives better performance. One should select one of them considering task and heuristic of model.

2.6. On-Policy and Off-Policy Concepts

We have already mentioned the exploration – exploitation dilemma which is an important concept of reinforcement learning on previous Section. Agents in non-optimal policy seek the optimal policy by exploring all actions and exploit this optimal policy to get maximum return. In this Section, we will mention two important concepts which are on-policy and off policy as a result of exploration-exploitation dilemma when agents decide how to change action values in order to reach a better result in the long term.

Before mentioning about on-policy and off-policy, it is important to note that there are two types of policy: behavior (β) and target policy (π). Target policy is how the agent learns the optimal policy and behavior policy is the policy used to generate training data or samples. In on policy methods, the behavior policy is the same as the target policy. Thus, the agent learns optimal policy and behaves using the same policy. However, in off policy methods, target policy which is evaluated and improved is different than the behavior policy. It can be said that it is an advantage since while all possible actions are sampled by behavior policy continuously; target policy such as epsilon greedy may be deterministic.

While Q-learning is mostly used and most famous method of off-policy, SARSA method can be given as an example for an on-policy methods. In order to explain better, agent can use absolute greedy policy to learn optimal policy and it can behave ϵ-greedy policy in Q learning. It can be seen that there is a difference between update and behavior policy since Q learning is type of off-policy method. However, if same policy such as ϵ-greedy is used during learning and behaving period of agent, etc. SARSA, it can be called on-policy. Obviously, that there is no difference between update and behavior policy in SARSA which is type of on-policy method. We will cover these methods on next Chapter.

3. REINFORCEMENT LEARNING METHODS

This chapter introduces the two main types of reinforcement learning methods: Model-Based and Model-Free which include algorithms of different methods, their mathematical background, advantages and limitation of these methods. Classification of Model-Based and Model-Free methods can be seen in following Figure 3.1

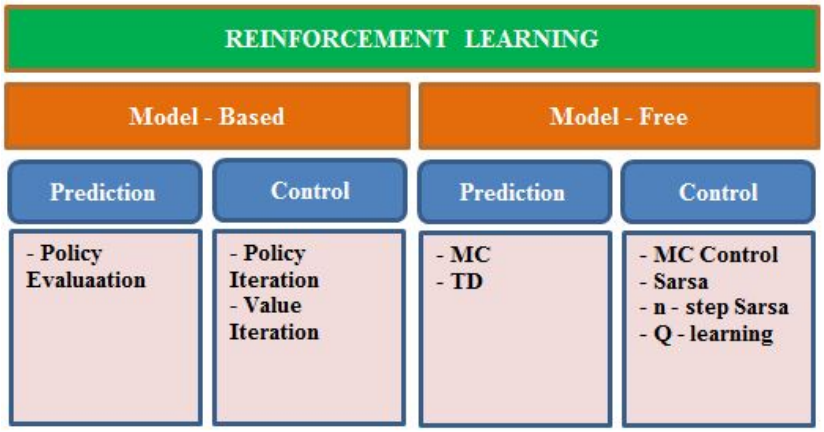

Figure 3.1. Reinforcement Learning Diagram.

3.1. Model Based Methods

3.1.1. Dynamic Programming

3.1.1.1. Policy Iteration. Policy iteration consists of two methods: policy evaluation and policy improvement. It starts with arbitrary policy and iteratively improves policy until convergence to an optimal policy π_* and optimal value function V_*. Policy evaluation is made by Bellman's Equation for all $s \in S$ and $\pi(a|s)$ is the probability of taking action a in state s under policy π. It can be seen as following:

$$v_{k+1}(s) = \sum_{a} \pi(a|s) \sum_{s',r} p(s', r|s, a) \left[r + \gamma v_k(s')\right] \tag{3.1}$$

Equation 3.1 shows that under the arbitrary policy π, v_π goes to infinity until specified termination criteria. New value is obtained at k+1 step from value of successor state s' at step k and expected reward r. Then, current value of state s is updated with new value which is taken from step k+1. In the policy evaluation, the environment's dynamics are assumed completely known. Algorithm of policy evaluation is seen in Figure 3.2 as following:

Input: π, policy to be evaluated
Input: (S, A, T, γ, R), MDP elements
initialise an array for new values, $V_{new}(s) = 0$, for all $s \in S$
initialise an array for old values, $V_{old}(s) = 0$, for all $s \in S$
initialise small positive number $\epsilon \sim 10^{-5}$
while True **do**
  $\Delta \Leftarrow 0$
  for each $s \in S$ **do**
    $V_{new} \Leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a) [r + \gamma V_{old}(s')]$
  end for
  $\Delta \Leftarrow \sum_{s \in S} |V_{OLD}(s) - V_{NEW}(s)|$
  if $\Delta < \epsilon$ **then**
    break
  end if
  $V_{old} \Leftarrow V_{new}$
end while
Output: $V_{new} \approx v_\pi$

Figure 3.2. Iterative Policy Evaluation Algorithm

We mentioned about how to state-value function v_π is computed iteratively under arbitrary and deterministic policy π. We should then ask ourselves how to improve policy in order to obtain maximum return for the model?

Using policy improvement method, we can decide whether current policy is good

to be used again or other policies which is better than current policy exist or not. Therefore, in order to search new policy, different actions which are not suggested by current policy should be taken in state s. Then, considering action-value function, it can be observed whether better policy is available or not. It means that if value of $q_\pi(s,a)$ is greater than $v_\pi(s)$, then this policy π which is under action a in state s should be followed. These concepts can be described as new greedy policy π' as following:

$$\begin{aligned}\pi'(s) &= \underset{a\in A}{\operatorname{argmax}}\, q_\pi(s,a) \\ &= \underset{a\in A}{\operatorname{argmax}} \sum_{s',r} p(s',r|s,a)\left[r+\gamma v_\pi(s')\right] \end{aligned} \tag{3.2}$$

According to value function v_π, current policy π is improved by new policy π' greedily by taking actions. Policy iteration algorithm works generally such that given arbitrary policy π, value function v under policy π is computed. Then, policy π is improved according to v_π in order to obtain better policy π'. This computation is repeated until convergence is reached for optimal π_* and v_*. Complete algorithm can be seen in Figure 3.3 as following:

3.1.1.2. Value Iteration. Value iteration is the iteration of value that policy evaluation is truncated without losing convergence property. Main idea is to use Bellman's Equation of V_* instead of V_π. [21]

Policy evaluation is truncated after one update operation of each state, and v_* is defined as combination of policy improvement and policy evaluation which is truncated without losing converge property for all of $s \in S$. Formulation of simple update rule can be defined as following:

$$v_{t+1}(s) = \max_{a\in A} \sum_{s',r} p(s',r|s,a)\left[r+\gamma v_t(s')\right] \tag{3.3}$$

Selection of an action a is performed greedily with respect to current value function

Input: (S, A, T, γ, R), MDP elements

1. Initialization

initialise arbitrarily two arrays for new values, V_{new} and π_{new}, for all $s \in S$

initialise to zero two arrays for old values, V_{old} and π_{old}, for all $s \in S$

initialise ϵ as a small positive number $\approx 10^{-4}$

2. Policy Evaluation can be seen in Figure 3.2.

3. Policy Improvement

Policy-stable $\Leftarrow$ TRUE

for each $s \in S$ **do**

$\pi_{new}(s) \Leftarrow \underset{a \in A}{\text{argmax}} \sum_{s',r} p(s', r|s, a) [r + \gamma V_{new}(s')]$

if $\pi_{old} \neq \pi_{new}$ **then**

Policy-stable $\Leftarrow$ FALSE

end if

$\pi_{old} \Leftarrow \pi_{new}$

end for

if policy-stable **then**

return $V_{new} \approx v_*, \pi_{new} \approx \pi_*$

else

go to 2

end if

Figure 3.3. Policy Iteration Algorithm

and it is used for update of value function in t+1 step. Update rule is similar to policy iteration but all actions are taken to be maximized value iteration function at each update until it converges the specified termination condition that is similar with policy iteration as well. Pseudo code of value iteration can be seen in Figure 3.4 as following:

Input: (S, A, T, γ, R), MDP elements
initialise replay memory D to capacity N;
initialise action-value function with random weights $V_{new}(s) = 0$, for all $s \in S$
initialise an array for old values, $V_{old}(s) = 0$, for all $s \in S$
initialise small positive number $\epsilon \sim 10^{-5}$
while True **do**
  $\Delta \Leftarrow 0$
  for each $s \in S$ **do**
    $V_{new} \Leftarrow \max_a \sum_{s',r} p(s', r|s, a) [r + \gamma V_{old}(s')]$
  end for
  $\Delta \Leftarrow \sum_{s \in S} |V_{OLD}(s) - V_{NEW}(s)|$
  if $\Delta < \epsilon$ **then**
    break
  end if
  $V_{old} \Leftarrow V_{new}$
end while
Output: a deterministic policy, $\pi \approx \pi_*$ such that:
$\pi(s) = \underset{a \in A}{\operatorname{argmax}} \sum_{s',r} p(s', r|s, a) [r + \gamma V_{new}(s')]$

Figure 3.4. Value Iteration Algorithm

3.1.1.3. Limitation of Dynamic Programming. In the dynamic programming methods such as policy iteration and value iteration, it is accepted that dynamics of model of Markov Decision Process including whole transition probabilities are known by agents. It means that in each action, agents know how to be responded by environment and dynamics of environment do not change over time. For this reason, dynamic programming

is called as “Model-based method”. However, It is noteworthy that it is not realistic and real world is not working like that. In real world which is non-stationary, probabilities of transition and reward are time-dependent. Accordingly, policy of agents change over time time as well. Instead of model-based method, “model-free” method which can overcome problems with stochastic transitions and rewards, without requiring adaptations is used in reinforcement learning. Model free algorithm is an algorithm which does not use the transition probability distribution and the reward function associated with the MDP. [21]

The most used model-free methods are Monte Carlo and Temporal-Difference methods which we will mention in next Sections.

3.2. Model Free Methods

3.2.1. Monte Carlo Method

Monte Carlo method is type of model free method that can be used due to limitation of dynamic programming if no prior knowledge of the environment is provided. Autonomous vehicle applications can be given as an example since vehicle (agent) has no prior information about environment and model dynamics such as set of transition probabilities of specified in MDP. Vehicle is trained by trial and error and gained an experience about environment so that it can find optimal policy to take correct decision such as preventing crash.

Monte Carlo Method consists of two problems similar with dynamic programming: prediction and control problems. Main idea of this method is to hold sampled action-value for each possible state-action pair in Q-table. Then, in order to estimate value-function $v_\pi(s)$ and $q_\pi(s, a)$, average of all returns that collected from past experiences which is called episode is recorded according to given policy π. The rest is similar with idea of policy iteration. Estimated value functions are updated according to state where agent visit and based on these value function estimation, policy is

updated. This process is repeated in each episode. Following formula defines idea of Monte Carlo Methods :

$$V_{t+1}(s) = (1-\alpha)V_t(s) + \alpha(G_t - V_t(s))\, for\, \forall\, s \in S \tag{3.4}$$

$$Q_{t+1}(s,a) = (1-\alpha)Q_t(s,a) + \alpha(G_t - Q_t(s,a))\, for\, \forall\, s,a \in S \times A \tag{3.5}$$

where t and G_t indicate the episode and expected return, respectively. Also, different returns are weighted by α.

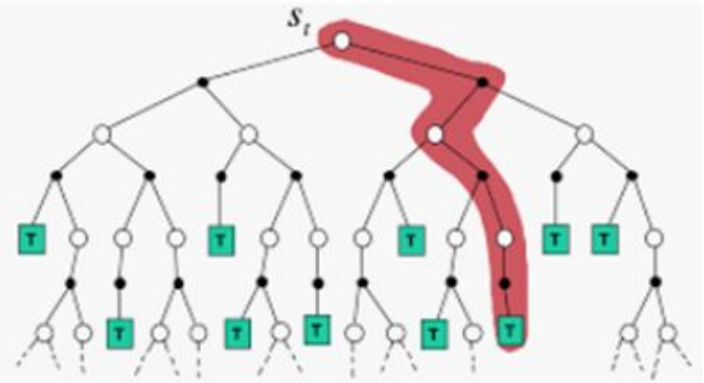

Figure 3.5. Monte Carlo Method

It can be seen from 3.4 and 3.5 that this method iterates in each episode. In the evaluation step for one episode, agent acts according to given policy π. Experience gained by agent is stored sequential form like $s_1, a_1, r_2, s_2, ..., s_t$ and Q table is updated according to them after ending each episode. Then, with the help of Equation 3.5, Q table is updated according to expected return G_t for each action-state pair (s_t, a_t) which is available as sequential form. After evaluation step is finished, policy π is updated according to recent Q-table in improvement step. "Greedy policy" which is famous and most conservative policy can be used to update policy π. These processes are iterated in each episode. Concept can be seen in Figure 3.6 as following.

Agents can encounter each state many times during every episode. Obviously,

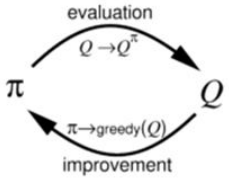

Figure 3.6. Evaluation and Improvement Concept

there are two alternatives to make prediction: First Visit Monte Carlo and Every Visit Monte Carlo. Average returns are considered only for first time s visited in an episode in the first visit Monte Carlo method whereas average returns are considered for every time s visited in an episode in the every visit one. In order to estimate expected value, one can simply add samples and division is performed by total number of samples.

$$\bar{V}_{\pi}(s) = \frac{1}{N} \sum_{i=1}^{N} G_{i,s} \tag{3.6}$$

where i episode index, s state

To better explain, it can be seen following example.

Suppose we have two states, i.e. state A and B, in specific environment. Let's consider 2 episodes are observed as following:

(i) $A + 2 \to A + 4 \to B - 3 \to A + 2 \to B - 4 \to$ terminate
(ii) $B - 3 \to A + 2 \to B - 1 \to$ terminate

It is noted that $A+2 \to A$ shows that transition occurs from state A to state A with 2 rewards. If you start counting when visiting first, then rewards which obtained for the state A in the first episode can be calculated as $2+4-3+2-4 = 1$. It can be calculated in the second episode as 2–1 = 1 .Thus, average of returns of two episodes gives $V(A)$, which is $(1+1)/2 = 1$. For state B, it can be evaluated as $-7/2$ for the first visit case. In the case of every state, rewards should be count in every appearance of A and B. If

you start counting when visiting every state, then rewards which obtained for the state A in the first episode can be calculated for the first appearance as $2+4-3+2-4=1$. For the second appearance: 4–3 + 2–4 = −1, for the third appearance: 2–4 = −2. In the second episode, it can be calculated for state A as simply $2-1=1$. Thus, average of returns of two episodes gives $V(A)$,which is $(1-1-2+1)/4=-1/4$. For state B, it can be evaluated as -3 for the every visit case. Then, value function table can be form for first visit MC and every visit MC as following Table 3.1

Table 3.1. First and Every Visit MC Example

First Visit Monte Carlo Value	Every Visit Monte Carlo Value
$V(A)=(1+1)/2=1$	$V(A)=(1-1-2+1)/4=-1/4$
$V(B)=(-5-2)/2=-7/2$	$V(B)=(-5-4-2-1)/4=-3$

Since completion of iteration needs to wait until end of episode in order to retrieve return in Monte Carlo method, this approach is very time consuming and lots of data should be trained since it can only learn from sequences that completed after end of episode. These can be taken into consideration as drawbacks of Monte Carlo Method.

