Technische Universität
Kaiserslautern

Fachbereich
Informatik

seda
software dependability

Master Thesis

# Cooperative Intersection Control for Autonomous and Connected Vehicles using Machine Learning

by: Rohan Ravindra Gugale
Matriculation Number : 402702

*Supervisors:*
Prof. Dr. -Ing. habil. Peter Liggesmeyer
M.Sc. Patrik Feth

November, 2018

# Declaration of Authorship

I, Rohan Ravindra Gugale, declare that this thesis titled, "Cooperative Intersection Control For Autonomous and Connected Vehicles using Machine Learning" and the work presented in it are my own. I confirm that:

- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

Date:

TECHNISCHE UNIVERSITÄT KAISERSLAUTERN

# *Abstract*

Commercial Vehicle Technology

Masters of Science

**Cooperative Intersection Control For Autonomous and Connected Vehicles using Machine Learning**

by Rohan Ravindra Gugale

One of the challenging issues of the 21st century is dealing with the increasing number of vehicles and thereby their congestion problems. The modern day advanced driver assistance systems and promising futuristic technologies such as autonomous vehicles withstand the potential to ensure much more efficient and safe traffic movements by reduced and / or absent human involvement in the dynamic driving tasks. Adaptive traffic light systems using reinforcement learning have been an active area of research for several years. However, with the advent of autonomous vehicles and advancement in the fields of wireless communications, non – signalized intersections or virtual traffic lights can soon be a reality. The aim of this thesis is to evaluate the concept of a system of systems level intersection control for vehicles using reinforcement learning. Each approaching vehicle towards the intersection is an active participant within this system and the system level controller which can be realized as a part of the intersection infrastructure, governs the maneuver of the intersection for each of the participants in order to maximize the traffic flow efficiency. The thesis aims at laying out this intersection strategy using Q – learning algorithm and Q – networks from reinforcement learning. The concept has been evaluated through several iterations of characterizations, experiments and suitable metrics. This thesis also serves as a feasibility study whether an agent can be trained to control intersections using reinforcement learning. One of the main motivations of this project is that several real – world problems are being addressed using machine learning with the recent research and advancements in the field of artificial intelligence. Reinforcement learning is becoming a popular technique to develop solutions to control problems such as intersection management. For majority of the traffic management projects led by the state agencies, the concepts are validated in their early stages in suitable micro – simulation tools before the on – field implementations and thus the micro – simulator used in the scope of this thesis is the SUMO – 'Simulation of Urban Mobility' simulator developed by the Deutsches Zentrum für Luft und Raumfahrt.

# *Acknowledgements*

I would like to express my gratitude to my supervisor M.Sc. Patrik Feth for the useful comments, remarks and engagement throughout the learning process of this master thesis. I would like to thank him for introducing me to the topic, laying forward this novel concept of reinforcement learning based system of systems level intersection management as well as for his strong support along the way. Also, I would like to thank Prof. Dr.-Ing. habil. Peter Liggesmeyer for giving me this opportunity to pursue this Thesis at the Chair Software Engineering: Dependability at the Technische Universität Kaiserslautern. I would like to thank Mr. Pascal Gerber for his precious time and support during the Thesis. I would also like to thank everyone involved and who have willingly shared their precious time during the completion of this Thesis ...

# Contents

# List of Figures

# List of Tables

# List of Abbreviations

| | |
|---|---|
| **ADAS** | **A**dvanced **D**river **A**ssistance **S**ystems |
| **ESP** | **E**lectronic **S**tability **P**rogram |
| **TCS** | **T**raction **C**ontrol **S**ystems |
| **V2X** | **V**ehicle to **X** |
| **V2V** | **V**ehicle to **V**ehicle |
| **I2V** | **I**nfrastructure to **V**ehicle |
| **V2I** | **V**ehicle to **I**nfrastructure |
| **AI** | **A**rtificial **I**ntelligence |
| **OEM** | **O**rignal **E**quipment **M**anufacturer |
| **ITS** | **I**ntelligent **T**ransportation **S**ystems |
| **VANET** | **V**ehicular **A**dhoc **N**etwork |
| **MANET** | **M**obile **A**dhoc **N**etwork |
| **WANET** | **W**ireless **A**dhoc **N**etwork |
| **RL** | **R**einforcement **L**earning |
| **VRU** | **V**ulnerable **R**oad **U**sers |
| **IEEE** | **I**nstitute for **E**lectrical and **E**lectronics **E**ngineers |
| **DSRC** | **D**edicated **S**hort **R**ange **C**ommunication |
| **SUMO** | **S**imulation of **U**rban **M**obility |
| **DLR** | **D**eutsches Zentrum für **L**uft und **R**aumfahrt |
| **LiDAR** | **Li**ght **D**etection & **R**anging |
| **TraCI** | **T**raffic **C**ontrol **I**nterface |
| **GUI** | **G**raphical **U**ser **I**nterface |
| **ML** | **M**achine **L**earning |
| **SL** | **S**upervised **L**earning |
| **UL** | **U**nsupervised **L**earning |
| **MDP** | **M**arkov **D**ecision **P**rocess |
| **TD** | **T**emporal **D**ifference |
| **DP** | **D**ynamic **P**rogramming |
| **CNN** | **C**onvolutional **N**eural **N**etwork |
| **DQN** | **D**eep **Q** **N**etwork |
| **CIM** | **C**ooperative **I**ntersection **M**anagement |
| **DDPG** | **D**eep **D**eterministic **P**olicy **G**radient |
| **API** | **A**pplication **P**rogramming **I**nterface |
| **ACC** | **A**daptive **C**ruise **C**ontrol |
| **SSV** | **S**afety **S**upervisor |

# Chapter 1

# Introduction and Literature

Automobile technology has come a long way since the first vehicle with a two stroke gasoline engine developed by Carl Friedrich Benz in the year 1879. The patent number – 37435: 'Vehicle powered by a Gas engine' is regarded as the birth certificate of the first automobile. The newspapers illustrated the first public outing of the three wheeled Benz Patent Motor Car in the year 1886 as the model number 1 [Mernd]. Since their development and wide spread adoption, efficient management of traffic, especially at an intersection has become a challenge within the transport systems for maintaining a safe and smooth traffic flow. The conventional traffic light based intersection management methods have been efficient but are not able to deal with the increasing mobility and societal challenges.

This thesis attempts to improve the management of traffic by means of machine learning approaches that can be trained to optimize the flow of vehicles using the knowledge about the traffic conditions. With improved intra – vehicular communication technologies and automation of the driving functions, cooperative intersection control can be realized for an efficient and safe movement across the intersection. This project is based on the concept of an infrastructure mounted system of systems level intersection controller, wherein the system represents an agent or a controller that regulates the intersection and the approaching individual vehicles represent the participants or members. These members demand release requests to the controller system to maneuver the intersection and the controlling system issues the subsequent release commands based on a strategy or a policy that it has learned using reinforcement learning during its training tenure.

This concept of reinforcement learning based system of systems level intersection management is demonstrated using the SUMO simulator developed by the Deutsches Zentrum für Luft und Raumfahrt. The $1^{st}$ chapter – introduction and literature gives a brief overview of the advancements in the field of ADAS, different levels of driving automation and V2X communication, since these are the recommended pre – requisites for the realization of such cooperative intersection management systems. The vehicles should be able to maneuver the intersection within the time window provided by the system in order to improve the traffic flow. Inability to do so or reliance on the humans might pose a threat to the other participants within the system. The introduction chapter i.e. chapter 1 also includes the basics of artificial intelligence, in depth explanation of reinforcement learning and its different approaches in context of its suitability with respect to this project. The $2^{nd}$ chapter on the related work provides insights into the classical intersection management concepts along with the research that has been already carried out in this field to improve traffic efficiency. Majority of the mentioned works deal with optimization of the traffic light phases using machine learning or intersection maneuvers

concerning only the ego – vehicles; none of them approach the intersection management problem with a holistic or a system level approach. The $3^{rd}$ chapter comes up with a solution starting with a highly simplistic intersection model with a simple representation in order to establish an interface between reinforcement learning and the SUMO simulator via the TraCI API; progressing through a few iterations to a complex intersection scenario that can be witnessed in our every day lives. Limited applicability of the existing verification and validation techniques, methods and tools for AI controlled systems motivates researchers for the need to gain confidence in safety if such systems are to be used in real environments [FSA17]. The $5^{th}$ chapter on future work and conclusion presents the concept of a safety supervisor to ensure safety of the AI based intersection management system and possibilities for future lines of research in context of this project.

Intersections are dangerous traffic situations and also cause congestions. With a goal of contributing to efficient and safe mobility in the future, intersections are a good target. Thus, this work addresses the devisal of an intersection controller or an agent that receives information about the incoming vehicles or participants and based on this information, controls their movements across the intersection using a strategy that it has learnt during the training tenures.

## 1.1 Advanced Driver Assistance Systems

The perpetual advancements in the field of vehicular electronics led to the advent of Advanced Driver Assistance Systems (ADAS) that eventually assist the driver in the process of maneuvering the vehicle along the course of the road. This section puts forward the broad classification of advanced driver assistance systems and introduces them in context of our project of system of systems level intersection control.

The ADAS provide the driver with active support for lateral and / or longitudinal control. One of the main functions of the ADAS is making environment perception and its evaluation easier for the driver [Kna+09]. The primary goal behind the development of such systems is reduction of traffic related accidents and 'Vision Zero' i.e. the futuristic concept of mobility in which no humans will be killed or seriously injured because of traffic accidents. Traffic related fatalities amount to almost 1.2 million people worldwide and ADAS can greatly contribute to reduce such fatalities [Tru13]. The overall road traffic safety can be ensured only through its widespread adoption and hence the social and political frameworks have to regulate their implementations by law, for e.g. the Traction Control Systems (TCS) have been made mandatory in all vehicles manufactured after November 2014 within the European Union [Eur18].

The driver assistance systems can be mainly classified based on their underlying sensing technologies. Proprioceptive sensor based systems measure the state of the ego vehicle for e.g. accelerations along the axes, rotational velocity, vehicle velocity, etc. [Ben+14]. The main purpose of such systems is to control the dynamics of the vehicle in order to pursue the trajectory requested by the driver. Past research has shown that such dynamic control systems are critical systems to save lives of people after seatbelts [AO03]. The second class includes systems which use exteroceptive sensors to acquire information from the vehicular surroundings [Ben+14]. Common examples of such sensor systems are Radar, LiDAR, Ultrasonic and Global Navigation Satellite System (GNSS) receivers. They are mainly used in environment perception that aid in the behavior planning of the ego vehicle. These sensors found their way in the second generation of driver assistance systems and mainly focus on warning and informing the driver for an enhanced driving experience. In order to realize a system of systems level intersection controller, the vehicles approaching the intersection must communicate about their state to the intersection controller or agent. The proprioceptive sensor based systems aid in the assessment of the state of the ego vehicle and a suitable communication channel can relay this information to the infrastructure based controller or an agent. Based on the state of the participating vehicles, the agent or the controller can communicate the corresponding actions to be undertaken by the vehicle controller.