3.2.2. Temporal Difference Method

Temporal difference (TD) is an important reinforcement learning method since contrary to dynamic programming problems; dynamics of environment is not needed to be known. As we mentioned earlier, it is necessary to know complete and accurate environment model as well as transition probabilities in dynamic programming (DP). In addition to this advantage, there are several advantages to use Temporal Difference methods. Firstly, these have computational simplicity. Secondly, when comparing with Monte-Carlo, these methods have lower variance relatively, and can work as online. TD methods which are combination of Monte-Carlo and dynamic programming approaches are methods that learn online after every step and they do not need to wait completion of sequences or episodes. Temporal difference learning consists of also two processes which are evaluation and control. Due to this reason, it is also considered as type of

initialise arbitrarily $Q(s,a)\,\forall s \in S, a \in A$;

initialise empty list $Re(s,a)$ of estimated returns;

initialise $\pi(s,a)$ arbitrarily;

for episode number $n < M_n$ Max episodes **do**

using π generate episode e and collect list of encountered pairs (s,a) and rewards;

for pair $(s,a) \in e$ **do**

$G \Leftarrow$ average return following the first visit (all visit) of (s,a)

Append G to $Re(s,a)$;

$Q(s,a) \Leftarrow$ average $(Re(s,a))$

end for

for s in episode e **do**

$a_* \Leftarrow \underset{a \in A(s)}{\operatorname{argmax}} Q(s,a)$

Update $\pi(s,a)$ with an exploration strategy $\forall a \in A(s)$

end for

end for

Figure 3.7. First Visit Monte Carlo Algorithm

generalized policy iteration methods. Iteration occurs as following: when agent is at time t and under policy π, it takes an action a_t in state s_t. Then transition is made by agent from state s_t to s_{t+1} with reward r_{t+1}. Value function of s_t is updated by temporal difference method is as following:

$$V_{(s_t)} \leftarrow V_{(s_t)} + \alpha \left[r_{t+1} + \gamma V_{(s_{t+1})} - V_{(s_t)} \right] \tag{3.7}$$

Temporal Difference error is shown in bracket in Equation 3.7. At time step t, TD-error of Q and V are called as δ_t^Q, δ_t^V and they can be explained as difference between current estimate value of s_t and new estimate for s_{t+1} .They are also corrected with reward that retrieved lastly. These can be seen in following formulas:

$$\delta_t^V(s_t) = R_t + V(s_{t+1}) - V(s_t) \tag{3.8}$$

$$\delta_t^Q(s,t) = R_t + Q(s_{t+1}) - V(s_t, a) \tag{3.9}$$

If these functions are repeatedly update with parameter $\alpha \in (0,1)$, errors are reduced. α is the step size parameter and it affects learning rate. Formula 3.8 and 3.9 can be improved with α as following:

$$\begin{aligned} V(s_t) &\leftarrow V(s_t)(1-\alpha) + \alpha(R_t + V(s_{t+1})) \\ &= V(s_t) + \alpha(R_t + V(s_{t+1}) - V(s_t)) \end{aligned} \tag{3.10}$$

$$\begin{aligned} Q(s_t, a_t) &\leftarrow Q(s_t, a_t)(1-\alpha) + \alpha(R_t + Q(s_{t+1}, a_{t+1})) \\ &= Q(s_t, a_t) + \alpha(R_t + Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)) \end{aligned} \tag{3.11}$$

When α is equal to one, most recent information is learnt by agent and when α is reduced gradually over time, agent learns gradually as well. Therefore, function reaches convergence.

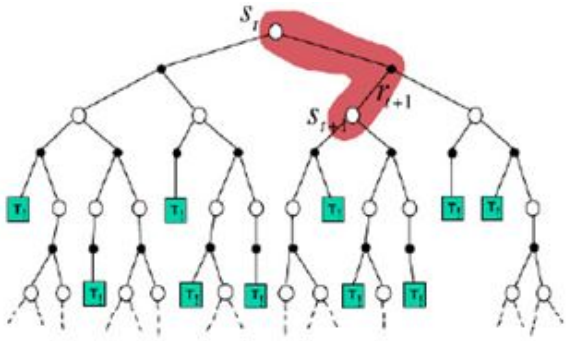

Figure 3.8. Temporal Difference Method

Algorithm for the TD method can be explained in Figure 3.9 as following:

As we mentioned on earlier Chapter, on-policy and off-policy are two types of approaches in TD methods for learning policy q_π. While on policy TD method is called as SARSA, off-policy learning one is called as Q-learning. We will cover these methods on next Sections.

3.2.2.1. SARSA Algorithm (On-Policy TD Control). Sarsa is type of on-policy Temporal Difference Method and its name comes from a tuple of transition experience as $(s_t; a_t; r_{t+1}; s_{t+1}, a_{t+1})$ State-Action-Reward-State-Action for each update. As we mentioned earlier, $q_\pi(s, a)$ is estimated according policy π and it changes greedily based on q_π in on-policy method. By means of transitions from a state-action pair to a state-action pair, SARSA method learns action- values.

Pseudo code of Sarsa algorithm can be seen in Figure 3.10 as following:

It is noteworthy that convergence with probability 1 is guaranteed in SARSA method if following requirements meet:

input policy π to be evaluated

initialise $V(s)$ arbitrarily (e.g. $V(s) = 0, \forall s \in S$)

repeat

  for each episode

  initialise s

  repeat

    for each step in episode

    $a \Leftarrow$ action given by π for s

    take action a, observe $r, s^{'}$

    $V(s) \leftarrow V(s) + \alpha\left[r + \gamma V(s') - V(s)\right]$

    $s \Leftarrow s'$

  until s is terminal

until

Figure 3.9. Tabular Temporal Difference Algorithm

- All the state-action pairs are visited an infinitely often.
- Following two conditions are satisfied by succession of parameter α:
  (i) $\lim_{N\to\infty} \sum_{N}^{t=1} \alpha_t^2 < \infty$
  (ii) $\lim_{N\to\infty} \sum_{N}^{t=1} \alpha_t = \infty$

3.2.2.2. Q-Learning Algorithm (Off-Policy TD Control). Q-learning method is also called as max-SARSA method and this method is the type of off-policy Temporal Difference Method. When optimal action-value is learnt by agent, policy is not being followed. It can be defined as following formula:

$$\begin{aligned} Q(s_t, a_t) &\leftarrow Q(s_t, a_t) + \alpha(R_t + \gamma \max_{a_{t+1}\in A(s_t)} Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)) \\ &= Q(s_t, a_t)(1-\alpha) + \alpha(R_t + \gamma \max_{a_{t+1}\in A(s_t)} Q(s_{t+1}, a_{t+1})) \end{aligned} \tag{3.12}$$

initialise $Q(s_{ter},) = 0 \, \forall s_{ter} \in S_T$ terminal states;

initialise arbitrarily $Q(s,a) \, \forall s \in S, a \in A$

initialise first state to the starting state $s = s_0$;

for episode number $n < M_n$ Max episodes **do**

choose a_t from $A(s)$ using a policy derived from Q (with an exploration strategy,e.g. ϵ greedy);

for step $t < M_s$ (max steps or infinity) **do**

take action a;

Observe reward $r = r_t$ and $s^{'} = s_{t+1}$;

Choose $a^{'} = a_{t+1}$ from $A(s_{t+1})$ using policy derived from Q (with an exploration strategy,e.g. ϵ greedy);

Update $Q(s,a)$:

$Q(s,a) \Leftarrow Q(s,a) + \alpha \left[r + \gamma Q(s^{'}, a^{'}) - Q(s,a) \right]$

$s \Leftarrow s^{'}$;

$a \Leftarrow a^{'}$;

end for

end for

Figure 3.10. SARSA Algorithm

It can be seen from Equation 3.12 that this method is off-policy method since q_* is approximated directly by Q value function. Pseudo code of Q-learning can be seen in Figure 3.11 as following:

initialise $Q(s_{ter},) = 0 \, \forall s_{ter} \in S_T$ terminal states;
initialise arbitrarily $Q(s,a) \, \forall s \in S, a \in A$
initialise first state to the starting state $s = s_0$;
for episode number $n < M_n$ Max episodes **do**
  for step $t < M_s$ (max steps or infinity) **do**
    Choose action $a = a_t$ from$A(s_t)$ using policy derived from Q(with an exploration strategy);
    Take action a;
    Observe reward $r = r_t$ and $s^{'} = s_{t+1}$;
    Update $Q(s,a)$:
    $Q(s,a) \Leftarrow Q(s,a) + \alpha \left[r + \gamma \max_{a'} Q(s^{'}, a^{'}) - Q(s,a) \right]$
    $s \Leftarrow s^{'}$;
  end for
end for

Figure 3.11. Q-learning Algorithm

4. DECISION MAKING

Based on theory and methods on current state of art decision making approaches which is shown in chapter 2 and 3, Markov Decision Process is obviously most suitable approach in autonomous driving problem and it can be implemented to traffic many traffic scenarios in practice. In this chapter, MDP which is formulated for autonomous decision making problem and proposed algorithms will be introduced. Following this, calculation of implemented algorithms will be shown in detail.

4.1. MDP Formulation of the Autonomous Driving Problem

Formulation of the problem is modelled by Markov Decision Process. The aim is to find optimal policy which maximizes sum of the expected discounted rewards. It is discounted since reward we obtain next steps may not as important as one obtained current step. It is important to note that while reward is the value received from one step, return is the value which agent obtains from one episode. Difference between them can be seen from following Figure 4.1.

As mentioned at previous Chapter, discrete state space is chosen for modelling of autonomous vehicle problem. States for the ego vehicle and other traffic objects are position (x, y), velocity (v_x, v_y) and acceleration (a_x, a_y). Other vehicle states depend on the uncertain behavior, it means that they perform lane changes randomly. States of ego car and other cars can be seen in 4.1 and 4.2

$$s = (s_{ego}, s_1, ..., s_n) \tag{4.1}$$

$$\begin{aligned} s_{ego} &= (x, y, v_x) \\ s_i &= (x, y) \end{aligned} \tag{4.2}$$

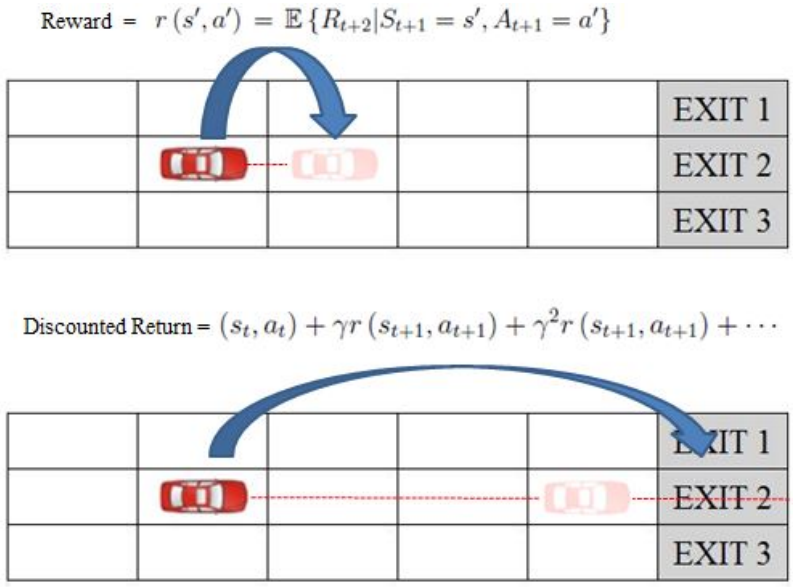

Figure 4.1. Difference Between Reward and Episode

where x and y are the position states of ego and other cars seperately based on environment's coordinates described in Figure 5.1 and v_x is the velocity state of ego car in longitudinal direction.

Action set is also implemented as discrete and it should be chosen carefully since when adding more action in action set, it causes growing of problem exponentially and it brings too much computational cost to find optimal policy. Action set in this thesis can be seen in 4.3 as follows:

$$a = (turn\ left, no\ change, turn\ right, brake, stay\ constant, accelerate) \tag{4.3}$$

4.1.1. Reward Function

Main goal of ego car is to reach specified destination as quick as possible without any crash with other cars. In addition to safety criteria such as safe arrival, ego car should continue its journey comfortably as a result of its decisions. Reward values

are assigned based on mentioned criteria so that optimal actions can be chosen by ego vehicle. Most important goal of decision making of autonomous car is safety. Therefore, negative rewards are assigned when collision with other cars and situation that ego vehicle is at out of lane occur. The second objective of the ego car is to complete the road without excess lanes and speed changes since constant velocity action instead of direct switching of braking and acceleration actions is considered as more conformable. Also, excessive lane change on highway affects comfortable ride negatively. Illustration of lane change rewards can be seen in 4.2 as follows:

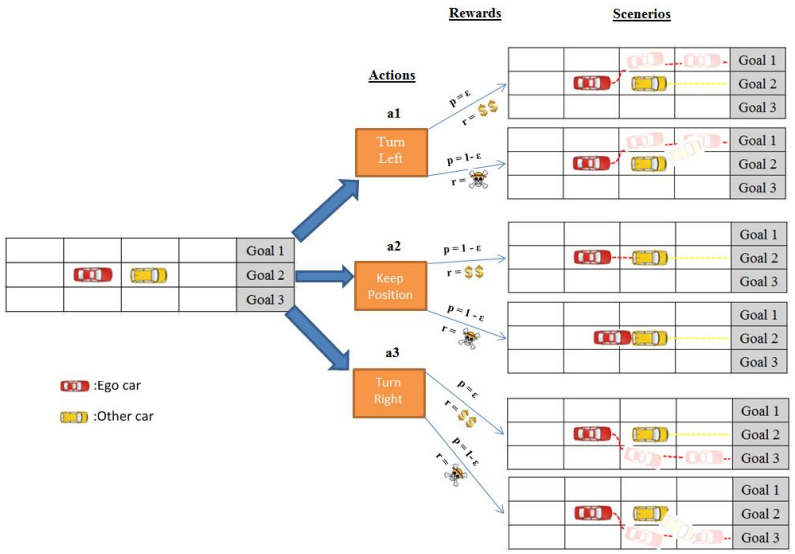

Figure 4.2. Rewards of Lane Changes

4.1.2. Algorithm of Reward function for Autonomous Driving

All of the reward values based on reward function in Section 4.1.1 and termination flags indicated in Table 5.1 are implemented according to reward algorithm which can be seen in Figure 4.3 as following:

4.1.3. Terminal State

There are three terminal criteria over entire horizon. The first terminal criterion is reached destination criteria. It means that, when the ego car reaches the one of the

function: Reward Function(state, action)

$reward = 0$

if out of lane **then**

  $reward+ = r_{out\,of\,lane}$

  termination flag = $TRUE$

else

  if previous position of ego y != current position of ego y **then**

    $reward+ = r_{excessive\,lane\,change}$

  end if

  if current speed of ego ¡= 0 **then**

    $reward+ = r_{zero\,or\,negative\,speed}$;

    termination flag = $TRUE$

  end if

  if previous speed of ego != current speed of ego **then**

    $reward+ = r_{speed\,boost}$

  end if

  if collision occurs **then**

    $reward+ = r_{collision}$

    termination flag = $TRUE$

  else

    if goal is reached **then**

      $reward+ = r_{goal}$

      termination flag = $TRUE$

    end if

  end if

end if

Figure 4.3. Reward Function Algorithm for Autonomous Driving

exit states, episode is over and ego vehicle stops to explore new states and gain reward. Second one is collision or violation of minimum distance criteria between ego car and other cars. When collision or violation of minimum distance between them occurs, it is detected by specified terminal flag in algorithm and episode is over. Last one is violation of limitation of lane. It must be ensured that ego car should not exceed limits of specified lane in highway. Representation of three terminal criteria can be seen in Figure 4.4 as following.

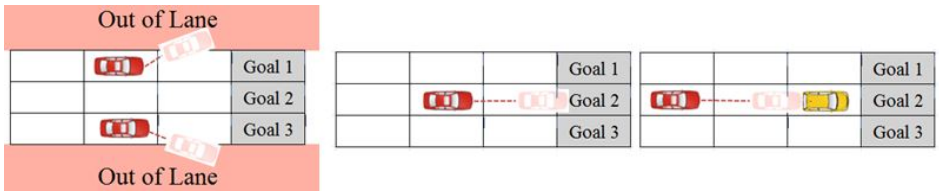

Figure 4.4. Representation of Three Terminal Criteria

4.1.4. Collision Detection

Collision detection is one of the terminal state condition and important phenomena for safety of decision making of autonomous car. Specification of mathematical model is critical and it must ensure that collision would never occur between ego car and other cars. Distance check in every state is simplest and reliable method to avoid collision. In this thesis, the Responsibility-Sensitive Safety (RSS) model deployed in Intel/Mobileye's test fleet of fully automated cars is used for safety assurance [34].

Our simulation setup consists of leader - follower cars. As generally known, follower car is responsible to arrange its acceleration to ensure safe longitudinal distance. Main principle for safety is : " ego vehicle should be able to keep enough distance with followed car so that it can find a time to stop if followed car brakes suddenly". Therefore, the RSS Following Distance Equation [34] for leader – following car scenarios can be defined as follows:

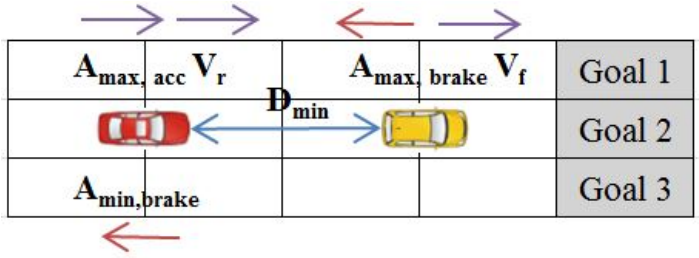

Figure 4.5. RSS - Following Distance Illustration

$$d'_{min} = MAX\left\{0, \left(v_r\rho + \frac{1}{2}a_{max,accel}\rho^2 + \frac{v_r + \rho a^2_{max,accel}}{2a_{min,brake}} - \frac{v_f^2}{2a_{max,brake}}\right)\right\} \tag{4.4}$$

where:

- d'_{min} is minimum following distance between two vehicles according to RSS
- v_f is longitudinal velocity of lead vehicle
- v_r is is longitudinal velocity of follower vehicle
- ρ is response time delay before follower vehicle start to brake
- $a_{max,brake}$ is maximum brake capability of lead vehicle
- $a_{max,accel}$ is maximum acceleration of follower vehicle
- $a_{min,brake}$ is minimum brake capability of follower vehicle

4.2. Solver Algorithms

In this thesis, we will use model free methods to find optimal policy using value - based approach since transition probability $P\left(s'|s_t, a_t\right) \forall\, s' \in S$ is not known for every states due to uncertainty in decision making of self-driving problems. In order to find V_π, repeating iteration for all $s \in S$ is performed in Equation 4.5 as following:

$$V_\pi\left(s\right) = R\left(s\right) + \gamma \sum_{s \in S} P(s'|s, \pi(s))V_\pi(s') \tag{4.5}$$

Let's consider we are in some state s_t , receiving reward r_t, and take an action $a_t = \pi(s_t)$ to shift state s_{t+1} according to MDP. It can be formulated as following:

$$V\left(s_t\right) = r_t + \gamma \sum_{s' \in S} P(s_{t+1}|s_t, a_t) V_\pi(s_{t+1}) \tag{4.6}$$

Due to the fact that we do not know transition probability $P\left(s^{'}|s_t, a_t\right) \forall\, s^{'} \in S$, it can not be updated to find $V(s_t)$. However, s_{t+1} is a sample from distribution of $P\left(s^{'}|s_t, a_t\right)$. Therefore, it could be updated as

$$V_\pi\left(s_t\right) = r_t + \gamma V_\pi(s_{t+1}) \tag{4.7}$$

At this point, in order to perform "smooth" update, learning rate (α) also called "step size" is chosen between 0 and 1. It can be described as how fast we are approaching the goal. Zero learning rate means that value is not updated, and agent will learn anything, whereas, one learning rate will cause to be learn most recent information by agent, and learning can occur quickly.

4.2.1. Implemented Algorithms for Autonomous Vehicle Problem

In order to explain value based method, following example can be given for autonomous vehicle problem simulated in this thesis. Considering that ago car is at 2 states at behind of goal at time t which is seen in Figure 4.6 as follows:

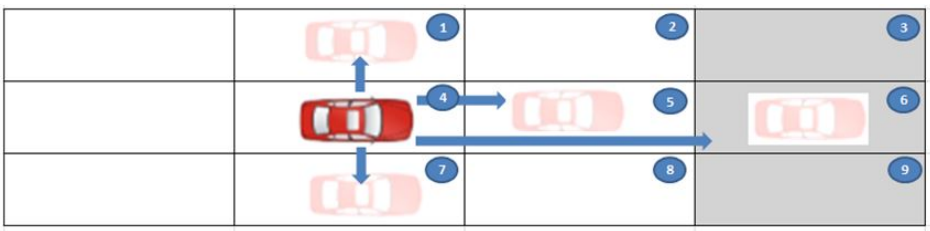

Figure 4.6. States of Ego Car at Behind of Goal

Q-value is the function of 2 terms: state and action. It represents quality of

taking this action being in the state. Mathematically, Q-value is the expectation sum of rewards you have if it takes action and follow the policy π. Q-table (or V-table) which is the representation of quality of policy in value based approach can be constructed in Table 4.1 as follows: Remember that our environment is that ego car is

Table 4.1. Tabular Representation of State-Action Pairs

Time	Actions						State-Features		
id	turn_left	no_change	turn_right	slow_down	stay_constant	speed_up	Ego_x	Ego_y	Ego_Vel
0	12	5	12	2	10	19	16	0	1
1	10	4	16	23	12	15	17	1	1
2	22	3	12	5	13	22	18	0	2
3	5	19	21	10	5	50	19	2	1
4	8	17	23	16	14	3	20	1	1
.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.

driving on straight road (highway) and it defines different states with position x and y and velocity. For each state, it should be allocated a value of Q for each of actions. At this point, it can be asked : how do we fill Q-table or how do we determine these Q values after randomly initialization? Lets consider we are in state 4 in Table 4.1, and we can find the value of this state according to transition reward as a result of action that agent takes using Bellman's Equation in Equation 4.7. According to this Equation, the value of this state is equal to sum of immediate rewards and discounted value of next state according to action that agent takes. According to this Equation, if agent is at one step behind of termination state, value of termination state is equal to zero since there is no next state which has any value. Thus, value of state where agent is at behind of termination state is equal to immediate reward (goal reward). At this point, the aim is to find optimal policy as we mentioned previously. It is performed by agent according to exploration/exploitation strategy using epsilon greedy method. Algorithms start as random policy with random actions, and they build optimal policy with small randomness. According to epsilon greedy method, epsilon value is chosen as float between 0 and 1 and action is chosen according to following:

ϵ Greedy policy:

- Probability $= 1 - \epsilon$
  (i) action $= \pi(state) = argmax_a Q_\pi(state, action)$
- Probability $= \epsilon$
  (i) action $= \pi(state) = random.choice(possible\, actions)$

It is really important that epsilon(ϵ) should start close to one it goes to close to zero. Decrease of this parameter is performed by decay parameter. Epsilon parameter is decayed by multiplying at each episode based on decay parameter as follows:

- $\epsilon = \epsilon_{init}$
  (i) $\epsilon = max(\epsilon_{end}, decay_\epsilon \times \epsilon)$

Decay parameter is really important in reinforcement learning and it has huge impact of performance of learning process of agent.

After filling Q-table regarding value based approach, update of these value based on optimal policy according to interaction with environment is another important concept in RL. Now, we have $Q_\pi(s, a)$ values which are initialized randomly. The value is known when agent is at state s and it takes action a. It is called as Q–estimate. Based on Bellman's Equation 4.7, new value can be evaluated and it is called as Q–target. Error between Q-estimate and Q–target can be evaluated as follows :

- Error = Q-target - Q-estimate

- Update
  (i) Q-estimate = Q-estimate + $\alpha\times$Error
  (ii) $\alpha =$ learning rate

In order to update Q-estimate, learning rate is used. At each experience for s and $s^{'}$, it is applied and updated corresponding cell in Q-table. Algorithm can be seen in Figure 4.7 as follows:

repeat

Take action A_t and observe R_{t+1}, S_{t+1}

Chose action A_{t+1} using policy derived from Q(e.g ϵ Greedy)

$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t))$

$t \Leftarrow t + 1$

until S_t is terminal

Figure 4.7. Update Algorithm of Q-values

Calculation of Q value of $(s^{'}, a^{'})$ are different according to selected algorithms. In the example given above, 6 actions are available in each state. Thus, there are 6 possible potential actions to be selected for next state as well as corresponding to 6 Q values for $(s^{'}$,$a^{'})$. If average of them is taken and used for Q values of next state, it is called "Expected Sarsa". If maximum Q value is selected among 6 possible Q values, and used for Q values of next state, it is called "Sarsa - Max". It is also called as "Q-learning" which is quite popular method in Reinforcement Learning. In both cases, current policy is not implemented. That's why they are the off-policy method. The third policy which is implemented in this thesis is "Sarsa" and it reuses same policy for estimate target. That's why it is on-policy method.

5. SIMULATION RESULTS

This chapter introduces implemented simulation setup including traffic model and environment setup, chosen parameter and evaluation criteria which is determined specifically for autonomous driving problem. Then, chosen test scenarios are introduced with corresponding results supported by various graphs.

5.1. Simulation Setup

In this thesis, Python which offer many options for development of GUI (Graphical User Interface) is used as software in order to test and verification of algorithms and implementation of environment. Among these GUI options, tkinter is most well-known, fastest and easiest method and it is used with standard Python interface to solve autonomous vehicle problems. Desired traffic environments including traffic objects such as lane, cars are built and they are implemented Python software as an input. Decision making framework is established in every 1 second and corresponding values which algorithms create are stored in .pkl format in training phase so that they are used to test specified scenarios which will be mentioned in next Subsections.

5.1.1. Trafic Model

Traffic model is important for autonomous vehicle test and development since agents have an interaction with environment. In this thesis, simple and realistic traffic environment is designed using Python Tkinter with canvas widgets. Portion of highway which includes number of straight lanes, exits and vehicles is implemented. There are many types of traffic models which depend on level of details. According to [35], low, medium and high details are classified as microscopic, mesoscopic and macroscopic models, respectively. While microscopic models emphasize on particular elements such as vehicle and pedestrian, macroscopic models focus on combination of microscopic models such as traffic flow and density, and these models are more complex than mi-

croscopic models. Mesoscopic models have medium detail and represent small groups of traffic entities described by microscopic models (low detail) and their interactions. In this thesis, our goal is to focus on behavior of autonomous vehicles in traffic environment. Thus, we use microscopic simulation model since this model describes space-time behavior of the entities of system and its interaction as individual level. Time scale classification is also important concept for traffic models. There are two types of time scales which are discrete and continuous. In the discrete model, environment state changes at discrete time instants take place discontinuously over time. However, in continuous models, state of traffic system changes continuously over time in response to continuous stimuli. In addition to time scale, variables such as velocity, position etc. can be classified as discrete, continuous and mixed. In this thesis, although continuous model is more complex and natural, we decided to use fully discrete model since it is simple.

5.1.2. Traffic Model Description

Environment is chosen as highway. Structure consists of lanes, exits, ego car, other cars etc. Two dimensional environment is built for highway and lanes are positioned as following:

Figure 5.1. Illustration of 2 Dimensional Environment

It can be seen from Figure 5.1 that highway environment has group of lanes and these lanes have group of cells. Ego car and others car positions are defined by lane and corresponding cells of defined lane. Since cars progress in two dimensional space, it can be easily indexed position of these using x and y coordinates. Figure 5.1 shows that

lanes are indexed as y coordinate such as 0,1,2 for three lanes, and cells are indexed as x coordinate as 0,1,2,. . . ,20,21. Reference of coordinate system 0,0 point is chosen as top left-hand corner and cell numbers increase from left to right. Exit lanes are located at the right – end and left - end of the environment. When the ego car and other cars reach the exit location, it means that they complete the highway as it happens in real life. In reinforcement learning, it is called as episode. It is also important to note that all cells have same distance due to discrete environment.