The field of ADAS grew rapidly exploring new horizons in making mobility safe and comfortable for its end users. The sophistication of such systems grew along the way starting from vehicle dynamics control systems, parking assistance systems using ultrasonic sensors, park maneuvering systems, Adaptive Cruise Control (ACC) using radar technology, forward collision mitigation systems using LiDAR sensors, lane departure warning systems using machine vision and intersection assistance systems. Fusing the data collected through several sensor systems helps the vehicle build a perception of the environment that facilitates in partial automation of the driving functions. Thus, merger of such lateral and longitudinal assistance systems

aids in low speed automated driving. The figure 1.1 gives an overview about the prevalent advanced driver assistance systems within a vehicle. These systems have been realized by the corresponding sensor systems [Maf16]. The ultimate aim of such systems is the detection of obstacles within the trajectory expanse of the vehicle and warning the driver about them or instituting preliminary level or emergency control.

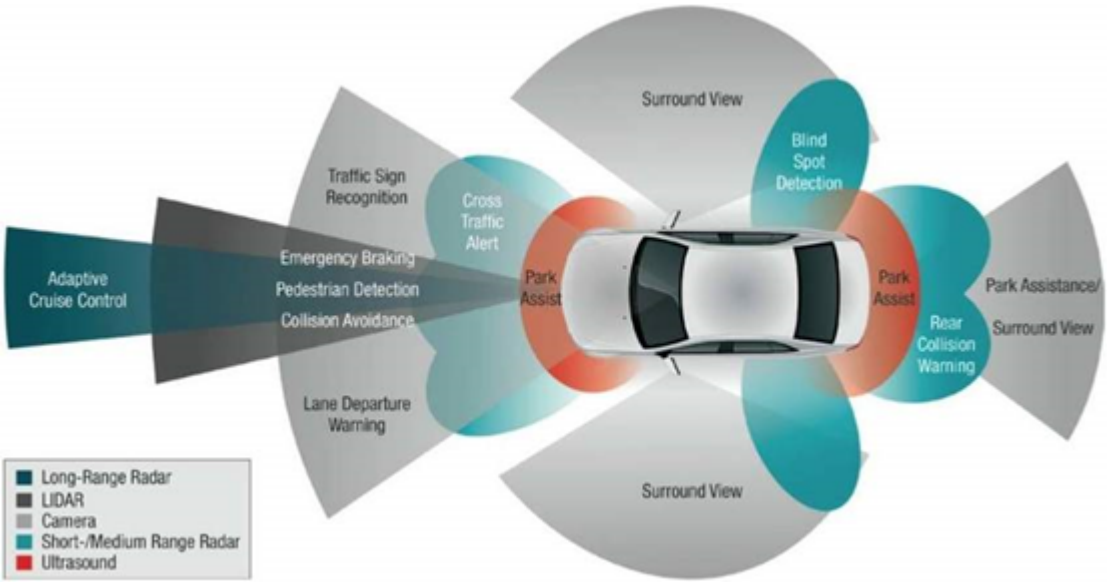


FIGURE 1.1: Sensor systems for advanced driver assistance systems [Maf16]

## 1.2 Autonomous vehicles and their History

Realizing a cooperative intersection control system requires automation of the driving functions, since the system of systems level intersection controller cannot afford to rely on the human ability to maneuver the intersection within a pre – determined time window. This section puts forward the history and possibility of such autonomous vehicles in the near future. The subsequent section also explains the classification of vehicles as per their levels of automations and their suitability in context of realization of our concept.

One of the aims of the advanced driver assistance systems in the near future would be the capability to drive autonomously without human interventions offering a pre – determined safety level that is significantly superior in comparison with humans. The future has been predicted to be inundated with autonomous cars and thus mobility as a service. With the advances in the technologies in the fields of computer vision, sensor data acquisition and higher computational capacities, autonomous driving would soon be a reality.

The development of the driverless cars dates back to the late 70's and 80's. The Mercedes Benz by Ernst Dickmann achieved a travel velocity of 100 kilometers per hour on restricted highways in absence of traffic [BSR15]. The DARPA Grand challenge of 2005 saw autonomous cars drive off – road on desert terrains. Today, autonomous cars have been driving in real world traffic during simulation runs. The automobile pioneers today are all engaged in a conquest to make these vehicles a reality. The predictions related to autonomous driving state that if the regulatory issues have been solved, by 2030, upto 15 % of the vehicles sold could be fully autonomous [KPM17]. The technology boasts of reduced fatalities, improved passenger comfort and productivity levels. Classic vehicle concepts such as steering wheels and pedals could be obsolete with the advent of autonomous vehicles and the vehicle cabins could be more of aircraft cockpits with meager controls at the hands of the passengers.

Such changes within the mobility sector may give rise to disruptive business models and result in revolutionary market changes. Tech giants such as Intel, Google, Apple or tech startups from the Silicon Valley could be the new market leaders of this autonomous era. The automobile industry would be driven by shared mobility, connectivity services and feature upgrades at a touch that could be the new business models of the future [KPM17]. The disruption in one of the largest industries across the world would change several dynamics of the world economy and still remains a widely researched topic.

However, the technology needs a lot more sophistication since comprehension of urban driving scenarios seem still a challenge because of its complex and dynamic nature. The current cognition capabilities of the systems are still at intermediate levels of development and hence all traffic situations cannot be conceived. The testing metrics currently widespread within the industry cannot be used for assessment of the performance of these intelligent machines. A few visionaries such as Prof. Dr. Phillip Koopman from Carnegie Melon University believe this to be an even greater challenge then the development of autonomous vehicles themselves. They believe that lack of proper testing and assessment metrics might even be the bottleneck in the development of this promising technology. Research scientists at the

Fraunhofer Institute for Experimental Software Engineering in Kaiserslautern, Germany also conclude that autonomy achieved by intelligent functions also generates unpredictable behavior and this unpredictability within the behavior is a threat for assuring safety since it cannot be explicitly specified. To conquer this issue, they suggest an approach that explicitly specifies a few critical combinations of vehicle behavior and runtime situations and safeguarding against them in their work 'Safety engineering for autonomous vehicles' [AFS16]. Also, an appropriate social and legal framework concerning this technology has to be devised prior to its market inundation. Within this social and legal framework, ensuring safety is one of the challenges since the existing standards have been designed for manually controlled systems and need to be augmented to cover all relevant aspects for autonomous systems [Fet+18]. The work 'Multi – aspect Safety Engineering for Highly Automated Driving' recommends steps to create a functional architecture that can realize a safe nominal behavior specification in order to apply the ISO 26262 [Fet+18]. Hence modern day driver assistance systems are sought to support the driver in the primary driving task whereby the driver can intervene in case of unknown and complex scenarios.

### 1.2.1 Levels of automation in Autonomous Driving

A professional association that develops standards in the field of mobility such as automotive, aerospace and other transport industries provides a taxonomy for the six different levels of driving automation in the context of road vehicles. This organization 'SAE International' i.e. Society of Automotive Engineers was founded in the 1905 in the United States of America and grants professional certifications and collegiate design competitions apart from standardization [Wik18b].

The six different levels of automation in the driving functions range from Level 0 (absence of driving automation) to Level 5 (full driving automation without human intervention) and have been described in detail in the SAE standard J3016. It identifies three critical actors involved in the dynamic task of driving – the user, the system that automates the driving function and the other vehicular systems. Systems that aid in emergency braking, traction control systems or lane keeping systems are not included within this taxonomy [Sta14]. The main reason for this exclusion is that such systems do not perform the dynamic driving task and only provide momentary interventions during certain pre – defined situations. The systems in which the hardware as well as the software are in conjunction able to realize sustainably the complete dynamic driving tasks are known as automated driving systems and usually refer to level 3, 4 or 5 driving automation systems.

In case of Level 0 of automation, the driver is completely responsible for the task of lateral and longitudinal control of the vehicle. Some of the lateral or longitudinal operations not in tandem can be taken up by the system in Level 1 automation for e.g. driver assistance functions such as ACC (Adaptive Cruise Control). Maneuvering an intersection involves both, lateral and longitudinal functions and hence a level 0 and level 1 autonomous vehicle may not able to participate in the system of systems level intersection management and has to rely on manual control.

The Level 2 automation or partial driving automation comprises of system driven lateral and longitudinal control of the vehicle in close supervision by the driver. The entire dynamic driving task is taken up by the automation system in the Level 3

or partial automated driving. The system is responsible for the object and environment detection along with the corresponding response and the driver or the user is only a fallback for the system. The fallback of the system completely relies on the driver and hence depends on the awareness of the driver. In case of obliviousness of the user for e.g. user in deep sleep state, the fallback might be inaccessible and may result in fatal injuries. The level 2 and level 3 autonomous vehicles would not be recommended to participate in the system of systems level intersection control, since the driver may not be in a state to take over the control of the vehicle suddenly during an intersection maneuver due to non – cognition of the situation by the onboard vehicle computer. Since the time window for a highly optimized intersection system also might be limited, the driver may not be able to cognize the environment and take suitable actions within a short span of time during a fallback situation.

The Level 4 or high driving automation boasts of the functioning of the entire driving task without any expectation of the user intervention. The only drawback of the system would be inability to react to every environmental and geographical condition such as absence of normal motorway characteristics or invisible road markings. The Level 5 or full driving automation systems would not pose this drawback of operations in limited and pre – defined scenarios [Sta14]. The vehicular automation conforming to level 4 or 5 are the ideal participants of such a system of systems level intersection control. With high levels of autonomy, the vehicles can maneuver the intersections within the time frame provided by the intersection controller, thus assuring high levels of traffic flow optimizations. They also do not pose a threat to the other participants of the system like in the case of human dependence.

The SAE J3016 standard addresses only the automation of the ego vehicle and concerns itself in context of standardization of these systems and their corresponding counterparts. Hence the concept of connected vehicles or cooperative traffic management systems have been excluded from this taxonomy.

## 1.3 Vehicle – to – X Communication

One of the important aspects in the realization of the system of systems level intersection controller is the effective communication between the system and its participating members. This section aims to put forward a brief introduction of such communication systems that have been developed in the context of automobiles and their corresponding applications.

The biggest disadvantage of the above defined driver assistance systems is their contribution to the field of traffic efficiency. Except for a handful of systems such as ACC, such systems do not contribute to the improvement of traffic efficiency. This increase can be facilitated only through the incorporation of cooperative driving and V2X (Vehicle to X) communication. Along with improved traffic efficiency, connected systems also improve the environment perception abilities of the participating vehicles. For e.g. problems pertaining to hidden vehicles that are blocked by buildings or natural infrastructure such as trees that are out of the line of sight of the drivers can be resolved by fully cooperative intersection negotiation systems facilitated through V2X communication. The sensors that are an integral part of the vehicle observe the environment and share their perceived knowledge with the other vehicles, thereby promoting cooperation and mutual aid between them. This may also reduce the cost of high end sensors systems used in all the vehicles.

Modern concepts such as vehicle platooning have proved improvement in traffic efficiencies and traffic flows through V2V (Vehicle to vehicle) communication. One of the major drawbacks of ADAS is that they communicate only with the ego vehicle, i.e. they are isolated from the other vehicles and road users [SWM12]. They are incapable of communicating with the infrastructure and the surrounding vehicles and hence cannot optimize traffic flows. An ideal ADAS should be widely used across diverse platforms of vehicles and interlinked with other users and the infrastructure. This enhanced communication could lead to overall holistic performance improvement of the future transportation systems.