5.2. Environment Setup

In this thesis, three different types of environment with three different traffic densities are implemented. First environment consist of 2 lanes and 20 cells . Ego car and other cars have same direction. Structure can be seen in Figure 5.2 as following:

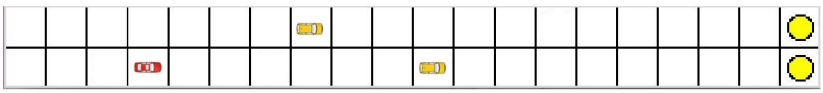

Figure 5.2. Illustration of 3 Vehicles Environment in Highway

Second environment is built with 3 lanes and 20 cells with different traffic densities. Ego car and other cars have same direction. Traffic density is chosen as 0.4 for test scenarios. Structure can be seen in Figure 5.3 as following:

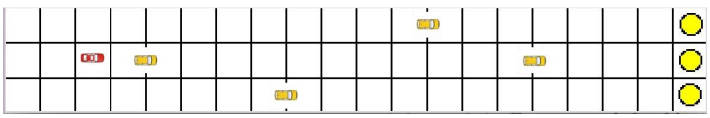

Figure 5.3. Illustration of 5 Vehicles Environment in Highway

Finally, third environment is implemented as 3 lanes and 40 cells with 0.9 traffic density. Ego car and other 9 cars have same direction. Structure can be seen in Figure 5.4 as following:

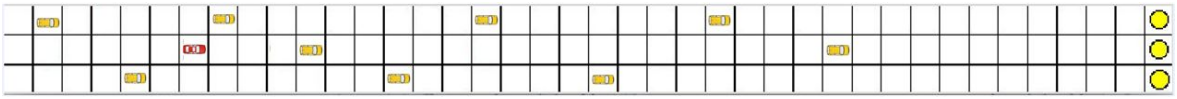

Figure 5.4. Illustration of 10 Vehicles Environment in Highway

5.2.1. Kinematic of Cars

Ego car's velocity and acceleration are discrete values. It can only take integer values and change at every time step t. Transition model among velocity, acceleration and position can be described according to Newton's laws of motion as kinematic Equations in 5.1 and 5.2

$$x_{t+1} = x_t + v_t * \Delta t \tag{5.1}$$

$$v_{t+1} = v_t + a_t * \Delta t \tag{5.2}$$

According to Equations 5.1 and 5.2, forward condition of car in discrete space can be illustrated in Figure 5.5 as following:

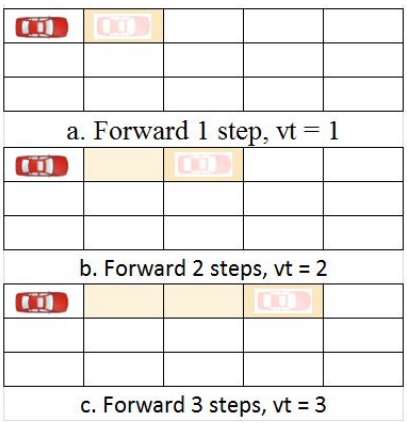

Figure 5.5. Illustration of Forward Condition of Cars

5.3. Parameter Configuration

In order to better understand results of tested scenarios, parameters are used to build decision making framework are listed and it is divided into categories as following Table 5.1.

Table 5.1. Parameters of Autonomous Vehicle Problem

Category	Parameter	Value
General	Time Step (delta T)	1
	Total Time Horizon (T)	20 and 40
	Discount Factor (Gamma)	0,9
Actions	Accelarate	$1 m/s^2$
	Constant	$0 m/s^2$
	Brake	$-1 m/s^2$
	Turn Left	Shift to Left Lane
	Keep Position	Keep Position in Current Lane
	Turn Right	Shift to Right Lane
Rewards	Collision	-20
	Excessive Lane Change	-5
	Zero or Negative Speed	-15
	Goal Reached	+50
	Out of Lane	-20
	Speed Boost	3*(Current Speed - Initial Speed)
Termination	Out of Lane	Termination Flag = True
	Collision	Termination Flag = True
	Goal reached	Termination Flag = True

5.3.1. Tuned Parameters in Training Phase

In this thesis, all implemented algorithms are trained as 100000 episodes and performances of algorithms are reported in Test Result Section according to specified criteria. In order to reach good performance, parameters of training such as learning rate, epsilon greedy start and end values and decay rate are tuned recurrently. In the training phase, the parameters tuned for each algorithm are listed below Table 5.2

In addition to tuned parameter listed in Table 5.2, rewards values specified in Table 5.1 are tuned many times to achieve best decision making performance. As a

Table 5.2. Tuned Parameters in Training Phase

Tuned Parameters	Tuned Parameter Values
Learning Rate	0.003, 0.01, 0.03, 0.1, 0.3, 1
Gamma (Discount Factor)	0.1, 0.3, 0.5, 0.7, 0.9, 0.95, 0.99, 1
Epsilon Decay	0.954, 0.984, 0.995, 0.998, 0.9995, 0.9999
Epsilon End Value	0.005, 0.01, 0.05, 0.1, 0.2

result of test and training phase, parameters for training of decision making algorithms are determined as follows:

- **Learning Rate :** 0,003
- **Gamma (Discount Factor) :** 0,9
- **Epsilon Decay :** 0,998
- **Epsilon Start Value :** 1,0
- **Epsilon End Value :** 0,01

5.4. Evaluation Criteria

As mentioned previous Chapters, most important goal for decision making of autonomous vehicle is to ensure safe driving. In addition to safe driving, comfort and good efficiency should be good enough for passenger inside of autonomous car. At this point, we can ask this question: how could decision making of autonomous car be good enough to ensure both safe driving and comfort? Unfortunately, there are no standard evaluation criteria or test to decide efficiency of decision making in literature yet. However, following criteria can be defined to test performance of decision making of autonomous car quantitatively:

- **Number of action changes over entire simulation period :** In order ensure comfort of passengers by means of smooth drive, number of action change should be as low as possible over entire episode.
- **Minimum distance between ego car and other cars :** Minimum distance between ego car and other cars can be measured in test scenarios. In order to

keep safety margin, distance between them must be larger than RSS, which is defined as formula 4.4. As shown in formula, distance is directly proportional to velocity of vehicles.

5.5. Test Results

Aim of this thesis is to test and compare different decision making framework in order to achieve an adequate performance and level of safety even when highway scenarios simulated in software according to specified traffic model has a large amount of uncertainty. Obviously, it is impossible to show all of tested scenes and scenarios here, chosen ones are displayed to provide reader with result of relatively wide range of feasible scenarios. In the following Subsections, implemented scenario will be described with figures, graphs as action-reward and position profile of each car which is available in traffic environment. In the provided graph, ego vehicle is specified as marked line to be seen easily. In the first result, position of the ego car and other cars can be seen as x and y coordinates. Following graph shows actions of ego vehicle and corresponding reward values in each time step. Then, number of action change over entire simulation period and minimum distance between ego car and other cars are reported in each test scenario. Finally, performance of each implemented algorithms are graphed as number of non-collision episode out of 100 tested episodes with same initial values.

5.5.1. Two lanes with Same Direction Scenario

This is the simplest test scenario where three cars (one of them is ego car) drives towards to same way. An illustration of the simulation environment is shown in Figure 5.6.

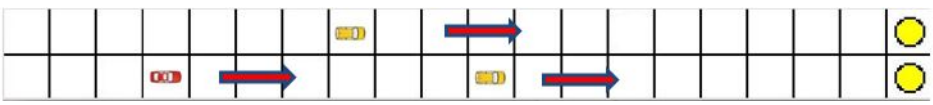

Figure 5.6. Illustration of Two Lanes with Same Direction Scenario

In the first scenario, three traffic objects including ego car start from same directions. Two traffic objects are called as “other cars” and they have relatively lower velocity than ego car in initial condition. They continue to follow the road and perform lane change randomly. At the point when other traffic objects are performing a lane change or they have lower velocity than ego car, it can be seen that ego car takes required action to arrange its velocity according to their positions. Also, it can be observed that ego car is able to perform lane change action in order to avoid potential collision and it can overtake other cars safely. Training and test output are given in following Subsections based on three different implemented algorithms.

5.5.1.1. Training and Test Result of Implemented Algorithm 1. Training is performed according to Q-learn algorithms and number of counted steps of ego car and total rewards taken by it in each episodes can be seen as following graph 5.7 and 5.8, respectively.

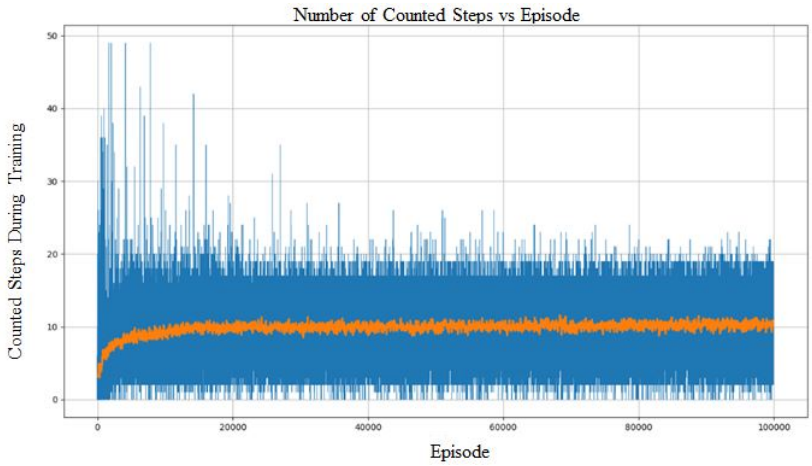

Figure 5.7. Number of Counted Steps of Ego Car in Each Episode for Algorithm 1

It can be seen from tables that ego car learns quickly to ensure safe driving. Number of step size in each episode increases and converges to specified value due to the fact that ego car is able to reach goal position in each episode after converging occurs. Maximum return is obtained as 83 among 100000 episodes in training phase.

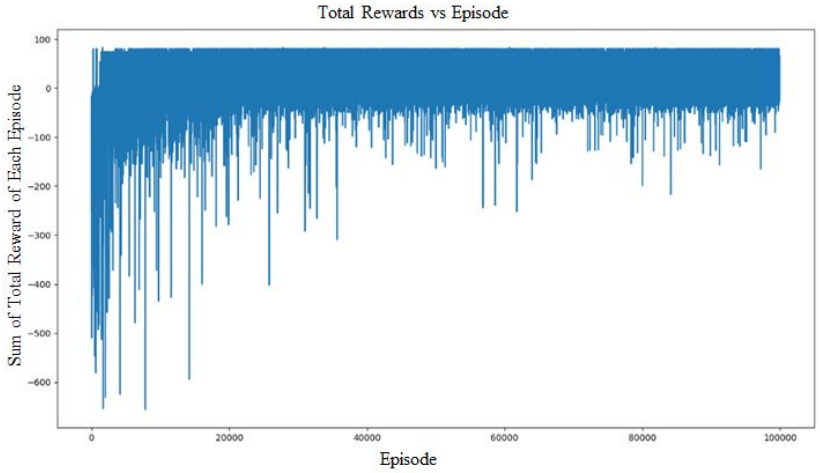

Figure 5.8. Sum of Rewards in Each Episode for Algorithm 1

Many test scenarios are performed and one of them is reported in Table 5.3.

Table 5.3. Information of First Tested Scenario for Algorithm 1

Time	Ego_x	Ego_y	Car1_x	Car1_y	Car2_x	Car2_y	Ego_Vel	Actions	Actions	Reward	Termination
0	0	1	3	1	8	1	1	'no_change'	1	0	False
1	1	1	3,5	1	8,5	1	1	'stay_constant'	4	0	False
2	2	1	4	1	9	1	1	'stay_constant'	4	0	False
3	3	1	4,5	1	9,5	1	1	'no_change'	1	0	False
4	4	1	5	1	10	1	1	'turn_left'	0	-5	False
5	4	0	5,5	1	10,5	0	1	'stay_constant'	4	0	False
6	5	0	6	1	11	0	1	'speed_up'	5	3	False
7	7	0	6,5	1	11,5	0	2	'slow_down'	3	0	False
8	8	0	7	0	12	0	1	'no_change'	1	0	False
9	9	0	7,5	0	12,5	0	1	'speed_up'	5	3	False
10	11	0	8	0	13	0	2	'slow_down'	3	0	False
11	12	0	8,5	1	13,5	0	1	'stay_constant'	4	0	False
12	13	0	9	1	14	0	1	'turn_right'	2	-5	False
13	13	1	9,5	1	14,5	1	1	'stay_constant'	4	0	False
14	14	1	10	0	15	1	1	'turn_left'	0	-5	False
15	14	0	10,5	0	15,5	1	1	'stay_constant'	4	0	False
16	15	0	11	0	16	1	1	'speed_up'	5	3	False
17	17	0	11,5	0	16,5	1	2	'no_change'	1	50	True
18	19	0	12	0	17	1	2				

Positions (x, y) of the ego and other cars are represented in graph 5.9 and it can be clearly seen that no crash occurs in tested scenario. In the time between 7s and 8s, while ego car and car1 are at the same x position, they have different y position since ego car overtakes the car1.

Reward change with corresponding action during 1 tested episode can be seen as

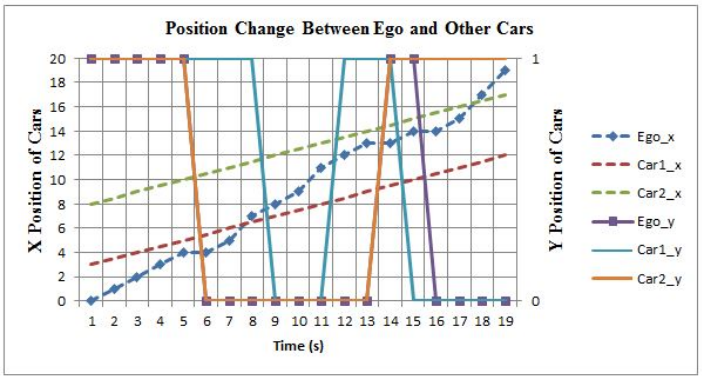

Figure 5.9. Position Change of Ego and Other Cars in x and y Coordinates

following graph 5.10 according to implemented algorithm. Return is obtained as 44 in selected test scenario.

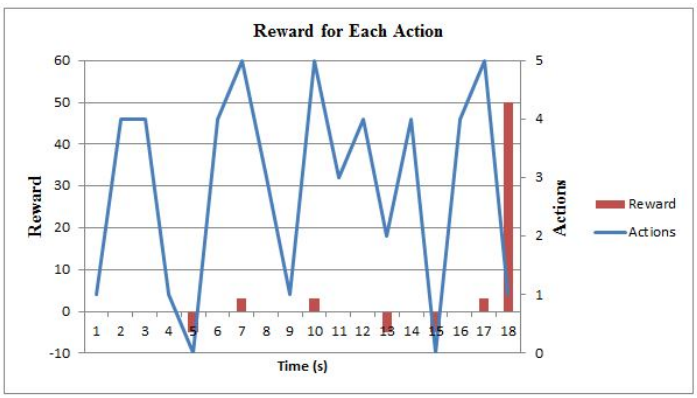

Figure 5.10. Actions vs Corresponding Rewards for First Tested Scenario

5.5.1.2. Training and Test Result of Implemented Algorithm 2. SARSA is the second algorithm trained for the "two lanes with same direction" environment as 100000 episode. Number of counted steps of ego car and total rewards taken by it in each episode can be seen as following graphs 5.11 and 5.12, respectively. Maximum return is obtained as 95 among trained episodes.