Recent advancements in the field of wireless communication have been instrumental in enabling new methods of communication between the vehicles and the infrastructure. A popular concept of a vehicular ad hoc network (VANET) has been devised to share the information pertaining to environmental perception to improve safety and efficiency of operation. A wireless ad hoc network (WANET) or a mobile ad hoc network (MANET) is principally a decentralized type of wireless network. It is known as an ad hoc network because it does not rely on a pre – existing infrastructure such as a router in case of wired networks or access points in managed wireless networks [Wik18c]. The ad – hoc communication is a mode in case of operating systems that allows machines to communicate with each other without a router. It enables participating devices to create and join networks 'on – the – fly'. Thus within such VANETs, vehicles would be communicating with each other through Vehicle – to – vehicle (V2V) communication and Infrastructure – to – vehicle (I2V) communication together referred as V2X. This real – time information exchange facilitates a close cooperation between vehicles, thus forming a cooperative transport system. The Institute for Electrical and Electronics Engineers (IEEE) has published vehicular communication specifications and standards that include the usage of IEEE 802.11p for Dedicated Short Range Communication (DSRC). It also defines the peripheral functionalities and architectures. These specifications provide a framework and aid

in the development of cooperative transport systems. A cooperative intersection management system like ours can be realized through V2I (communicate the state and demand release requests) and I2V communication (issue wait or release commands) using a suitable communication channel. The participants of the systems i.e. vehicles approaching the intersection communicate about their state variables such as position, intended direction of travel, velocity, etc to the infrastructural agent. The agent issues either wait or go commands based on the characterization of the action space corresponding to the strategy or policy that it has learnt during its training period.

Research has been carried out worldwide to tackle one of the grave issues of traffic efficiency. Majority of the work includes incorporation of inter – vehicle communication such as the CVRIA – Connected Vehicle Reference Implementation Architecture led by the United States department of transportation, optimization of traffic light control based on situational traffic and city wide optimization of traffic lights [Cas17]; allocation of parking spaces prior to the arrival of vehicles or community based parking [Bosnd], etc. The integration of traffic participants within a shared network can realize the next generation of vehicles for improved traffic efficiencies and safety levels. This can be made possible only through improvements in the qualitative and quantitative information available about the traffic situation. A wireless access point at an intersection or a junction as a viable replacement for a traffic light has been a quite old innovation. However, such systems can be made prevalent only with automated vehicles, since human driving errors can result in complete disruption of the system.

## 1.4 Artificial Intelligence

In context of our project concerning system of systems level intersection controller, the agent has been trained using reinforcement learning that is a machine learning technique. This section aims at giving a brief introduction to artificial intelligence, its bifurcations and in depth foundations for reinforcement learning. The term Intelligence has Latin origins from the Latin nouns *'intelligentia'* or *'intellectus'*. It is basically the ability to include the capacity for logic, learning, self – awareness, understanding, reasoning, planning, problem solving, etc. [Wik18a]. It is commonly witnessed in humans but can also be observed in animals and plants. Intelligence demonstrated by machines is called machine intelligence or 'Artificial Intelligence'. It is in contrast with that displayed by living beings. The term was basically coined to indicate the function of mimicking the cognitive functions of the humans such as 'learning' and 'problem – solving'. The basic foundations of the term lie in the fact that human intelligence can be described in such a way that it can be simulated for machines.

Tasks considered being intelligent in the early days such as 'Optical Character Recognition' or OCR have become obsolete since they are a common and widely available technology these days. The modern day applications of the field of Artificial Intelligence include comprehension of human speech, machines capable of self – control such as autonomous cars, learning to play strategic games such as 'Chess' and 'Go' or optimal and efficient routing of delivery networks.

Artificial Intelligence is today one of the widely studied and researched areas within the field of computer science. Machine Learning (ML), knowledge curation and reverse engineering of the brain are the high level classifications of the field of AI [Kie18]. One of the widely used formal definitions of machine learning is that it is a computer program that learns from experience (E) with some given class of tasks (T) and a performance measure (P) if its performance at tasks in T as measured by P improves with E. So to summarize on a whole

$$\begin{aligned} \textit{Learning} = \; & \textit{Improving over task T} \\ & \textit{with respect to performance measure P} \\ & \textit{Based on experience E} \end{aligned}$$

In a broader sense, the steps of every machine learning task more or less include a pre – defined process chain. This process chain may deviate depending upon the applications and the context in which it is used, but on the higher abstraction level, it stays the same. The steps include collection of data, cleaning, preparing and manipulating data, training a model on the data, testing the model, subsequent iterative improvements within the data and derivation of results.

The main motivation behind using ML in majority of applications is due to the fact that intelligent systems cannot be implemented directly in a high – level language due to their complexity and inability to describe in an understandable form. A few of the examples include detection of objects using Deep Learning or in the field of robotics or autonomous vehicles in which highly complex behavior is implemented in high – level languages using hybrid approaches from ML and programming [Ert18]. The broad classification of ML falls into three main categories – supervised learning (SL), unsupervised learning (UL) and reinforcement learning

(RL). These approaches have been elaborated in the subsequent sections with main emphasis on reinforcement learning.

### 1.4.1 Supervised Learning

Supervised learning is basically a machine learning task where the functions are inferred from the labeled training data and the training data consist of a pre – defined set of training examples. A supervised learning problem is typically characterized by a learning agent, an environment in which the learning takes place or in which the data is generated and a learning unit. The task of the agent is to give the solution to a typical classification or regression problem for the given object of the environment. With respect to classification, the object is to be classified in one of the pre – defined classes or assigned a continuous value in case of regression problems. The object may be characterized by one or more attributes based on which the solution is derived. The training of the agent needs a learning unit which sends the agent a sufficiently large amount of training data with specified attributes of the object and its class affiliation or its particular continuous value of interest. The agent receives a correction for each training data through an actual or target comparison of the course unit. As per these corrections received, the agent tries to adjust the underlying classification or regression process in such a way that the deviations between the predicted results and the actual results of the training data are minimized [Kie18].

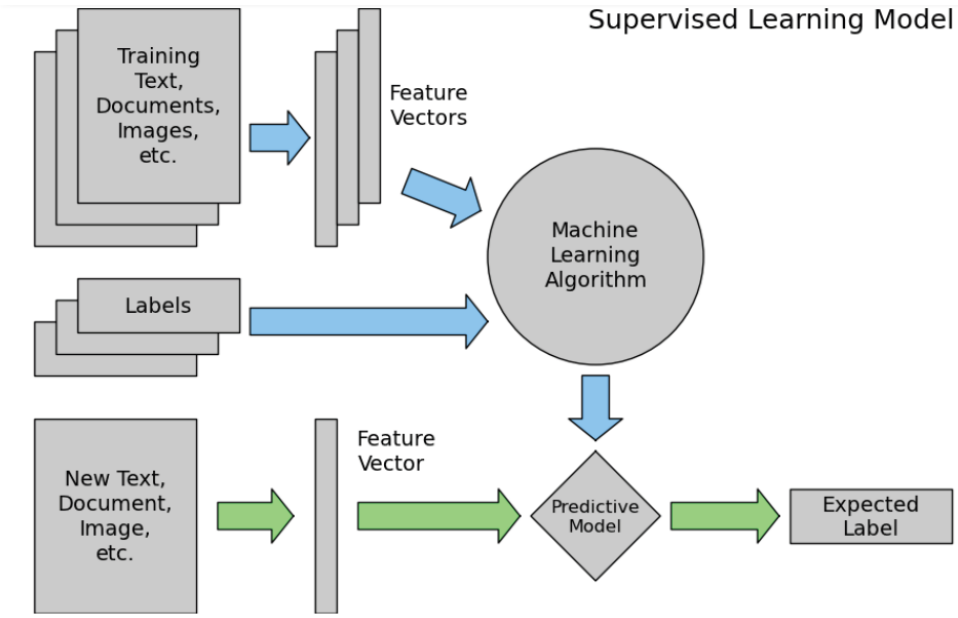


FIGURE 1.2: A typical Supervised Learning Model [Kie18]

After the above specified training process has been performed, the learning unit is omitted and the classification or regression depending upon the application is carried out on unknown objects with the similar attributes. Thus the general steps in case of SL include collection and preparation of the data, training a model on the data, testing the model, subsequent iterative improvements within the data and the hyper parameters and derivation of results.

The automotive industry is one of the biggest application centers of supervised learning. Typical data analytics applications in the automotive industry include real – environment perception, conversational user interfaces to converse with the users for an improved driving experience, time location based estimation of fuel

prices, optimal fleet management and routing of the delivery network, improving customer comfort through personalized data analysis, supply chain optimization and efficiency improvement across the production line [HNB17].

Deep learning is becoming one of the highly sophisticated fields of machine learning and is being used in the automotive industry for environment perception. The popular experiment by Hubel and Wiesel in 1959 aided in the understanding of images and understanding of structures such as edges within the primary visual cortex of the brain. Deep learning basically refers to a set of machine learning techniques that may utilize convolutional neural networks with several hidden layers for the tasks such as image classification, speech recognition and language understanding [Luc+16]. Apart from the application of AI in the much awaited level 3 automation of vehicles, it is also used to aid in several advanced driver assistance systems such as lane departure warning systems or traffic sign recognition. The figure 1.3 shows the bounding box drawn by the onboard computer across the detected objects [Jan+17].

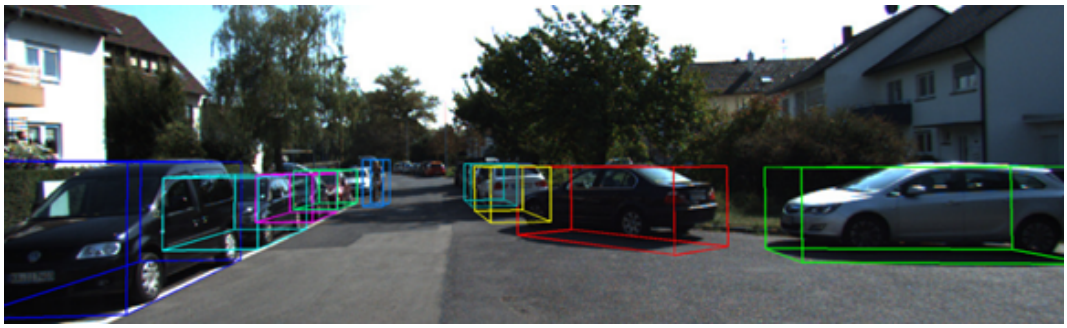

FIGURE 1.3: Supervised learning in autonomous vehicles [Jan+17]

#### 1.4.1.1 Neural Networks

An artificial neuron basically calculates a weighted sum of the inputs, adds a bias to it and based upon the type of activation, decides whether it has to be fired or not. The general equation for a neuron can be represented as –

$$Y = \sum (weight * input) + bias$$

Artificial neural networks are built by connecting several artificial neurons. The firing pattern of the artificial neuron has its inherent nature adapted from a human brain that works in the same way – a testimony of an intelligent system. The output or Y can take values ranging upto infinity (positive or negative values). The activation function checks for this 'Y' value produced by the neuron and activates it or deactivates it for its subsequent connections. The selection of the activation functions is a critical aspect in the design of networks and depends upon the application of the network output. Several such neurons connected with each other with a pre – defined architecture result in an artificial neural network. Neural networks are principally function approximators and can be used for diverse range of applications. With a training data set containing features $x$ and corresponding class $f(x)$ or a continuous value defined by $f(x)$, the artificial neural network tries to approximate the

function $f$ by $f'$ such that $|f'(x) - f(x)|$ is minimized.