Many test scenarios are performed and one of them is reported as following Table 5.4

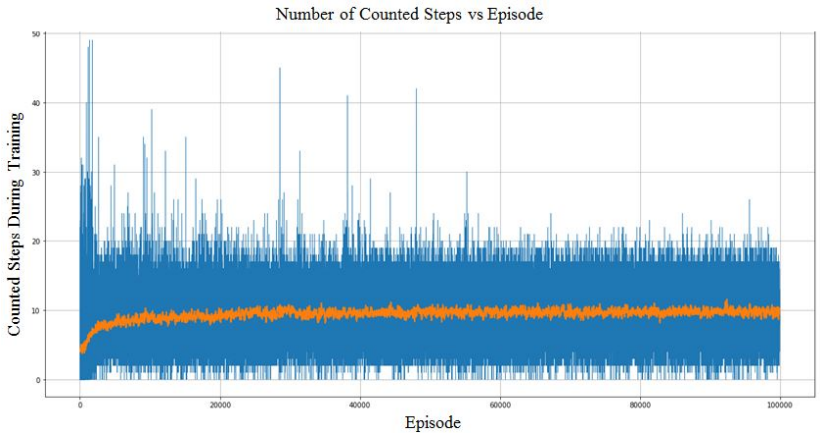

Figure 5.11. Number of Counted Steps of Ego Car in Each Episode for Algorithm 2

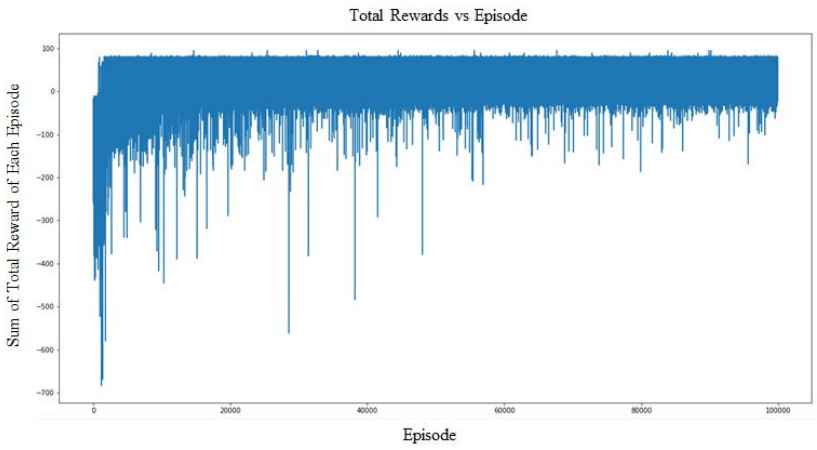

Figure 5.12. Sum of Rewards in Each Episode For Algorithm 2

Table 5.4. Information of First Tested Scenario for Algorithm 2

Time	Ego_x	Ego_y	Car1_x	Car1_y	Car2_x	Car2_y	Ego_Vel	Actions	Actions	Reward	Termination
0	0	1	3	1	8	1	1	no_change	1	0	False
1	1	1	3,5	1	8,5	1	1	no_change	1	0	False
2	2	1	4	1	9	1	1	no_change	1	0	False
3	3	1	4,5	0	9,5	1	1	speed_up	5	3	False
4	5	1	5	0	10	1	2	slow_down	3	0	False
5	6	1	5,5	0	10,5	1	1	no_change	1	0	False
6	7	1	6	0	11	1	1	speed_up	5	3	False
7	9	1	6,5	1	11,5	1	2	slow_down	3	0	False
8	10	1	7	1	12	1	1	no_change	1	0	False
9	11	1	7,5	1	12,5	1	1	no_change	1	0	False
10	12	1	8	1	13	1	1	turn_left	0	-5	False
11	12	0	8,5	1	13,5	0	1	stay_constant	4	0	False
12	13	0	9	1	14	0	1	turn_right	2	-5	False
13	13	1	9,5	1	14,5	1	1	stay_constant	4	0	False
14	14	1	10	1	15	1	1	turn_left	0	-5	False
15	14	0	10,5	1	15,5	1	1	stay_constant	4	0	False
16	15	0	11	1	16	1	1	speed_up	5	3	False
17	17	0	11,5	1	16,5	1	2	no_change	1	50	True
18	19	0	12	1	17	0	2				

Position change as x and y coordinates can be seen in Table 5.4 in selected test scenario. It can be said that the ego car decided to take its velocity 5 times as taking 2 decelerate and 3 accelerate actions in order to perform safe driving. According to graph 5.13, it is obviously seen that ego car changes its lane 3 times. It can also be concluded that 2 lane changes violate the comfort requirement in highway since they occur when ego car has possible overtake option to car2 but it chooses to trace car2.

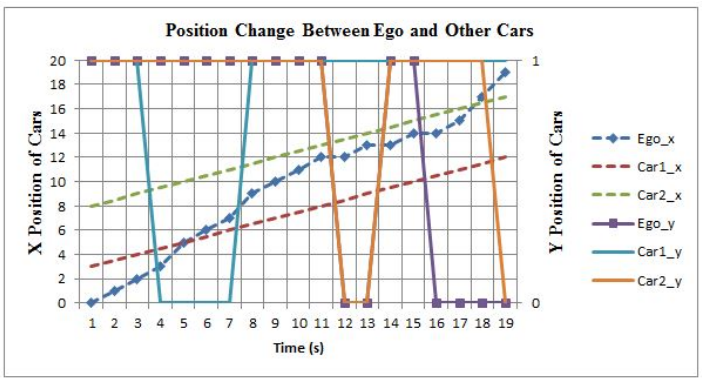

Figure 5.13. Position Change of Ego and Other Cars in x and y Coordinates

Action vs rewards graph can be seen in 5.14 according to tested episode for implemented algorithm. Return is obtained as 44 in selected test scenario.

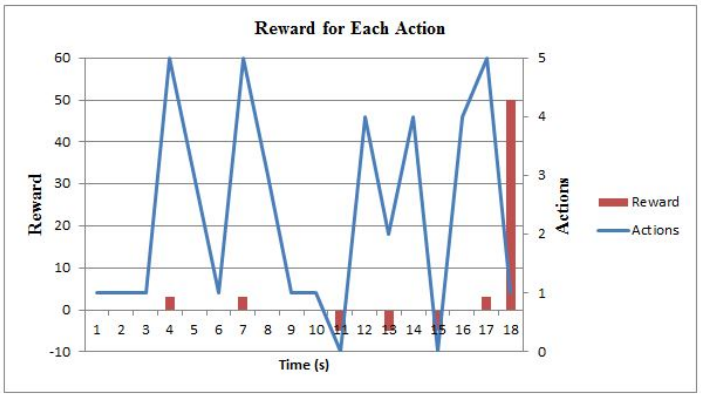

Figure 5.14. Actions vs Corresponding Rewards for Tested Scenario

5.5.1.3. Training and Test Result of Implemented Algorithm 3. Third algorithm "Expected SARSA" is trained for the same environment. Number of counted steps vs episode and total rewards vs episode graphs can be seen in Figures 5.15 and 5.16, respectively. Maximum return is obtained as 95 among trained episodes.

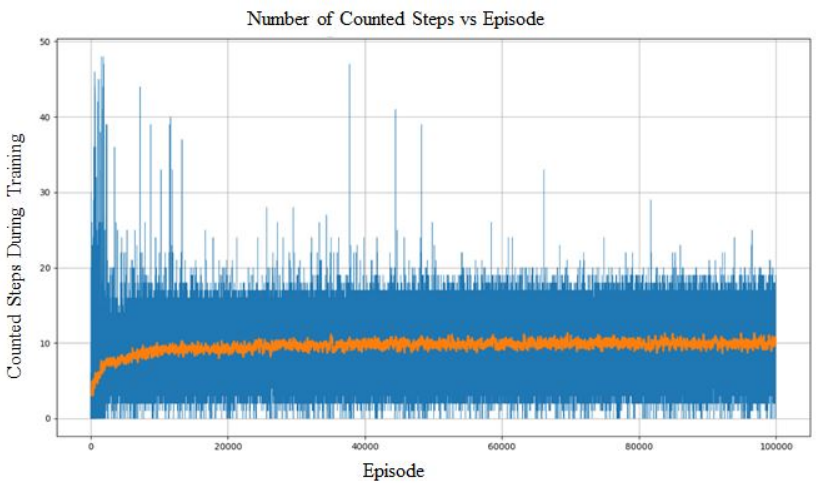

Figure 5.15. Number of Counted Steps of Ego Car in Each Episode for Algorithm 3

Specified test scenario among many of them is reported as following Table 5.5.

According to position (x, y) graph in 5.17, while ego car takes 2 lane change

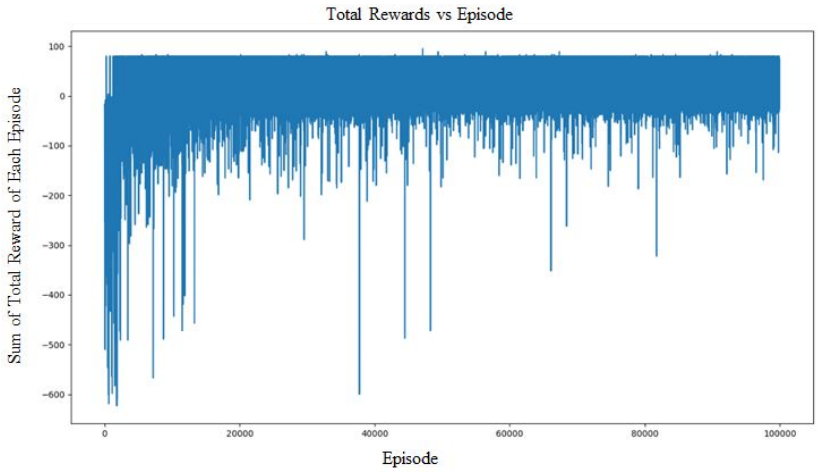

Figure 5.16. Sum of Rewards in Each Episode For Algorithm 3

Table 5.5. Information of First Tested Scenario for Algorithm 3

Time	Ego_x	Ego_y	Car1_x	Car1_y	Car2_x	Car2_y	Ego_Vel	Actions	Actions	Reward	Termination
0	0	1	3	1	8	1	1	'stay_constant'	4	0	False
1	1	1	3,5	1	8,5	1	1	'stay_constant'	4	0	False
2	2	1	4	1	9	1	1	'no_change'	1	0	False
3	3	1	4,5	1	9,5	1	1	'no_change'	1	0	False
4	4	1	5	1	10	1	1	'turn_left'	0	-5	False
5	4	0	5,5	1	10,5	1	1	'no_change'	1	0	False
6	5	0	6	0	11	1	1	'turn_right'	2	-5	False
7	5	1	6,5	0	11,5	0	1	'no_change'	1	0	False
8	6	1	7	0	12	0	1	'speed_up'	5	3	False
9	8	1	7,5	0	12,5	0	2	'slow_down'	3	0	False
10	9	1	8	0	13	0	1	'speed_up'	5	3	False
11	11	1	8,5	0	13,5	1	2	'slow_down'	3	0	False
12	12	1	9	0	14	1	1	'no_change'	1	0	False
13	13	1	9,5	0	14,5	1	1	'no_change'	1	0	False
14	14	1	10	1	15	0	1	'speed_up'	5	3	False
15	16	1	10,5	1	15,5	0	2	'stay_constant'	4	0	False
16	18	1	11	1	16	0	2	'no_change'	1	50	True
17	20	1	11,5	1	16,5	0	2				

actions, it increases its speed 3 times by taking 3 acceleration actions. It can reach goal destination without collision and sum of reward is obtained as 49 which is seen in graph 5.18

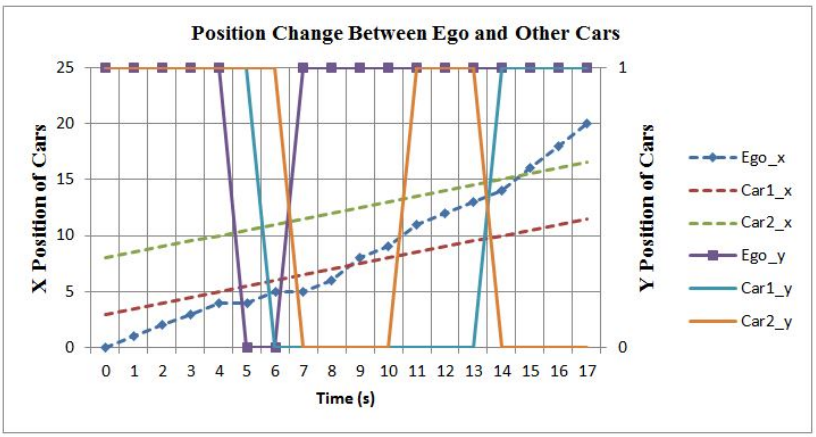

Figure 5.17. Position Change of Ego and Other Cars in x and y Coordinates

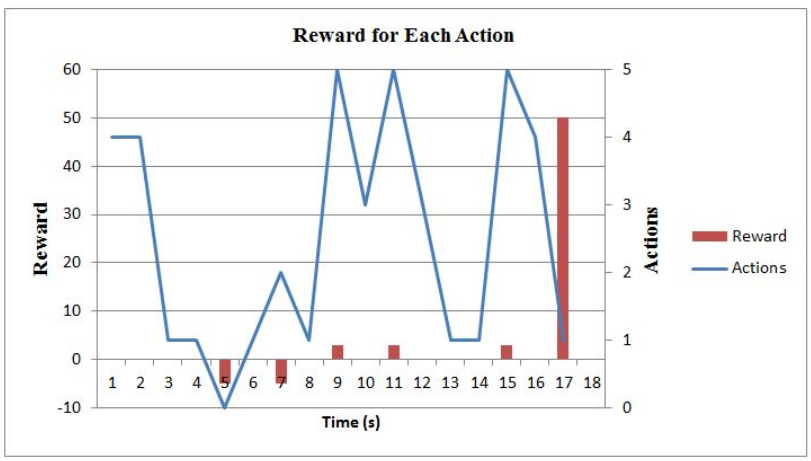

Figure 5.18. Actions vs Corresponding Rewards for Tested Scenario

5.5.2. Three Lanes with Same Direction Scenario

In the second scenario, five traffic objects including ego car start from same directions with 3 lanes in highway. An illustration of the simulation environment is shown in Figure 5.19

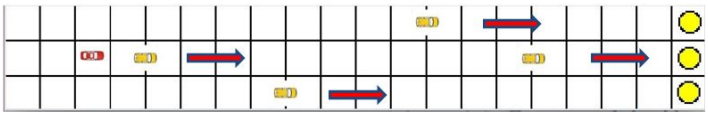

Figure 5.19. Illustration of Three Lanes with Same Direction Scenario

As it can be seen in Figure 5.19, ego car and four other cars drive towards to same direction. Other cars have constant velocity. However, they perform random lane change in highway. Ego car performs overtake the other cars by taking action "lane change" and "speed up" according to implemented algorithms in order to avoid potential collision and ensure safe driving. Training and test output are given in following Subsections based on three different implemented algorithms. It is important to note that tested scenarios with different algorithms are started with same initial states for ego and other cars in order to make proper comparison between algorithms which are implemented.

5.5.2.1. Training and Test Result of Implemented Algorithm 1. Training is performed according to Q-learn and number of counted steps of ego car and total rewards taken by it in each episode can be seen in graph 5.20 and 5.21, respectively. It can be observed that convergence of training phase takes much more time when comparing with 3 cars scenario.

Ego car learns optimal policy to ensure safe driving. In the graph 5.21, total rewards has huge fluctuation since agent tries to find higher return by taking different actions according to epsilon greedy method in each episode. For this reason, end value of epsilon greedy is selected as 0.01 instead of 0 in order not to be stop searching new action to find maximum reward in each episode. Many test scenarios are performed and one of them is reported as following Table 5.6.

In the graph 5.22 positions (x, y) of ego and other cars are represented. No collision occurs in tested scenario. Ego car takes only lane change action towards to

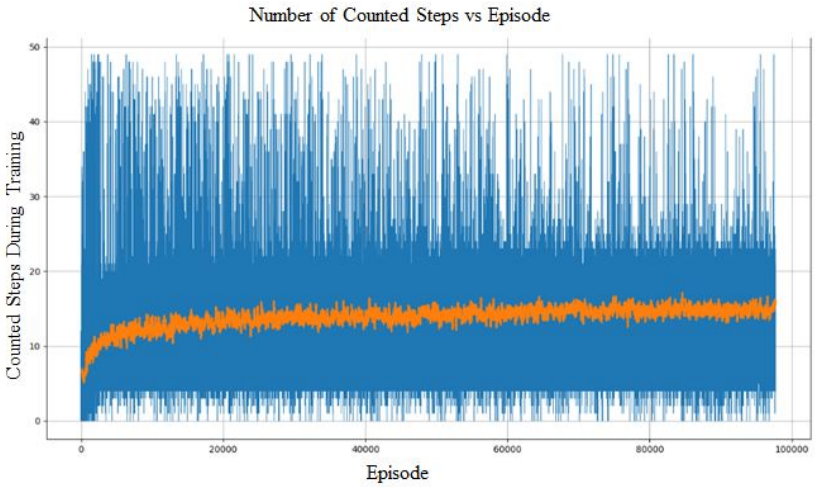

Figure 5.20. Number of Counted Steps of Ego Car in each Episode for Algorithm 1

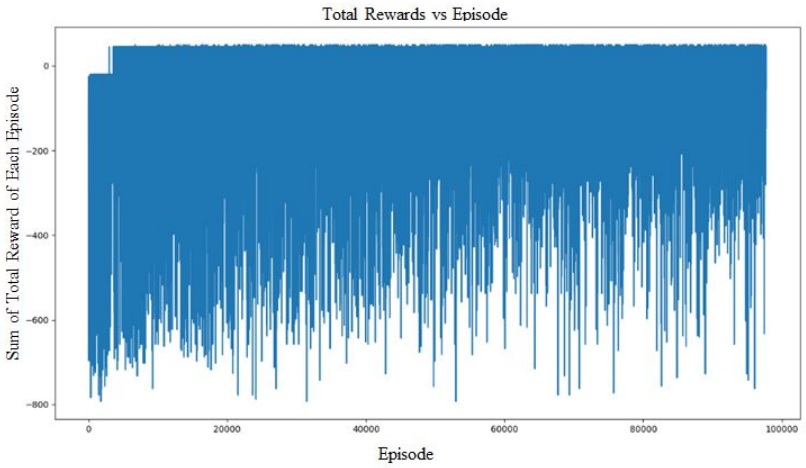

Figure 5.21. Sum of Rewards in Each Episode For Algorithm 1

Table 5.6. Information of Second Tested Scenario for Algorithm 1

Time	Ego_x	Ego_y	Car1_x	Car1_y	Car2_x	Car2_y	Car3_x	Car3_y	Car4_x	Car4_y	Ego_Vel	Actions	Actions	Reward	Termination
0	0	1	3	1	7	2	10	0	13	1	1	'stay_constant'	4	0	False
1	1	1	3,5	1	7,5	2	10,5	1	13,5	1	1	'no_change'	1	0	False
2	2	1	4	1	8	2	11	1	14	0	1	'stay_constant'	4	0	False
3	3	1	4,5	1	8,5	2	11,5	1	14,5	0	1	'stay_constant'	4	0	False
4	4	1	5	1	9	2	12	0	15	0	1	'turn_left'	0	-5	False
5	4	0	5,5	1	9,5	2	12,5	0	15,5	0	1	'no_change'	1	0	False
6	5	0	6	1	10	2	13	0	16	0	1	'turn_right'	2	-5	False
7	5	1	6,5	1	10,5	2	13,5	0	16,5	0	1	'turn_right'	2	-5	False
8	5	2	7	1	11	2	14	1	17	0	1	'stay_constant'	4	0	False
9	6	2	7,5	1	11,5	2	14,5	1	17,5	0	1	'stay_constant'	4	0	False
10	7	2	8	1	12	2	15	1	18	0	1	'no_change'	1	0	False
11	8	2	8,5	0	12,5	2	15,5	1	18,5	0	1	'stay_constant'	4	0	False
12	9	2	9	0	13	2	16	0	19	0	1	'no_change'	1	0	False
13	10	2	9,5	0	13,5	1	16,5	0	19,5	0	1	'stay_constant'	4	0	False
14	11	2	10	0	14	1	17	1	20	0	1	'no_change'	1	0	False
15	12	2	10,5	0	14,5	1	17,5	1	20,5	0	1	'no_change'	1	0	False
16	13	2	11	0	15	1	18	1	21	0	1	'stay_constant'	4	0	False
17	14	2	11,5	0	15,5	1	18,5	1	21,5	0	1	'no_change'	1	0	False
18	15	2	12	0	16	1	19	1	22	0	1	'stay_constant'	4	0	False
19	16	2	12,5	0	16,5	0	19,5	1	22,5	0	1	'no_change'	1	0	False
20	17	2	13	0	17	0	20	1	23	0	1	'no_change'	1	0	False
21	18	2	13,5	0	17,5	0	20,5	1	23,5	0	1	'no_change'	1	50	True
22	19	2	14	0	18	0	21	1	24	0	1				

right lane at time 5s, and it goes to goal destination with constant speed. Return is obtained as 45 in selected test scenario and reward change with corresponding action during 1 tested episode can be seen as following graph 5.23 according to implemented algorithm 1.

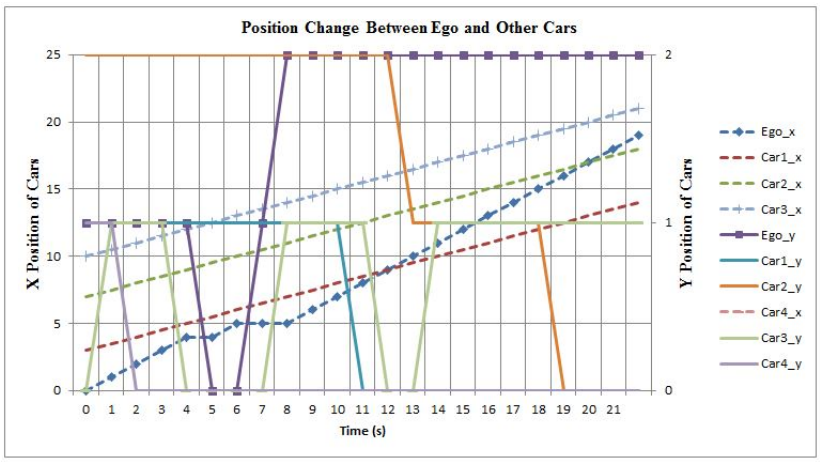

Figure 5.22. Position Change of Ego and Other Cars in x and y Coordinates

5.5.2.2. Training and Test Result of Implemented Algorithm 2. SARSA is the second algorithm trained for the "three lanes with same direction" environment as 100000 episode. Number of counted steps of ego car and total rewards taken by it in each episode can be seen as following graphs 5.24 and 5.25, respectively.

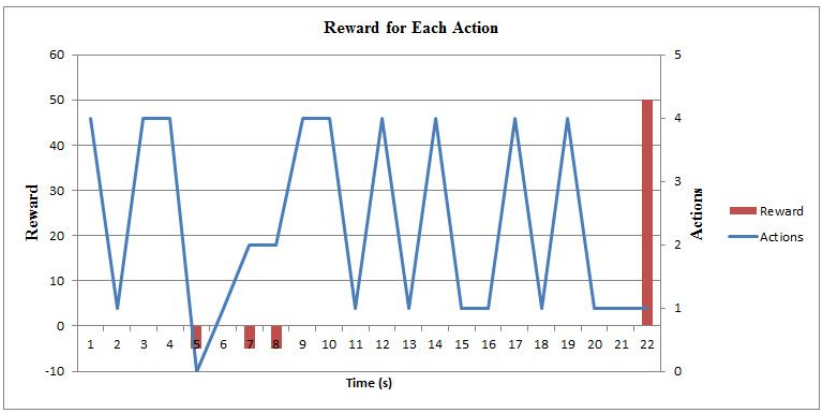

Figure 5.23. Actions vs Corresponding Rewards for Tested Scenario

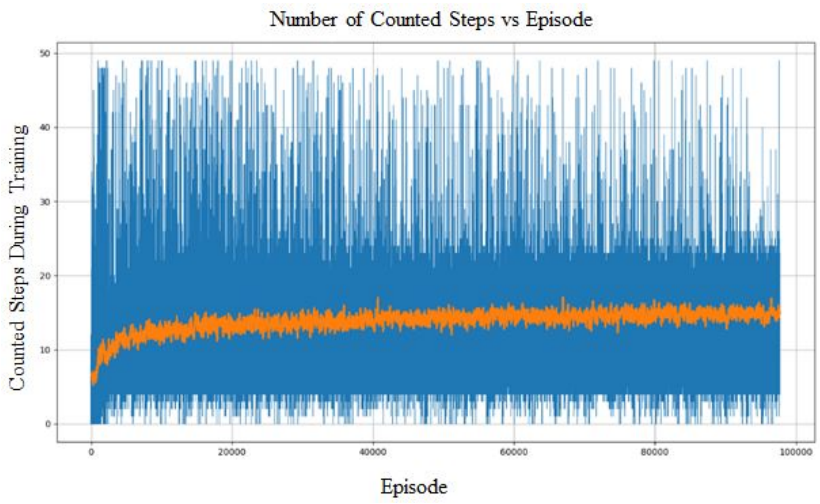

Figure 5.24. Number of Counted Steps of Ego Car in Each Episode for Algorithm 2

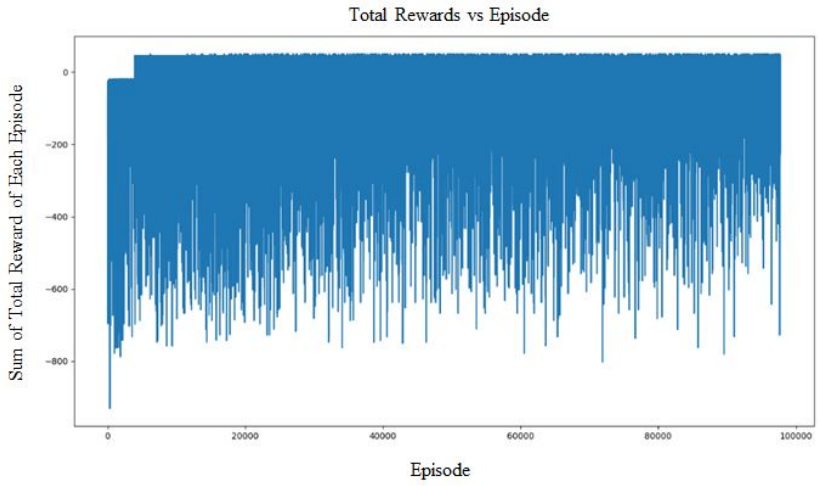

Figure 5.25. Sum of Rewards in Each Episode For Algorithm 2

Information of tested scenario regarding time, position, actions and corresponding reward is tabulated in Table 5.7 as following:

Table 5.7. Information of Second Tested Scenario for Algorithm 2

Time	Ego_x	Ego_y	Car1_x	Car1_y	Car2_x	Car2_y	Car3_x	Car3_y	Car4_x	Car4_y	Ego_Vel	Actions	Actions	Reward	Termination
0	0	1	3	1	7	2	10	0	12	1	1	'stay_constant'	4	0	False
1	1	1	3,5	1	7,5	2	10,5	0	12,5	0	1	'turn_left'	0	-5	False
2	1	0	4	1	8	1	11	0	13	0	1	'stay_constant'	4	0	False
3	2	0	4,5	1	8,5	1	11,5	0	13,5	0	1	'no_change'	1	0	False
4	3	0	5	2	9	1	12	0	14	0	1	'no_change'	1	0	False
5	4	0	5,5	2	9,5	1	12,5	0	14,5	0	1	'stay_constant'	4	0	False
6	5	0	6	2	10	1	13	0	15	1	1	'stay_constant'	4	0	False
7	6	0	6,5	2	10,5	0	13,5	0	15,5	1	1	'no_change'	1	0	False
8	7	0	7	2	11	0	14	0	16	1	1	'no_change'	1	0	False
9	8	0	7,5	1	11,5	1	14,5	0	16,5	1	1	'stay_constant'	4	0	False
10	9	0	8	1	12	1	15	0	17	1	1	'stay_constant'	4	0	False
11	10	0	8,5	1	12,5	0	15,5	0	17,5	1	1	'stay_constant'	4	0	False
12	11	0	9	0	13	0	16	0	18	0	1	'no_change'	1	0	False
13	12	0	9,5	0	13,5	0	16,5	0	18,5	0	1	'no_change'	1	0	False
14	13	0	10	0	14	0	17	0	19	0	1	'turn_right'	2	-5	False
15	13	1	10,5	0	14,5	0	17,5	1	19,5	0	1	'no_change'	1	0	False
16	14	1	11	0	15	0	18	1	20	0	1	'stay_constant'	4	0	False
17	15	1	11,5	0	15,5	0	18,5	1	20,5	0	1	'no_change'	1	0	False
18	16	1	12	0	16	0	19	1	21	0	1	'no_change'	1	0	False
19	17	1	12,5	0	16,5	0	19,5	1	21,5	0	1	'no_change'	1	0	False
20	18	1	13	0	17	1	20	1	22	0	1	'no_change'	1	50	True
21	19	1	13,5	0	17,5	1	20,5	1	22,5	0	1				

According to selected scenario and implemented algorithm, Position change as x and y coordinates can be seen in graph 5.26. It is obvious that other cars changes their position randomly whereas ego car takes only 2 actions and completes its driving with constant speed until goal destination without any collusion.