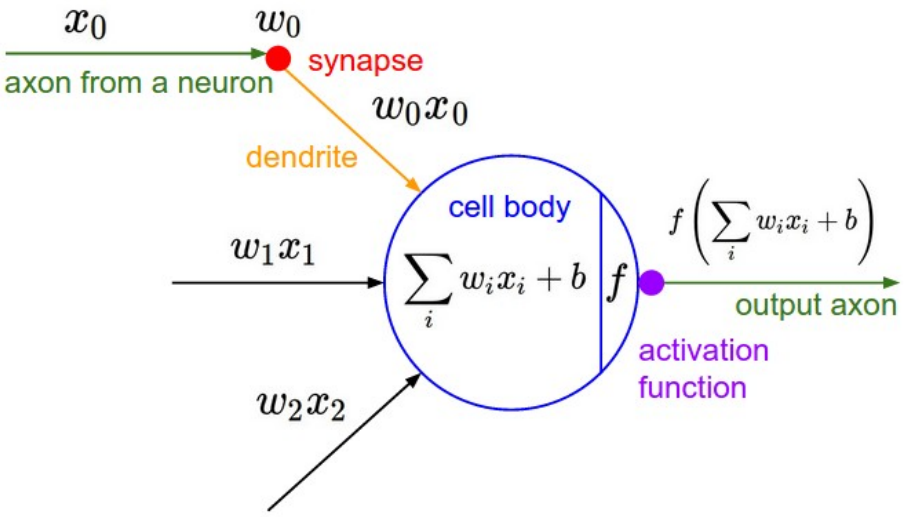


FIGURE 1.4: The mathematical model of a neuron in a neural network [LJY17]

The figure 1.4 shows a node or a neuron along with the weighted edges that connect them with the other neurons. The structure of the network and the number of neurons largely depends upon the function that is to be approximated and has no pre – laid rules to be followed but only a few guidelines. The weights of the edges are adjusted during the training period in order to approximate the expected output values or labels as closely as possible. The general procedure is to follow the guidelines and determine a structure, train the network and adjust the weights and change the structure based on the training results. The numerical inputs to the net are propagated through the input nodes to all the following nodes of the first hidden layer. The inputs are multiplied by their weighted sums of the edges and a pre – defined activation function is applied before the output. The output is then propagated to the neurons in the subsequent layer in which the similar calculations take place. This process is repeated until the output layer directly outputs the final function values.

The weights in the network decide the impact a particular input has on the overall output of the network. These weights need to be tuned in order to approximate the required behavior at the output of the network. The difference between the actual output values and the predicted output values is known as the error and the function that is used to compute this error is known as the loss function. Diverse loss functions give different errors for the same input and are selected depending upon the end application. The minimization of errors increases the prediction accuracy of the network and can be performed using the common back propagation technique. In elementary words, the existing error is transmitted to the previous layer neurons in order to modify the weights and the bias such that the error is eventually reduced. This modification of the weights is done using a function known as the optimization function. The performance of the entire network revolves around the architecture, the activation functions, the chosen loss function and the optimization algorithm.

As per the two broad classes of machine learning problems i.e. regression and classification, their corresponding loss functions are also classified as regressive loss functions or classification loss functions. In case of regression problems where the target variables consist of continuous values, the commonly used loss functions are mean square error, absolute error and smooth absolute error. The widely used loss functions in case of classification problems where the target variable consists of binary values are binary cross entropy, soft margin classifier, margin classifier and negative log likelihood.

Optimization algorithms are based on gradient descent i.e. updating the weights in the negative direction of the error derivative. The learning rate characterizes the per update change intensity of the weights. Another hyper – parameter is momentum that determines the speed at which the learning rate has to be modified as we approach the loss minima. Gradient descent updates the parameters only once whereas stochastic gradient descent updates the parameters for each training step. Adagrad is basically an adaptive gradient descent optimization algorithm that makes heavy updates for infrequent parameters whereas smaller updates for the frequent ones. This avoids the manual modifications of the learning rate. Adam is another algorithm that stands for adaptive moment estimation and differs the learning rate over a period of time. Stochastic gradient descent is simpler in implementation but has limited accuracy whereas the adaptive ones are computationally expensive but offer better accuracy levels.

One of the important applications of supervised learning concerning the automotive industry is the real – time perception of the environment and object detection for autonomous vehicles. In the former days, the classification of all candidates in the image was carried out using the sliding window approach where a rectangular region of fixed width and height slides across an image and an image classifier determines if the window has an object that might be of specific interest. Support vector machines in combination with Histogram of Orientation were popular tools for classification. However, all these methods relied on hand – crafted features that were difficult to design. With the advent of deep learning and convolutional neural networks or CNNs as a special type of networks with a special structure, this performance of these functions has been significantly boosted. The classification of an input image is done by an arrangement of various layers using the deep convolutional neural network. The former layers filter out simple features such as edges or lines of the input image using convolution. In the intermediate layers operations such as 'pooling' take place where the high dimensional input is reduced in dimensions. Through several repetitions of the processes, further complex features can be recognized within the higher layers. Based on these characteristics, the CNN makes a prediction as to which particular class does the image belong to. Apart from object detection and classification, CNN's have also demonstrated the ability of controlling self – driving vehicles using supervised learning [Boj+16]. CNN's have also reported to provide solutions for the processing of high dimensional sensor data in the areas of reinforcement learning. Representation of the perceived environment facilitates the navigation of the vehicle autonomously. Pixel based representations are basically obtained by segmentation of an image by assigning a label from a pre – defined set of categories into atomic regions that are ideally similar in color and texture with respect to their image boundaries [Jan+17]. The image 1.5 illustrates the concept of semantic segmentation accomplished through supervised learning using structured CNNs.

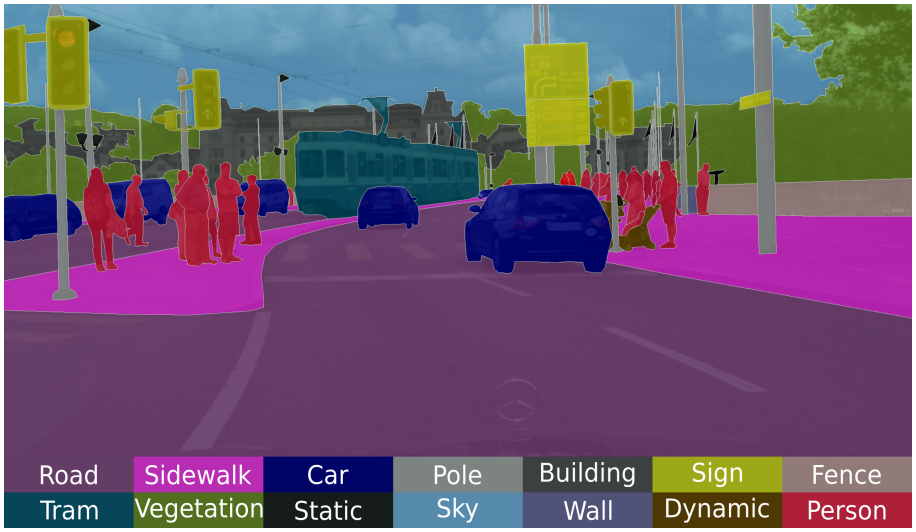


FIGURE 1.5: Semantic Segmentation of a scenario [Jan+17]

In contrast with pixel based representation, 3 D primitives are actually used for fast reconstruction of urban areas achieved through CNNs using well – defined geometrical shapes such as planes and cuboids. Other applications of CNNs and supervised learning include extraction of three dimensional information from two dimensional images shot by stereo cameras. The task of ego – motion estimation was traditionally solved by using only wheel odometry i.e. wheel encoders. But the errors induced due to wheel slip in uneven terrain led to development of hybrid approaches where visual odometry using images and SL along with the conventional wheel odometry are used [FS12].

In general, usually reinforcement learning is preferred for control tasks. In the scope of our project of system of systems level intersection control, supervised learning has not been used in order to govern or regulate the intersections, since; supervised learning may offer a solution with a probability if the agent perceives itself in a state that was not a part of the training data. In context of vehicular traffic, the distribution is completely random and the agent cannot be trained against every scenario that can be witnessed for every type of intersection. The principal idea is that neural networks are great at memorization or approximation of non – linear hypotheses, but not yet great at reasoning. This capability of reasoning can be imparted to neural networks using reinforcement learning. Hence, supervised learning is used for the tasks of environment perception for vehicles and reinforcement learning for their control and navigation [WCLF18]. The field of deep reinforcement learning is a combination of environment perception and then the subsequent control of the ego – vehicle in order to impart end – to – end control for autonomous vehicles [Jar+18].

### 1.4.2 Unsupervised Learning

As the name suggests, this machine learning approach functions in the absence of a supervisor. The major difference between supervised and unsupervised learning is the absence of a course unit and labeled training data. There may or may not be a particular training phase for the algorithms in this type of machine learning. The major aim of UL is to learn the overall structure of the data set and get a grasp of its attributes. The common problems that can be solved with this approach are clustering or dimension reduction problems. The data set is divided into suitable subdivisions or groups or clusters in case of clustering problems. The image 1.6 gives an insight into the primary methodology of unsupervised learning. The general steps in case of UL include collection of the non – categorical / unlabeled data, training a model on the data to find similarities and subsequent categorization, subsequent iterative improvements within the model and the hyper parameters and derivation of results.

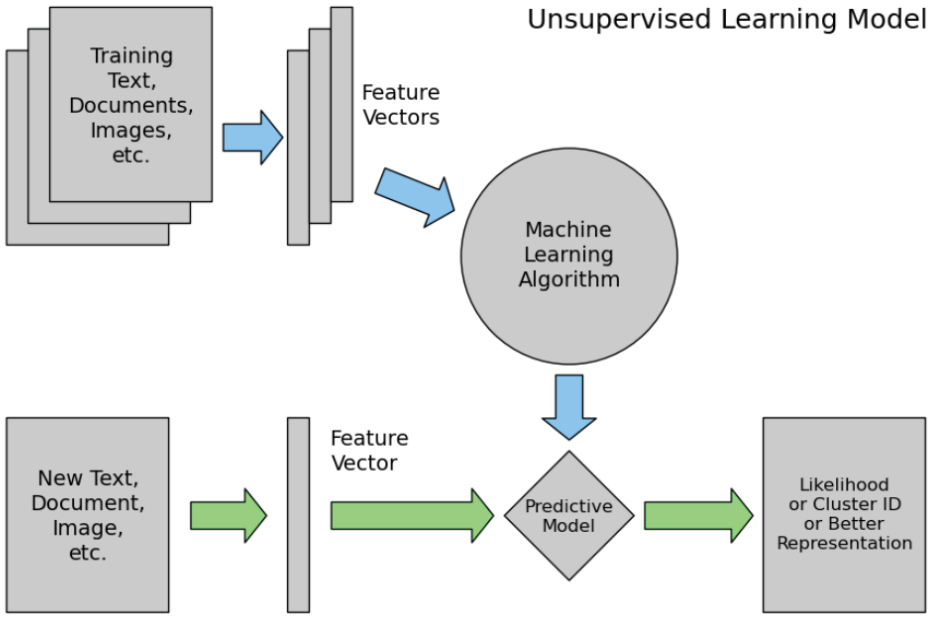


FIGURE 1.6: A typical Unsupervised Learning Model [Kie18]

The data points within a cluster are characterized by similarity whereas the similarity between different clusters should be as low as possible. The similarity within clusters can be mathematically expressed by a relation function. Euclidean distance is often used for representing this relation function between different groups or sub – groups. One of the most popular algorithms used for clustering the data points is the k – means algorithm. 'K' is often a hyper parameter for such algorithms where it indicates the number of various groups the data set is supposed to be split into. The data points are assigned to suitable clusters so that the sum of the variances weighted by the number of data sets in a cluster becomes minimal. The figure 1.7 shows that the value of 'k' should be appropriately selected for a better abstraction of the structure.

The UL methodology can also be used for the detection of anomalies. The popular applications of such techniques include in the field of medical research [CP11].

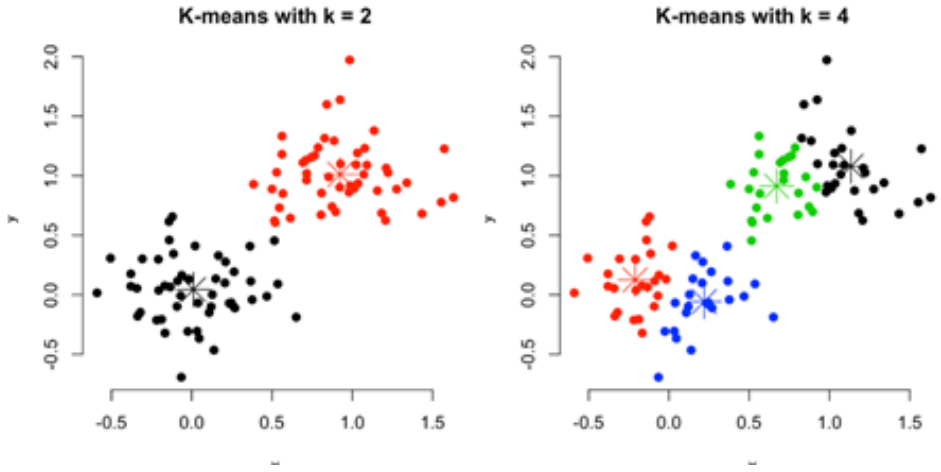


FIGURE 1.7: Unsupervised Learning using k – nearest neighbor with k = 2 & 4 [DAR]

Another class of UL algorithms includes dimension reduction where the data points in higher dimension space are projected onto a smaller subspace so as to retain originality as much as possible. The biggest advantage of such methods is that it helps to avoid the 'curse of dimensionality' in typical ML applications. The 'curse of dimensionality' basically indicates that with increase in number of features or attributes for a particular data point, the dimension space for this data point grows exponentially [Kie18]. Dimensionality reduction thus simplifies the pre – processing in order to process data faster. The prevalent techniques include principal component analysis (PCA) or singular value decomposition (SVD).

UL finds very limited applications within the automotive industry. One of the research areas using UL in the automotive industry is in the field of improvement of customer comfort and enhancing the overall driver experience. This involves collection of data in the form of user – defined settings within the cockpit, likes or dislikes of the user and thereby categorization of the user in a particular profile category. Later on based on this category, the onboard computer uses this data to enhance the driving experience by putting forward suggestions or driving advices such as pre – selection of the favorite music playlist, setting the cockpit temperature for the climate control system of the vehicle and suggestions pertaining to the driving modes.

## 1.5 Reinforcement Learning

The entire concept of the system of systems level intersection controller for autonomous vehicles revolves around the agent or controller that has been trained using the machine learning technique of reinforcement learning to govern or regulate the maneuver of the intersection by the vehicles. This section explains in detail the concept of reinforcement learning, its foundations i.e. the Markov decision processes and diversity amongst the different reinforcement learning algorithms in context of its suitability with respect to our project.

Reinforcement learning is different from other ML approaches, because it is highly focused on goal – directed learning through interaction. It is much like living beings, learning what to do, how to map different situations to different actions with a goal to maximize a numeric or a materialistic reward. The learner is not explicitly told of which actions he is supposed to perform, but must figure out on its own which actions in which situations yield the maximum reward. The actions may not only affect the immediate rewards but also subsequent situations and rewards. This approach can be seen as another class within ML alongside the other approaches that are supervised and unsupervised learning [SB17].

The basic elements of reinforcement learning (RL) are the policy, a reward signal, an agent and an environment or model of the environment. The agent captures the current complete or incomplete state of the environment and takes an action based on this state. This action leads to a change in the environment and the directly connected state. The agent receives a reward for this action performed which may not necessarily occur immediately. The maximization of this reward over a period of time is the goal of a reinforcement learning problem. Since the reward function might be an immediate response, a value function indicates what is good in the long run. In simple terms, the value of a state as the name suggests is the total amount of reward the agent can expect starting from that particular state. The policy is basically accumulation of such transitions or mapping from the observed states in the environment and what corresponding actions are to be taken in those states. Assuming that such an optimal policy exists that maximizes the sum of the rewards; the task of the agent is to approximate them as exactly as possible [HM+07].

The major difference between SL and RL is that mapping the same members to supervised learning would mean to have a training data that specifies a strategy in the form of an action for a corresponding state or a set of attributes. However, in case of reinforcement learning, the agent itself figures out the actions in the corresponding states. It is also different from unsupervised learning since no data is initially available in case of reinforcement learning.

The common examples of reinforcement learning include learning to play a game of 'Chess'. Such a problem cannot be solved using SL and needs an entirely different approach. Every move by the agent completely depends upon the opponents move i.e. the environment feedback. Most of the time, the moves by the agent would be such that they accumulate maximum rewards, i.e. get down members of the opponent. However, occasionally, the agent can also select random moves to understand how the opponent reacts or explore what consequences it might result into. This helps the agent cognize experiences he never even might have accumulated.

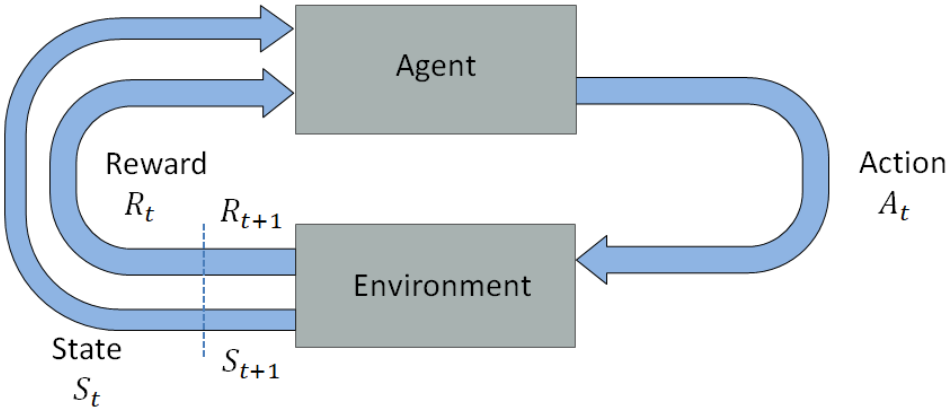


FIGURE 1.8: Reinforcement Learning Methodology

### 1.5.1 Markov Decision Processes

Sequential decision making problems are often modeled as Markov Decision processes or MDPs. They are a classical formalization of decision making processes, where the actions taken do not just affect the immediate rewards, but also the subsequent situations or states and thus the future rewards. MDPs are highly straightforward characterization of the problem of trying to learn from interactions to achieve a goal. The decision maker here is the agent and the thing it interacts with is called the environment. The agent and the environment interact in a sequence of discrete time steps

$$t = 0, 1, 2, 3, \ldots$$

At each time step t, the agent receives a partial or complete representation of the environment's current state from the finite set of states such that,

$$S_t \in S$$

And on the basis of this state S, it selects an action A from a finite set of executable actions such that,

$$A_t \in A$$

One – time step later, as a consequence of this action A, the agent receives a numerical reward R, such that the reward function $R : (S, A, S) \to \mathbb{R}$, $R(S_t, A_t, S_{t+1}) \in \mathbb{R}$ determines the reward for the transition from $S_t \in S$ for $A_t \in A$ to $S_{t+1} \in S$ in the presence of a discount factor $\gamma \in [0, 1] \subset \mathbb{R}$ that has been explained subsequently and $\mathbb{R}$ indicates the set of real numbers

$$R_{t+1} \in R \subset \mathbb{R},$$

This results in a new state $S_{t+1}$ with a transition probability $T : (S, A, S) \to [0, 1]$, which defines the conditional probability $p(S_{t+1} | S_t, A_t)$ of a transition to a state $S_{t+1} \in S$ for a given previous state $S_t \in S$ and upon selecting the action $A_t \in A$. The MDP and the agent thus generate a sequence that has continuous structure such as

$$S_0, A_o, R_1, S_1, A_1, R_2, S_2, A_3, R_3, \ldots\ldots\ldots A_{T-1}, R_T, S_T$$

A considerable abstraction of the entire problem of goal – directed learning through interaction with the environment is provided by the MDP framework. For a concise understanding, the actions can be any decisions we wish to learn and the states can be anything that might help us in making these decisions. The framework might not be suitable for all decision making problems, but has been widely useful in several applications [SB17]. The use of a reward signal is to quantify the idea of a goal and is one of the peculiar features of reinforcement learning. It is important to note that the above definition limit the states and the actions to finite states however this limitation does not hold true in case of complex reinforcement algorithms that can be used for solving problems with continuous states and actions and have been covered in the subsequent sections.

One of the most important properties of the MDP framework is that it is independent of the history. It is often known as 'memorylessness' or the Markov property and applies to the transition probability. Thus the conditional probability does not depend on the history of the previous states and actions, but only on the current state and the currently executed action [SB13].

$$\begin{aligned} P\left(s_{t+1} \mid s_t, a_t, s_{t-1}, a_{t-1}, \ldots, s_0, a_0\right) &= P\left(S_{t+1} = s_{t+1} \middle| S_t = s_t,\ A_t = a_t\right) \\ &= T(s_t,\ a_t,\ s_{t+1}) \end{aligned}$$

This relationship also holds true for the reward function. As shown above, the agent starts in a particular start state $S_0$ and takes a particular action $A_0$ which results in a state change $S_1$ and a subsequent reward $R_1$ is received. This process is repeated until a terminal state $S_T$ is reached. Along this trajectory, the goal of the agent is to find a policy $\pi$ which maximizes the sum of the resulting rewards by specifying the actions to be performed in each respective state. For a particular sequence of rewards, we seek to maximize what is known as the expected return and this return is denoted by $G_t$. One of the simplest descriptions of this return is the simple sum of rewards

$$G_t = R_{t+1} + R_{t+2} + R_{t+3} + R_{t+4} + \ldots + R_T$$

According to the commonly used discounting approach, the agent tries to select the actions in such a way that the sum of the discounted rewards it receives over the future is maximized. This is known as the discounted return and is given as follows

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

Where the parameter $\gamma$ is called as the discount rate and lies between $0 \leq \gamma \leq 1$. The discount factor actually represents the weighting between present and future rewards such that for $\gamma$ close to 0, the immediate rewards are favored whereas for $\gamma$ close to 1, far sightedness or long term rewards are favored along with the immediate rewards. Such policies can be either stochastic or deterministic depending upon the reinforcement learning algorithm. These have been defined as a probability distribution over the set of actions A for a given state $s \in S$ as follows – $\pi : (\mathrm{S}, \mathrm{A}) \rightarrow [0, 1]$ for the sake of simplicity. Formally the goal can be described as follows

$$\max_{\pi}\ \mathbb{E}_{\pi}\left[\sum_{t=0}^{\infty} \underbrace{\gamma^t r_{t+1}}_{*} \,|\, \underbrace{S_t = s}_{**}\right] \quad \forall s \in S$$