Action vs rewards graph can be seen in 5.27 according to tested episode for

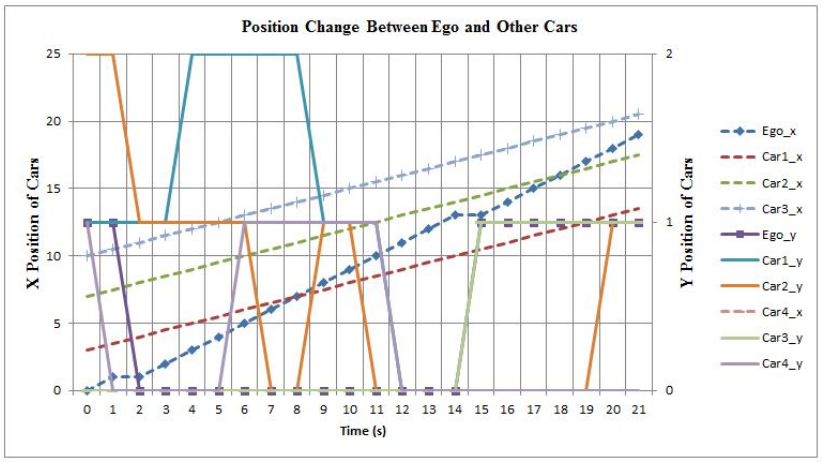

Figure 5.26. Position Change of Ego and Other Cars in x and y Coordinates

implemented algorithm. Return is obtained as 40 in selected test scenario.

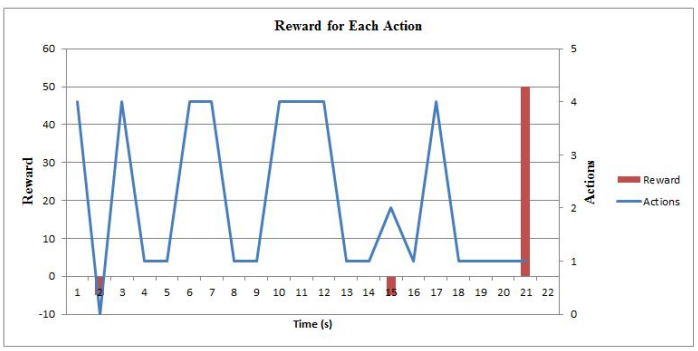

Figure 5.27. Actions vs Corresponding Rewards for Tested Scenario

5.5.2.3. Training and Test Result of Implemented Algorithm 3. "Expected SARSA" which is the third algorithm is trained for same environment. Number of counted steps vs episode and total rewards vs episode graphs can be seen in 5.28 and 5.29, respectively. There is huge fluctuation available in both graphs since epsilon greedy method allows keeping exploration instead of being performed purely exploitation by agent.

Specified test scenario among many of them is reported as following Table 5.8.

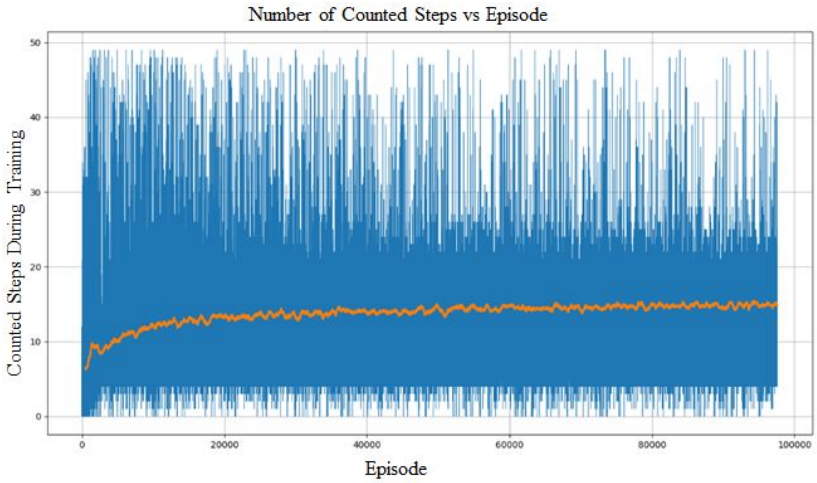

Figure 5.28. Number of Counted Steps of Ego Car in Each Episode for Algorithm 3

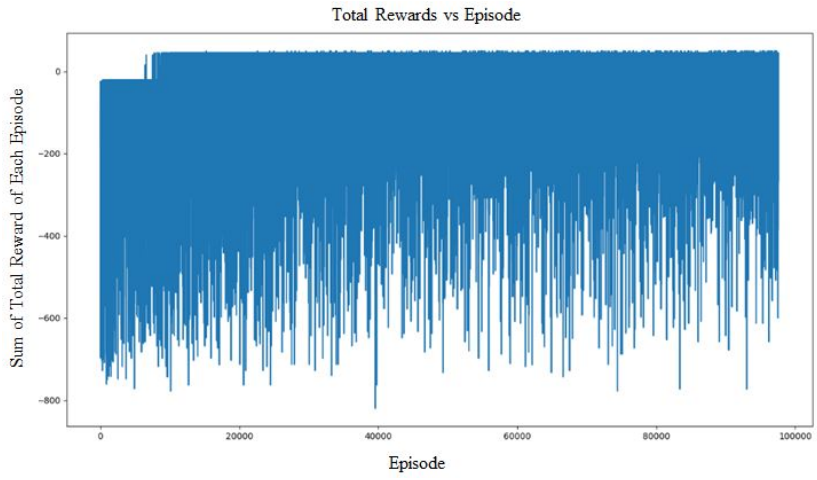

Figure 5.29. Sum of Total Rewards in Each Episode For Algorithm 3

Table 5.8. Information of Second Tested Scenario for Algorithm 3

Time	Ego_x	Ego_y	Car1_x	Car1_y	Car2_x	Car2_y	Car3_x	Car3_y	Car4_x	Car4_y	Ego_Vel	Actions	Actions	Reward	Termination
0	0	1	3	1	7	2	10	0	13	1	1	'no_change'	1	0	False
1	1	1	3,5	1	7,5	2	10,5	0	13,5	1	1	'stay_constant'	4	0	False
2	2	1	4	1	8	2	11	0	14	2	1	'stay_constant'	4	0	False
3	3	1	4,5	0	8,5	2	11,5	0	14,5	2	1	'stay_constant'	4	0	False
4	4	1	5	0	9	2	12	0	15	2	1	'stay_constant'	4	0	False
5	5	1	5,5	0	9,5	2	12,5	0	15,5	2	1	'turn_right'	2	-5	False
6	5	2	6	0	10	2	13	0	16	2	1	'no_change'	1	0	False
7	6	2	6,5	0	10,5	2	13,5	0	16,5	2	1	'no_change'	1	0	False
8	7	2	7	0	11	2	14	0	17	2	1	'stay_constant'	4	0	False
9	8	2	7,5	0	11,5	2	14,5	0	17,5	2	1	'stay_constant'	4	0	False
10	9	2	8	0	12	2	15	0	18	2	1	'no_change'	1	0	False
11	10	2	8,5	0	12,5	2	15,5	0	18,5	2	1	'no_change'	1	0	False
12	11	2	9	1	13	1	16	0	19	2	1	'stay_constant'	4	0	False
13	12	2	9,5	1	13,5	1	16,5	0	19,5	2	1	'no_change'	1	0	False
14	13	2	10	2	14	1	17	0	20	2	1	'no_change'	1	0	False
15	14	2	10,5	2	14,5	1	17,5	0	20,5	2	1	'no_change'	1	0	False
16	15	2	11	2	15	1	18	0	21	2	1	'no_change'	1	0	False
17	16	2	11,5	2	15,5	1	18,5	0	21,5	2	1	'no_change'	1	0	False
18	17	2	12	2	16	0	19	0	22	2	1	'no_change'	1	0	False
19	18	2	12,5	2	16,5	0	19,5	0	22,5	2	1	'no_change'	1	50	True
20	19	2	13	2	17	0	20	0	23	2	1				

According to position (x, y) graph in 5.30, while the ego car takes 1 lane change action, other cars perform lane changes randomly. It can reach goal destination without collision and sum of rewards is obtained as 45 which is seen in graph 5.31.

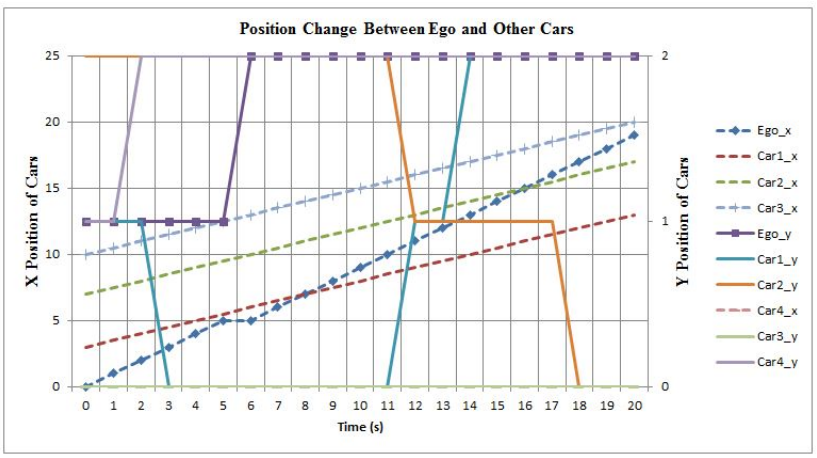

Figure 5.30. Position Change of Ego and Other Cars in x and y Coordinates

5.5.3. Three Lanes in Long Distance Highway Road Scenario

In the third scenario, ten traffic objects including the ego car start from same directions with 3 lanes in long distance highway. This scenario is tested only using Q-learning algorithm due to huge computational costs. It can be also noted that other cars have constant velocity and they do not perform lane change. An illustration of

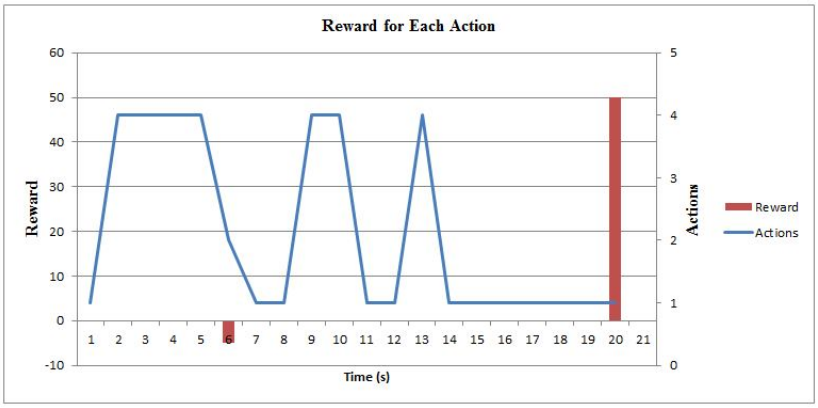

Figure 5.31. Actions vs Corresponding Rewards for Tested Scenario

the simulation environment is shown in Figure 5.32

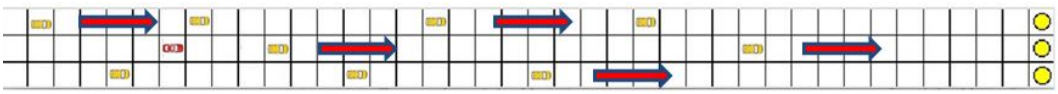

Figure 5.32. Three Lanes in Long Distance Highway Road Scenario

5.5.3.1. Training and Test Result of Implemented Algorithm 1. In order to achieve best return, the training phase takes almost 302400 second which is 84 hours. It means that training of such big environment result in huge computational cost. As a result of training phase, ego car performs good performance and it can achieve goal destination without collision almost every test scenario. Action change of ego car can be seen in Figure 5.33 instead of table.

In the graph 5.34 positions (x,y) of ego and other cars are represented . There is no collision in tested scenario. Ego car takes "lane change" action in time 9s, 21s, 27s, 28s and 39s and it reaches goal destination safely based on implemented algorithm 1. Return is obtained as 30 in selected test scenario and reward change with corresponding action during 1 tested episode can be seen as following graph 5.35 according to implemented algorithm 1.

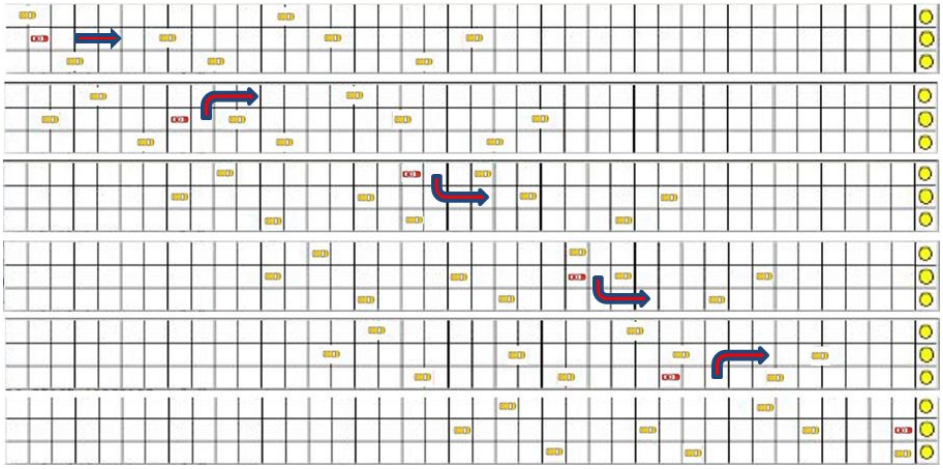

Figure 5.33. Action Change of Ego Car in Three Lanes Long Distance Highway

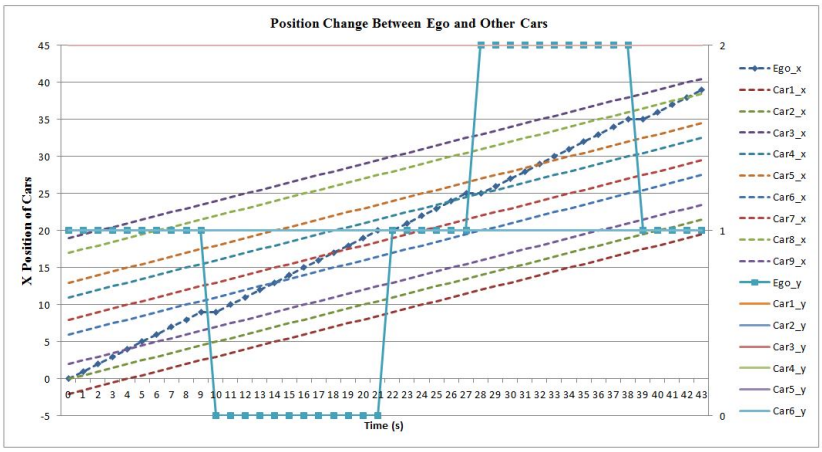

Figure 5.34. Position Change of Ego and Other Cars in x and y Coordinates

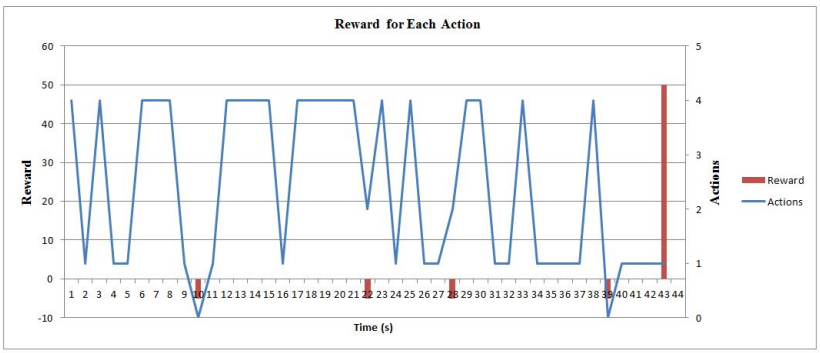

Figure 5.35. Actions vs Corresponding Rewards for Tested Scenario

5.5.4. Comparison of Tested Algorithms

The work done in this thesis was demonstrated on an Intel Core i7-3540M processor using a dual core at a clock speed of 3.00GHz. A comparison of execution times of implemented algorithms for 3 different environments can be seen in Table 5.9.