The above equation describes that we are looking for a policy $\pi$ that maximizes the sum of the expected discounted rewards (*) starting from any given state (**). Thus the optimal policy would be a strategy that advices the agent, for the state it might be in, the action to be taken that promises him the highest sum of rewards. If the reinforcement learning problem under consideration is an episodic task, i.e. the sequence terminates in a final state, then the above expectation value exists in the actual sense for $\gamma \in [0, 1]$. With continuous tasks, that do not terminate, it is possible that the series diverges against $\pm\infty$, as long as there are no transitions into the absorbing or recurrent states with no reward or the reward function is defined by the zero function [SB17]. In order to ensure that the expected value exists independent of the type of the task, in many reinforcement learning environments, $\gamma = 0.9$ is specified. Since the stochastic transitions and policies are usually defined by probabilities, the exact sum of the rewards cannot be calculated for the states, but only an expected sum can be. The expected value to be maximized for a given state and a policy can be calculated as shown below. The value of a state $s$ under a policy $\pi$ is denoted by $V_\pi(s)$ and is the expected return when starting in $s$ and following $\pi$ therafter.

$$V_\pi(s) = \left[\sum_{t=0}^{\infty} \gamma^t r_{t+1} | S_t = s\right] = \sum_{a \in A} \pi(\mathrm{s},\ \mathrm{a}) \sum_{s' \in S} \mathrm{T}\left(\mathrm{s},\mathrm{a},\mathrm{s}'\right) \underbrace{\left[\mathrm{R}\left(\mathrm{s},\mathrm{a},\mathrm{s}'\right) + \gamma V_\pi\left(\mathrm{s}'\right)\right]}_{=:B\left(s'\right)}$$

The above equation represents the sum of the expected rewards that can be achieved starting from this particular state $S_t$ and following the policy $\pi$. This calculated value for a particular state is known as the value of the state. It basically provides us with an estimate of how good it is to be in a given state. The value function can be cognized to be analogous to a long term reward whereas the reward or the reward function as short term rewards [Sil15]. The value of the start state equals the (discounted) value of the expected next state, plus the reward expected along the way. The reward for taking an action and the discounted value of the subsequent state (B(s')) are weighted on the basis of the probability distribution of policy and transition. The sum of all paths calculated from this finally represents the value. Like the state – value function for a policy $\pi$, there also exists an action – value function. It defines the value of taking an action $a$ in state $s$ under a policy $\pi$ and is denoted by $q_\pi(s,\ a)$. The action value function can be similarly given as –

$$\begin{aligned} q_\pi(s,\ a) &= \left[\sum_{t=0}^{\infty} \gamma^t r_{t+1} | S_t = s,\ A_t = a\right] \\ &= \sum_{a \in A} \pi(\mathrm{s},\ \mathrm{a}) \sum_{s' \in S} \mathrm{T}\left(\mathrm{s},\mathrm{a},\mathrm{s}'\right) \underbrace{\left[\mathrm{R}\left(\mathrm{s},\mathrm{a},\mathrm{s}'\right) + \gamma(q_\pi\left(s'_{max}\right)\right]}_{=:B\left(s'\right)} \end{aligned}$$

The action value of the start state for a given action equals the (discounted) action value of the expected next state, plus the reward expected along the way.

### 1.5.2 Reinforcement learning algorithms and the appropriate solution

As seen in the above section, the ultimate aim of a MDP is to arrive up to an optimal policy in order to maximize the rewards and reach the goal through interaction with the environment. Thus learning in context of reinforcement learning implies

learning this policy that maximizes the cumulative rewards. There are several approaches in order to achieve this optimal policy and can be divided into three broad classes – actor only, critic only and actor critic [Sil+14]. The basic difference lies between the underlying algorithms that calculate the optimal policy. The algorithms which only approximate a value function (state value function or action value function) are called critic only algorithms. In contrast with these approaches, the class of algorithms that compute a policy without a state or action value function are called actor only algorithms. A combination of both these approaches fall under the actor – critic class of algorithms [SB17].

For the actor only methods, the policy structure itself is known as the actor as it is used to select actions. It involves mapping of specific states to specific actions and does not include the calculations for the action – value or value functions for e.g. policy gradient algorithm. In simple words it is about playing an unknown video game and trying to figure out the right actions to take by introspecting upon the total score at end of each episode. The algorithms which base their action decisions upon the calculation or estimation of the value functions are known as critic only methods, since they criticize the actions based on the estimates for e.g. SARSA algorithm. It simply means learning to take actions at each step while playing a video game by keeping a track of the feedback received from the video game environment at each time step. Actor – critic methods are temporal difference methods that consist of separate memory structures to explicitly represent the policy independent of the action – value or value functions. The learning is on – policy i.e. the critic must learn and critic whether things have gone better or worse than expected whatever policy is currently being pursued by the actor. The popular example of this class is the deep deterministic policy gradient algorithm. It would simply involve a combination of a player who is learning to play a video game by taking actions and a friend who is watching him play gives him a feedback about these actions so that he can improve his game play subsequently for ensuring better scores.

The algorithms not only differ on their dependency on the approximations of the action and the state value functions but also on the knowledge of the underlying reinforcement learning environments. The model – based approaches make use of a model in the form of the transition probability T and the reward function. Each transition from a particular state with a particular action to a new state has to be assigned a transition probability along with the corresponding reward. It basically gives a mapping about the consequence of all the possible actions for a given state. If such a model is already available at our disposal, an optimal policy can be calculated without interaction with the reinforcement learning environment. It just requires this map or model (T and R) of the reinforcement learning environment. However, it is important to note that some model – based approaches do not necessarily require the model as input, but first try to approximate it by interacting with the environment. The approaches which do not possess this knowledge of the transition probabilities T and the reward function R are called model – free approaches.

Furthermore, not all the approaches allow the derivation of this optimal policy for continuous tasks and can be used only for episodic ones. Thus the effectiveness of the reinforcement learning algorithms in environments with discrete and continuous states and action spaces can also be used for their differentiation.

Furthermore, not all approaches allow the solution of continuous tasks, so that

one can distinguish between continuous and episodic tasks. Finally, the solvability of RL problems in environments with discrete and continuous state and action spaces can be used as a differentiating factor.

The table 1.1 summarizes this differentiation of the various reinforcement learning approaches [Ger18]. On the basis of the classification criteria laid above, the selection of an appropriate algorithm has to be carried out depending upon the use cases. The subsequent sections describe these approaches in detail with an overview of the algorithms along with their advantages and limitations.

TABLE 1.1: Comparison between reinforcement learning algorithms [Ger18]

| Approach | Class | Model – free | Tasks (Episodic / Continuous) | State and Action Space (Discrete / Continuous) |
|---|---|---|---|---|
| **Policy Iteration** | Critic only | ✗ | ✓ / ✓ | ✓ / ✗ |
| **Monte – Carlo Control** | Critic only | ✓ | ✓ / ✗ | ✓ / ✗ |
| **Q – Learning** | Critic only | ✓ | ✓ / ✓ | ✓ / ✗ |
| **Genetic Algorithm** | Actor only | ✓ | ✓ / ✓ | ✓ / ✓ |
| **Deep Deterministic Policy Gradient** | Actor – Critic | ✓ | ✓ / ✓ | ✓ / ✓ |

#### 1.5.2.1 Policy Iteration (Dynamic Programming)

Dynamic programming basically refers to a collection of algorithms that can be used in order to calculate the optimal policies, provided a perfect model of the environment as a MDP is available. It provides a significant and fundamental understanding of the several other reinforcement learning algorithms. As a matter of fact, all the other methods are basically an extension to achieve the same effect as dynamic programming, but with less computation and without assuming a perfect model of the environment. The major limitations of this classical method are its assumption of a perfect model and its high computational effort. One of the primary ideas of dynamic programming and thereby reinforcement learning is the use of the value functions to organize and structure the search for good policies.

In dynamic programming, computation of the state value function $V_\pi$ for an arbitrary policy $\pi$ is known as policy evaluation (prediction). The value function upon following the policy $\pi$ can thus be given as –

$$V_\pi(s) = \sum_a \pi(\mathrm{a}, \mathrm{s}) \sum_{s',\, r} \mathrm{p}\left(\mathrm{s}', \mathrm{r} \mid \mathrm{s}, \mathrm{a}\right) \left[\mathrm{r} + \gamma(V_\pi\left(s'\right)\right]$$

Since in the case of dynamic programming, the environment's dynamics are completely known, the above calculation is a system of simultaneous linear equations. Even though the calculation is tedious, the solution is straightforward. Computing this for a sequence of states starting from an initial state and following the policy $\pi$ is known as iterative policy evaluation. After the value function $V_\pi$ has been determined for an arbitrary deterministic policy $\pi$, for further optimization, we would like to know from a state $S$ whether or not we should change the policy to deterministically choose the action. Suppose rather than following the policy in state $S$, we choose an action $a$ and thereafter follow the policy $\pi$. If the value is greater than

$V_\pi(s)$, we know to select action $a$ in state $S$. This process of improving upon an original policy and making a new policy $\pi'$ by making a greedy selection with respect to the value function is known as policy improvement. After this policy $\pi$ has been improved using $V_\pi$ to yield a better policy $\pi'$, we can calculate $V_{\pi'}$ and improve it further to yield an even better policy $\pi''$. This results in continuous finite iterations to result in an optimal policy. This is known as policy iteration. The algorithm from figure 1.9 gives an overview of the entire process of policy iteration [SB17].

**Policy iteration (using iterative policy evaluation)**

1. Initialization
   $V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

2. Policy Evaluation
   Repeat
   $\Delta \leftarrow 0$
   For each $s \in \mathcal{S}$:
   $v \leftarrow V(s)$
   $V(s) \leftarrow \sum_{s',r} p(s', r|s, \pi(s))[r + \gamma V(s')]$
   $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
   until $\Delta < \theta$ (a small positive number)

3. Policy Improvement
   *policy-stable* $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
   *old-action* $\leftarrow \pi(s)$
   $\pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s', r|s, a)[r + \gamma V(s')]$
   If *old-action* $\neq \pi(s)$, then *policy-stable* $\leftarrow$ *false*
   If *policy-stable*, then stop and return $V \approx v_*$ and $\pi \approx \pi_*$; else go to 2

FIGURE 1.9: Algorithm for Policy Iteration [SB17]

Since policy belongs to the critic – only class, the state value function is used to compute the optimal policy. Each iteration consists of an evaluation and a subsequent improvement until no further improvements are seen and a deterministic policy with its corresponding value function is achieved. In many applications of reinforcement learning, only the reward function is known and the policy has to be determined in the absence of a model. Determining a model and the subsequent transition probabilities is often not feasible or computationally too expensive. For e.g. in case of our use case, the simple single lane model for realizing autonomy at intersections has such a state space characterization that there are approximately 65 million different states possible. In such a use case, laying down the entire transition dynamics is not possible and also carrying out the extensive tree search to find an optimal policy like in the case of dynamic programming does not make sense.

#### 1.5.2.2 Monte Carlo Methods

Monte Carlo methods are class of algorithms that are capable to operate without the complete knowledge of the transition dynamics. They require only experience – sample sequences of states, actions and rewards from either actual or simulated interaction with the environment. The model is required to only generate sample transitions and not the complete probability distributions of all the possible transitions. Model – free algorithms are regarded as an alternative solution to the previous method since the specification of T in comparatively less complex reinforcement

learning environments such as the Cartpole environment also turns out to be difficult. The Cartpole environment has been developed by 'Open AI' to promote research in the reinforcement learning community for tackling classical control problems using AI. It consists of a pole attached by a non – actuated joint to a cart that moves along a frictionless track and the cart is controlled by applying a force + 1 or – 1 to the cart. The goal is to prevent the pole from falling over and maintaining it in an upright position [Opend]. The Monte Carlo methods rely on well – defined returns and are used for episodic tasks. The underlying assumption is that the experience is divided into episodes and all the episodes eventually terminate irrespective of the actions selected. The value estimates and the policy changes are carried out only at the end of the episodes i.e. based on averaging the complete returns.

Dynamic programming methodology shows the all possible one – step transitions building up an extensive tree structure, in contrast with that, the Monte Carlo diagram shows only the sampled states in one episode till the end of the episode. The estimate for one state does not build upon the estimate for any other state like in the case of dynamic programming. Thus the computational effort required to estimate the value of a single state is independent of the number of states and this makes the Monte Carlo methods advantageous.

In the absence of a model, the state values are not sufficient in order to make the right estimations like in the case of dynamic programming and hence we require the action value function. With this approach, since we traverse till the end of the episode following a particular policy $\pi$, not all the state – action pairs might be visited. It will observe returns for only one action in each state. In order to solve this problem, an appropriate balance between exploration and exploitation has to be maintained.

The value function in case of Monte Carlo methods is iteratively altered to approximate the value function for the current policy as closely as possible and the policy is continuously improved with respect to the current value function in a constant loop [SB17]. The policy improvement is carried out by making the policy greedy with respect to the current action value function. This greedy policy is defined as deterministic choice of an action with the maximum action value.

$$\pi(s) = argmax_a q(s,a) \;\; for\ all\ s \in S$$

The action with the highest action value function in each state forms the optimal policy. The algorithm for arriving upto this optimal policy is the Monte Carlo ES ( Exploration Starts) which has an underlying assumption that the probability of starting from any state $S_0 \in S$ and $A_0 \in A$ i.e. all state – action pairs have a non – zero probability. The algorithm from figure 1.10 gives an overview of the entire process of Monte Carlo ES [SB17].

In the algorithm, the returns for all the state – action pairs are accumulated and averaged, irrespective of what policy was used when they had been observed. It is clearly indicative of the fact that Monte Carlo ES may not converge to any suboptimal policy. Several other concepts such as importance sampling and off – policy prediction improve the overall performance of the algorithms. Off – policy learning basically involves two policies – a target policy that is being learned and becomes the optimal policy (deterministic) and the second that is more exploratory (more

**Monte Carlo ES (Exploring Starts), for estimating $\pi \approx \pi_*$**

Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:
  $Q(s,a) \leftarrow$ arbitrary
  $\pi(s) \leftarrow$ arbitrary
  $Returns(s,a) \leftarrow$ empty list

Repeat forever:
  Choose $S_0 \in \mathcal{S}$ and $A_0 \in \mathcal{A}(S_0)$ s.t. all pairs have probability $> 0$
  Generate an episode starting from $S_0, A_0$, following $\pi$
  For each pair $s,a$ appearing in the episode:
    $G \leftarrow$ the return that follows the first occurrence of $s,a$
    Append $G$ to $Returns(s,a)$
    $Q(s,a) \leftarrow$ average$(Returns(s,a))$
  For each $s$ in the episode:
    $\pi(s) \leftarrow \arg\max_a Q(s,a)$

FIGURE 1.10: Algorithm for Monte Carlo Exploration Starts [SB17]

stochastic) and is used to generate behavior known as the behavior policy. Importance sampling is weighting the returns by the ratio of the probabilities of taking the actions under the two policies [SB17].

One of the biggest problems with this method is that it learns only at the tails of episodes. Thus in case of continuous state space applications such as our project of intersection control of vehicles using reinforcement learning, the method does not hold good. Also, the method seems to learn from greedy actions at the tail of the episodes, however if the non – greedy actions are common, the learning is particularly very slow. In the context of our project, slow learning at the end of an episode rather than with every instance controlled might negatively hamper the entire performance of the system.

#### 1.5.2.3 Temporal Difference Learning

Temporal difference learning is principally a combination of Monte Carlo control and dynamic programming. It takes up the advantages of both the approaches in order to realize a complete different but useful one. TD methods can directly learn from raw experience without the need for the model of the environment dynamics like Monte Carlo methods. Like dynamic programming, they carry out part updates without waiting for the final outcome i.e. they bootstrap. Monte Carlo methods update their $V$ of $V_\pi$ until the return following the visit is known and then use it as a target for $V(S_t)$

$$V(S_t) \leftarrow V(S_t) + \alpha [G_t - V(S_t)]$$

Where $G_t$ is the actual return following time t and $\alpha$ is the learning rate. Monte Carlo methods must wait until the end of the episode in order to increment $V(S_t)$ since only then $G_t$ is known. In contrast, TD methods need to wait only until the next time step. At the next time step they immediately form a target and make a useful update using the observed reward and the new state estimate. They bootstrap i.e. they use one or more estimated values in the update step for the same kind of estimated value.

$$V(S_t) \leftarrow V(S_t) + \alpha [R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]$$

Since TD(0) bases its update on an existing estimate, it is also a bootstrapping method like dynamic programming. Both TD and Monte Carlo methods are often known as *sample updates* since they include looking ahead to a sample successor state (or state – action pair) and using the value of the successor and the reward along the way to compute a backed – up value and then updating the value of the original state (or state – action pair). The update takes place in magnitude of the difference between the target and the predicted values and this is known as TD error given by

$$\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$$

The two main algorithms that fall within this class of actor only approach are on – policy control or SARSA and off – policy TD control or Q – learning.

**SARSA (State – Action – Reward – State – Action)**

In case of SARSA or an on – policy method, we need to estimate the value of $q_\pi(s, a)$ for a current policy behavior $\pi$ and for all states $s$ and actions $a$. The update takes place after every transition as follows

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

In case of SARSA, we continuously estimate the value of $q_\pi$ for the behavior policy $\pi$ and at the same time change $\pi$ towards greediness with respect to $q_\pi$. The general algorithm for SARSA is given in the figure 1.11 [SB17].

**Sarsa (on-policy TD control) for estimating $Q \approx q_*$**

Initialize $Q(s,a)$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(terminal\text{-}state, \cdot) = 0$
Repeat (for each episode):
  Initialize $S$
  Choose $A$ from $S$ using policy derived from $Q$ (e.g., $\epsilon$-greedy)
  Repeat (for each step of episode):
    Take action $A$, observe $R$, $S'$
    Choose $A'$ from $S'$ using policy derived from $Q$ (e.g., $\epsilon$-greedy)
    $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma Q(S',A') - Q(S,A)]$
    $S \leftarrow S'$; $A \leftarrow A'$;
  until $S$ is terminal

FIGURE 1.11: Algorithm for Temporal difference control using SARSA [SB17]

SARSA converges with a probability 1 to an optimal policy and action – value function as long as all state – action pairs are visited an infinite number of times.

**Q – learning**

One of the most popular reinforcement learning algorithm is the off – policy TD control algorithm known as Q – learning. The major difference between SARSA and Q – learning is in the update that is performed after each transition. The update for Q – learning is given as

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha [R_{t+1} + \gamma \, max_a Q(S_{t+1}, a) - Q(S_t, A_t)]$$

In case of Q – learning, the learned action – value function directly approximates

$q_*$ , the optimal action value function without depending upon the policy being followed. This simplifies the algorithm and enables convergence. Q – learning uses the $\in$ greedy strategy for the selection of an action. The action with the highest value is determined and then this action is carried out with a probability of $1- \in$ and the remaining actions are carried out with a probability proportionate to $\in$ . The optimal policy is achieved if all the state – action pairs are visited infinitely often and the increment converges eventually towards 0. Following the similar practices as in the Monte Carlo approaches, the algorithm is executed only until the changes to the Q value or the increments become negligibly small. The main difference between SARSA and Q – learning is that SARSA does not use the maximum value function Q to choose an action to update the value function $Q$. Instead of this, it takes another action $a'$ according to the policy corresponding to the value function $Q$. The algorithm has been described for reference in the figure 1.12 [SB17].

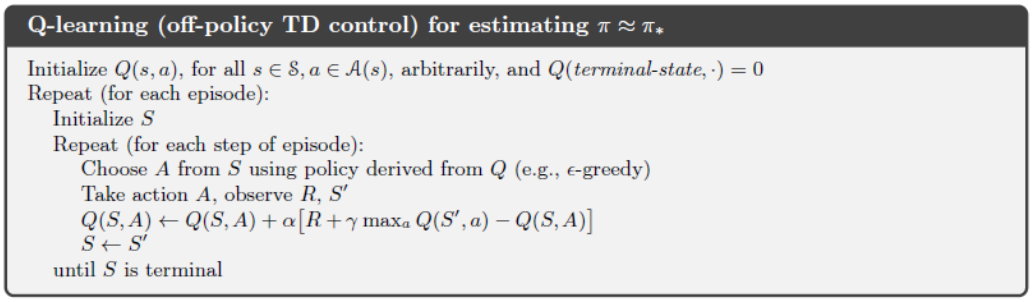


FIGURE 1.12: Algorithm for Temporal difference control using Q – learning [SB17]

In the context of our project as well, Q – learning has been used. The major advantages of using Q – learning include off – policy control, model – free approach and its ability to converge. Since the application is a real – time application to control the intersection of vehicles, an off – policy control ensures stability. Q – learning has also been proven to converge to an optimal policy as long as majority of the pairs $(s,\ a)$ are visited a large number of times during the iteration. A limited discrete action space i.e. choice of the lane from which the vehicle is to be released along with an every step update makes Q – learning highly suitable in the context of our project. As per the differentiation laid in the section 1.5.2 through the table 1.1 for the comparison between the different reinforcement learning algorithms, our application well suits the criterion put forward for Q – learning. As described in the above sections, determination of the transition probabilities is not always feasible and too heavy on computations. In the subsequent sections below it would be witnessed that the characterization of the state space for a simple single lane intersection model involves approximately 65 million different states. Also, the inherent nature of traffic is random and no function can perfectly model this randomness within the traffic. Hence drafting the transition dynamics and thereby creating a perfect model is out of scope within this application of reinforcement learning. Hence the policy iteration approach cannot not be applied within this project. The Monte Carlo control has to be avoided because of its episodic nature and non – learning during each transition but only at the tail of each episode. Within the framework of our project, the action space consists of only decisions, regarding the lane from which the $1^{st}$ waiting or approaching vehicle is to be released. Since these actions are of discrete nature and the action space is limited, the usage of highly complex algorithms such as genetic

algorithm or the actor – critic class of algorithms such as DDPG that have been explained in the subsequent section cannot be justified. With a limited and discrete action space, they may not even attain stability.

**Q – Networks and Q – Tables**

Q – tables quantify the deterministic reinforcement learning policy that has been developed by the agent in terms of numerical action – values for a relatively small state space characterization. The concept can be explained through a small application of reinforcement learning where the agent is supposed to reach a goal. Say the game consists of 10 tiles in a row and the agent spawns on any of the tiles and can move either left or to the right. The goal is to reach the fruit tile (reward + 100) and avoid the hole tile (reward – 100) resulting in termination of the episode in both the cases.

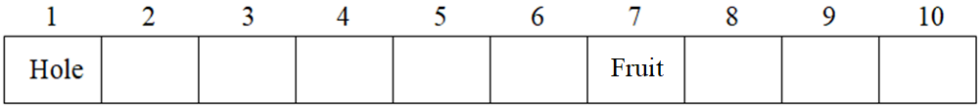


FIGURE 1.13: Reinforcement learning application using Q – tables

Applying the above described Q – learning algorithm and running the training for 1000 episodes results in a deterministic policy that can be interpreted from the Q – table. The Q – table 1.2 has been initialized to 0 and over the training period of 1000 episodes, it updates itself at each time step to give out a policy [Rlu].