Table 5.9. Training Time (s) for Different Algorithms in Different Scenerios

Training Time (s)						
3 Vehicle Model			5 Vehicle Model			10 Vehicle Model
Sarsa	Expected Sarsa	Q-learning	Sarsa	Expected Sarsa	Q-learning	Q-learning
22945	27445	17547	43566	48249	39535	302400

Q-learning performs the best efficiency and Expected SARSA performs the worst efficiency based on training duration among three implemented algorithms according to graph 5.36 and 5.37 for both three and two lanes with same direction scenarios, respectively.

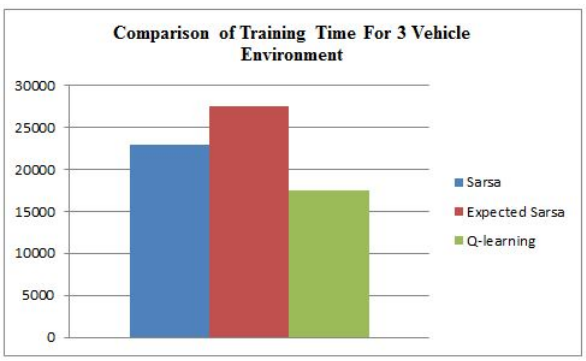

Figure 5.36. Comparison of Training Time of Implemented Algorithms for Three Traffic Objects Scenario

Number of non-collision test out of 100 tested episodes is reported in Table 5.10 for different algorithms. Implemented Q-learning method gives the best performance when we compare other algorithms, and the ego vehicle is able to avoid a collision in 84 episodes in 3 vehicle model with the other traffic objects whereas 77 episodes are able to be completed for 5 vehicle model among 100 tested episodes. It is important

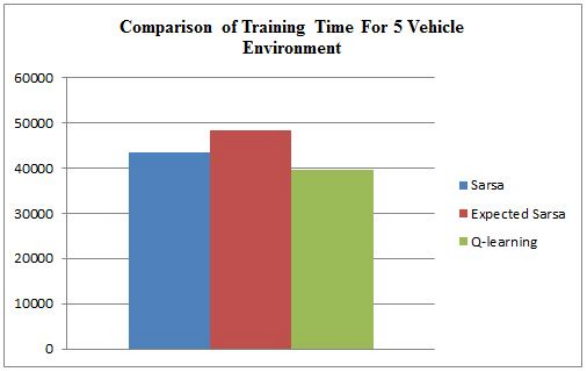

Figure 5.37. Comparison of Training Time of Implemented Algorithms for Five Traffic Objects Scenario

to note that when uncertainty of other traffic objects regarding lane change is removed from model in 10 vehicle model, algorithm gives outstanding performance and there is no collision in 100 tested episodes.

Table 5.10. Number of Non-Collision Episodes out of 100 Test Scenerio

Number of Non-Collision out of 100 Test Scenerio						
3 Vehicle Model			5 Vehicle Model			10 Vehicle Model
Sarsa	Expected Sarsa	Q-learning	Sarsa	Expected Sarsa	Q-learning	Q-learning
77	79	84	71	73	77	100

Action changes such as accelerating or decelerating and lane change during driving to ensure safety or comfort criteria is another evaluation criterion of tested scenarios. It is observed that increasing of traffic objects result in decreases of redundant acceleration or deceleration in Highway. Results of different implemented algorithms in different scenarios can be seen in Table 5.11

Table 5.11. Comparison of Tested Scenarios According to Evaluation Criteria

	Comparison of Tested Scenerios According to Evaluation Criterias						
Model Types:	3 Vehicle Model			5 Vehicle Model			10 Vehicle Model
Implemented Algorithms:	Sarsa	Expected Sarsa	Q-learning	Sarsa	Expected Sarsa	Q-learning	Q-learning
Number of Action Change:	8	7	8	2	1	3	4
Minimum Distance:	0,5	0,5	0,5	0,5	0,5	0,5	0,5

6. CONCLUSION and FUTURE WORK

6.1. Conclusion

In this thesis, a tactical to strategic decision making based on reinforcement learning algorithms modeled as Markov Decision Process is proposed to represent ego behaviors interacting with the stochastic behaviors of the environmental vehicles in highway traffic. Ego car is capable of lane change and accelerate or decelerate in order to perform safe driving without any collision with other cars which have uncertain behavior.

The results show that policies of implemented algorithms can generalize over different types of traffic objects and environments as well as driving behaviors. It is concluded that same polices are able to be used traffic scenarios with a varied number of cars.

In the Test Results Section, comparison of implemented algorithms is performed based on specified evaluation criteria, execution times in training and accuracy of tested algorithms regarding number of non-collision episodes. The results of simulation show that Q-learning algorithm tends to converge a little slower when comparing Sarsa and Expected Sarsa algorithms. It can be seen that while maximum return is obtained as 95 in "Two Lanes with Same Direction Scenario" in training phase for implemented Sarsa and Expected Sarsa algorithms, 83 return is obtained for implemented Q-learning algorithms. As a result of number of non-collision test out of 100 tested episodes, implemented Q-learning method gives the best performance for all tested scenario. It has also minimum training duration among three implemented algorithms. It is noted that implemented Expected Sarsa algorithm is more complex computationally than implemented Sarsa algorithm for all scenario. It can be also deduced that variance in update step for Expected Sarsa algorithm is smaller than the variance for Sarsa algorithm and this variance difference is observed as largest when exploration phase

has large amount of data. The agent that is trained in implemented Expected Sarsa algorithm learns faster than others due to lower variance. It can also be seen from the result of number of action changes that agent performs in different algorithms. Finally, the approaches work with a good level of efficiency and safety in tested scenarios based on evaluation criteria and number of non-collision episodes.

6.2. Future Work

The defined work in this thesis was modelled in discrete space in order to make simple evaluation. Ego vehicle states can only take integer values and change at every time step t. However, although continuous space gives huge computational cost and complexity, behavior of traffic objects is changing continuously and state of traffic system changes continuously over time in response to continuous stimuli as in the real world. Therefore, using continuous space would be natural and more realistic in the future.

In addition to using continuous space, further improvements would be about other traffic object behavior. In this thesis, although other cars perform lane change randomly, they have constant velocity in highway. It is not considered as realistic.

Moreover, environment would be extended in future work. New traffic objects such as trucks, pedestrians, traffic lights or merged lanes would be added to environment and decision making framework would be increased in order to handle extended environment requirements.

Finally, making real world test of simulated scenarios is vital although tested scenarios have stochastic driving behavior in this thesis. In the real world, drivers react the other traffic object movements and it is possible to make mistakes. Therefore, actual validation must be performed by real world testing.

REFERENCES

1. *Global status report on road safety 2018*, World Health Organization, Nov 2019, https://www.who.int/violenceinjuryprevention/roadsafetystatus/2018.

2. Thorpe, C., M. H. Hebert, T. Kanade and S. A. Shafer, "Vision and Navigation for the Carnegie-Mellon Navlab", *IEEE Trans. Pattern Anal. Mach. Intell.*, Vol. 10, No. 3, pp. 362–372, May 1988, http://dx.doi.org/10.1109/34.3900.

3. Jochem, T., D. Pomerleau, B. Kumar and J. Armstrong, "PANS: a portable navigation platform", *Proceedings of the Intelligent Vehicles '95. Symposium*, pp. 107–112, Sep. 1995.

4. Singh, S., *The DARPA Urban Challenge: Autonomous Vehicles in City Traffic*, 11 2009.

5. Shalev-Shwartz, S., N. Ben, A. Cohen and A. Shashua, *Long-term Planning by Short-term Prediction*, 02 2016.

6. Brechtel, S., T. Gindele and R. Dillmann, *Probabilistic MDP-Behavior Planning for Cars*, 10 2011.

7. Kuwata, Y., S. Karaman, J. Teo, E. Frazzoli, J. How and G. Fiore, "Real-Time Motion Planning With Applications to Autonomous Urban Driving", *Control Systems Technology, IEEE Transactions on*, Vol. 17, pp. 1105 – 1118, 10 2009.

8. Karaman, S. and E. Frazzoli, "Sampling-based Algorithms for Optimal Motion Planning", *International Journal of Robotic Research - IJRR*, Vol. 30, pp. 846–894, 06 2011.

9. Janai, J., F. Guney, A. Behl and A. Geiger, *Computer Vision for Autonomous Vehicles: Problems, Datasets and State-of-the-Art*, 04 2017.

10. Ulbrich, S. and M. Maurer, "Probabilistic online POMDP decision making for lane changes in fully automated driving", *16th International IEEE Conference on Intelligent Transportation Systems (ITSC 2013)*, pp. 2063–2067, 2013.

11. Kim, C. and R. Langari, "Adaptive Analytic Hierarchy Process-Based Decision Making to Enhance Vehicle Autonomy", *IEEE Transactions on Vehicular Technology*, Vol. 61, pp. 3321–3332, 2012.

12. Wei, J. and J. M. Dolan, "A robust autonomous freeway driving algorithm", *2009 IEEE Intelligent Vehicles Symposium*, pp. 1015–1020, June 2009.

13. Niehaus, A. and R. Stengel, "Probability-Based Decision Making for Automated Highway Driving", Vol. 43, pp. 1125 – 1136, 11 1991.

14. Wei, J. and J. M. Dolan, "A robust autonomous freeway driving algorithm", *2009 IEEE Intelligent Vehicles Symposium*, pp. 1015–1020, 2009.

15. Schubert, R. and G. Wanielik, "A Unified Bayesian Approach for Object and Situation Assessment", *IEEE Intelligent Transportation Systems Magazine*, Vol. 3, pp. 6–19, 2011.

16. Schubert, R., K. Schulze and G. Wanielik, "Situation Assessment for Automatic Lane-Change Maneuvers", *IEEE Transactions on Intelligent Transportation Systems*, Vol. 11, pp. 607–616, 2010.

17. Brechtel, S., *Dynamic Decision-making in Continuous Partially Observable Domains: A Novel Method and its Application for Autonomous Driving*, Ph.D. Thesis, 07 2015.

18. Nilsson, M., *Decision making for automated vehicles in merging situations - using partially observable Markov decision processes*, 2014.

19. Chandiramani, J. R., *Decision Making under Uncertainty for Automated Vehicles*

in Urban Situations Master of Science Thesis For the degree of Master of Science in Systems and Control at Delft University of Technology, 2017.

20. Hastie, T., R. Tibshirani and J. Friedman, *Overview of Supervised Learning*, pp. 9–41, Springer New York, New York, NY, 2009, https://doi.org/10.1007/978-0-387-84858-7-2.

21. Sutton, R. S. and A. G. Barto, *Introduction to Reinforcement Learning*, MIT Press, Cambridge, MA, USA, 1st edn., 1998.

22. LeCun, Y., Y. Bengio and G. Hinton, "Deep Learning", *Nature*, Vol. 521, pp. 436–44, 05 2015.

23. Busoniu, L., R. Babuska, B. D. Schutter and D. Ernst, *Reinforcement Learning and Dynamic Programming Using Function Approximators*, CRC Press, Inc., Boca Raton, FL, USA, 1st edn., 2010.

24. Xu, X., L. Zuo, X. Li, L. Qian, J. Ren and Z. Sun, "A Reinforcement Learning Approach to Autonomous Decision Making of Intelligent Vehicles on Highways", *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, Vol. PP, pp. 1–14, 12 2018.

25. Ngai, D. C. K. and N. H. C. Yung, "Automated Vehicle Overtaking based on a Multiple-Goal Reinforcement Learning Framework", *2007 IEEE Intelligent Transportation Systems Conference*, pp. 818–823, 2007.

26. Ngai, D. C. K. and N. H. C. Yung, *Title A multiple-goal reinforcement learning method for complex vehicle overtaking maneuvers*, 2011.

27. Rui Zheng, Chunming Liu and Qi Guo, "A decision-making method for autonomous vehicles based on simulation and reinforcement learning", *2013 International Conference on Machine Learning and Cybernetics*, Vol. 01, pp. 362–369, July 2013.

28. Sallab, A., M. Abdou, E. Perot and S. Yogamani, "Deep Reinforcement Learning framework for Autonomous Driving", *Electronic Imaging*, Vol. 2017, pp. 70–76, 01 2017.

29. Mukadam, M., A. Cosgun, A. Nakhaei and K. Fujimura, *Tactical Decision Making for Lane Changing with Deep Reinforcement Learning*, 12 2017.

30. Watkins, C. J. C. H., *Learning from Delayed Rewards*, Ph.D. Thesis, King's College, Cambridge, UK, May 1989, http://www.cs.rhul.ac.uk/ chrisw/new$_t$*hesis.pdf*.

31. Sutton, R. S., H. R. Maei and C. Szepesvári, "A Convergent O(n) Temporal-difference Algorithm for Off-policy Learning with Linear Function Approximation", D. Koller, D. Schuurmans, Y. Bengio and L. Bottou (Editors), *Advances in Neural Information Processing Systems 21*, pp. 1609–1616, Curran Associates, Inc., 2009.

32. Watkins, C. J. C. H. and P. Dayan, "Q-learning", *Machine Learning*, Vol. 8, No. 3, pp. 279–292, May 1992, https://doi.org/10.1007/BF00992698.

33. Kaelbling, L. P., M. L. Littman and A. W. Moore, "Reinforcement Learning: A Survey", *J. Artif. Int. Res.*, Vol. 4, No. 1, p. 246, May 1996, http://dl.acm.org/citation.cfm?id=1622737.1622748.

34. Shalev-Shwartz, S., S. Shammah and A. Shashua, *On a Formal Model of Safe and Scalable Self-driving Cars*, 08 2017.

35. Hoogendoorn, S. and P. Bovy, "State-of-the-art of vehicular traffic flow modeling", *J. Syst. Cont. Eng.*, Vol. 215, pp. 283–303, 06 2001.