TABLE 1.2: Q – table for the reinforcement learning application

| **State** | **Action** | |
|---|---|---|
| | **Left** | **Right** |
| **1** | 0 | 0 |
| **2** | -100 | 65.61 |
| **3** | 59.049 | 72.9 |
| **4** | 65.61 | 81 |
| **5** | 72.9 | 90 |
| **6** | 81 | 100 |
| **7** | 0 | 0 |
| **8** | 100 | 81 |
| **9** | 90 | 72.9 |
| **10** | 81 | 0 |

The Q – tables aid in clear understanding of the policy through a visual representation. However, as described above, they are suitable only for applications where the diversity within the states is limited. In case of applications such as playing ATARI games or maneuvering an autonomous vehicle, the state space is extremely large and conventional Q – tables cannot be used. Also, the Q – tables deliver a deterministic policy i.e. selection of an action in a particular state with the largest action – value function. However, it cannot approximate the selection of the action if the table entry for the particular state is empty and has been encountered for the first time. Neural networks are one of the viable solutions for dealing with these limitations. Neural networks in context of Q – learning may also deliver a fair amount of

robustness by approximating the suitable action selection even upon encountering a new state.

With respect to reinforcement learning and specifically concerning Q – learning, $f(x)$ represents the continuous action – values of the input state $x$ and the artificial neural network tries to approximate these action – values by $f'$ such that $|f'(x) - f(x)|$ is minimized. Thus, the function $f$ represents the optimal policy and $f'$ indicates the approximation of this optimal policy by the network. With respect to the neural network or the agent used in the context of our project to control the intersections, the network should output the probabilities for the different release actions from each lane. The release action with the highest probability is then selected. These probabilities consist of continuous values and hence we deal with regression based loss functions such as mean square error or absolute error.

In context of reinforcement learning, networks are often used in order to approximate the value functions or either the action – values concerning the action space. One of the highly researched areas of reinforcement learning today is deep Q – networks or DQNs. They are a multi – layered network that output action values $Q(s, a, \theta)$, where $\theta$ represents the network parameters [VHGS16]. The idea behind DQNs is to use them in context of reinforcement learning applications along with Q – learning and hence the name: deep Q – networks. Such DQNs analogous to the Q – tables are capable of approximating the action values for the given state $s$ and an action $a$, however for a very high dimensional state space. The typical applications of DQN include achieving almost human level or even better control at playing ATARI games, etc. This level of control was achieved through much advanced concepts such as using dual networks as two separate estimators – one of them to estimate the state value function and the other one to estimate the state – dependent action advantage function [Wan+15].

**Temporal Difference – TD($\lambda$) learning**

The TD($\lambda$) learning algorithms are an advancement in the Q – learning or SARSA algorithms. The TD target looks into n further steps into the future to make the predictions. The Q – learning or SARSA algorithms also are a part of the TD($\lambda$) learning algorithms with the parameter $\lambda = 0$. Thus, they predict the TD target value looking only one step in the future and making the prediction. With higher values of $\lambda$, the TD target value prediction is done by looking $\lambda$ steps in the future. The equation below explains the concept for the return value for a TD(3) i.e. $\lambda = 3$ algorithm.

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 V(S_{t+4})$$

Thus, the Monte Carlo class of algorithms can also be said to be a part of the TD($\lambda$) with the parameter $\lambda = \infty$ i.e. the TD target prediction is carried after looking into $\infty$ steps in the future or at the end of the episode – the Monte Carlo algorithm. These methods are known as bootstrapping $\lambda$ methods spanning from the TD(0) to Monte Carlo control wherein the TD(0) methods aim towards shallow backups or instantaneous look ups and the n – step i.e. TD($n$) methods and Monte Carlo control aim towards deep backups [Sil15]. This class of algorithms also has several variations depending upon the manner in which the estimations of the TD target are made i.e. differentiations in the weighted sum and the weights in the final return.

#### 1.5.2.4 Genetic Algorithms

Genetic algorithms belong to the family of AI algorithms that have been inspired by biological evolution that are used for finding solutions for optimization problems (Maximization or minimization) but do not guarantee the global optimal solution.

They belong to the actor – only class of algorithms thus do not use information of a value function and are a model – free approach that can be used within continuous or discrete state and action spaces. They require fitness function that evaluates the solutions and the solution is represented as a string of chromosomes i.e. for e.g. an array of bits. It stores a pool of candidate solutions known as a generation. Iterations upon the previous generation yield the next generation that has the candidate solutions with higher fitness values and this is repeated until a target goal fitness value is reached for a pre – determined number of generations. The members or subset with the highest fitness values are chosen as the parents for the next generation. Merging these parents generates us new offsprings and this is known as crossover. The children can have features that have not been inherited from the parents and this is known as mutation [Mednd].

In context of our system of systems level intersection controller, the application of genetic algorithms would involve selection of a pre – determined number of random solutions and then iterating through say 15 generations by doing selection, crossover and mutation. After the elapse of the defined generation iterations, we may have a trained agent. However, selection of the solutions is a task in itself, since, the state space is too large for the simplistic single lane four – way model (around 65 million states). Determining these string of solutions i.e. actions for each participant of the system for an entire episode is not possible for ensuring the best possible optimization.

#### 1.5.2.5 Deep Deterministic Policy Gradient

The deep deterministic policy gradient is quite a recent work in the field of reinforcement learning by Google DeepMind. The main motivation behind the development of the algorithm was to tackle reinforcement learning problems with continuous action spaces. The DDPG is a policy gradient actor – critic algorithm that is off – policy and also model – free.

The policy gradient methods were devised to assume a stochastic policy $\mu\left(a \mid s\right)$ which gave a probability distribution over actions. Fundamentally, the algorithm increases probability of good actions by observing a huge number of training examples of high rewards as a result of good actions and those with negative rewards as a result of bad actions. This algorithm belonging to the actor – critic class is used to represent the policy function (known as actor) independently of the value function (referred to as the critic). Given the current state of the environment, the actor gives out an action and the critic produces a TD error corresponding to the state and the rewards received [Lil+15]. Basically, the output of the critic steers the learning process in both the actor as well as the critic.

The DDPG algorithm makes use of a stochastic behavior policy for ensuring descent amount of exploration but estimates a deterministic target policy that is much easier to learn. The policy gradient algorithms evaluate a policy and then follow the

policy gradient to maximize the performance. The actor and the critic structures are generally neural networks and predict the action in a particular state and generate a TD error signal at each transition. The main task of the critic network is to estimate the action value function or the Q – value of the current state and the action outputted by the actor. The updates of the actor network are executed in accordance with the deterministic policy gradient [Sil+14]. The critic network updates itself from the gradients observed from the TD error signal.

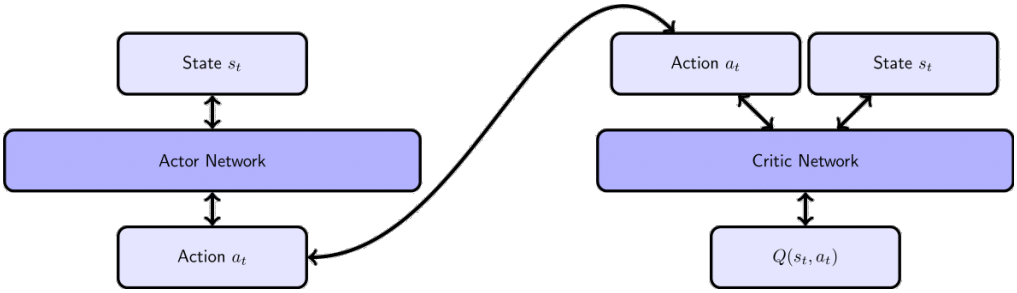


FIGURE 1.14: Methodology of the deep deterministic policy gradient algorithm

One of the primary advantages of this approach lies in the accurate calculations of the gradients. In actor – only algorithms, this is performed by slightly changing the policy and a subsequent evaluation based on the limited environmental processes. This evaluation may not prove sufficient and may affect the overall improvement of the policy adversely. By using an adaptive critic, the approximation of the gradient becomes much more accurate.

One of the primary issues with DQNs is that they can handle only discrete and low dimensional action spaces even though they can solve problems with high dimensional state spaces. DQNs cannot be used in context of continuous high dimensional action spaces since they find the action that maximizes the action value function, which is optimized iteratively at each transition. The discretization of the action spaces in such cases may not always be an option due to the increase in the action space exponentially [Lil+15]. The execution of the optimization process might also be too slow with large action spaces. And hence, the DDPG algorithms were designed to suffice these needs within reinforcement learning use cases.

With reference to our project of cooperative intersection control using reinforcement learning, our action space consisting is discrete and consists of finite number of actions that are the choice of the lane number from which the vehicle must be released. With absence of a continuous and too large action space, the application of such a complex algorithm such as DDPG is not justified. Due to the limited action and state spaces and thus subsequently limited data, it may not even stabilize. It may be used in context of traffic control applications concerning to the entire city or a region. As seen in the section of literature survey and prior work, DDPG has been used for the optimization of the traffic light timings for an entire region in order to ensure high levels of efficiency within the traffic flow [Cas+17].

# Chapter 2

# Related Work and Classical Intersection Management

This section aims to put forward the current intersection management strategies and intersection assistance systems in use. The automobile industry has taken huge technological strides and expedited the widespread use of cars by common people. The global automotive census reported almost 1.2 billion vehicles on the road currently with the figure on the rise each day. It has been predicted by leading strategist across the globe that this figure could hit the 2 billion mark by 2035. The global automotive profit predictions state that by 2020, a steep rise in the OEM profits would be seen hitting the 79 billion EUR mark i.e. almost a 50 % increase [Moh+13]. The grave issue encountered by traffic engineers is the lack of space and limited public funds for developing new traffic infrastructure in order to tackle the traffic management issues.

The conventional traffic light systems have been characterized by fixed programs also known as pre – timed control. The sequences of the red, yellow and green phases have fixed durations and operate in a pre – determined cycle in conjunction with all the traffic signals at a junction. With increasing traffic situations, this leads to long waiting times for a few vehicles on one side of the crossing even in the absence of vehicles on the opposite side. This eventually leads to worsening in the situations pertaining to greenhouse gas emission and increase in the total travel time. Various projects have estimated an average 2 % increase in the fuel consumption due to congestion. Predictions have also projected a potential increase in this figure to almost 4.2 % by 2050 [WML16]. For past few decades, many researchers and engineers have tried to approach the grave issue of traffic management in several ways. Majority of the work done in this context has been in the field of adaptive traffic lights that change their phases in coordination with the traffic situation at the intersection. Such adaptive traffic lights often make use of traffic detectors commonly known as induction loops to monitor the traffic situation efficiently. Traffic engineers found this to be one of the inexpensive solutions to alleviate the issue by making use of the existing infrastructure for optimization and modifications in the traffic signal control algorithms. Several projects have come up within the framework of development such as one by the Horizon 2020 by the European Union under their project of smart, green and integrated transport [Eurnd].

The major goal of the ATSC or Adaptive Traffic Signal Control has been to optimize the traffic flows for the highest level of efficiency, maintaining the safety levels of the conventional systems. The complexity of the optimization has led researchers to use diverse approaches to ensure an optimal solution. These approaches include Artificial Intelligence (AI) methods such as Fuzzy Logic, Neural Networks, Genetic

Algorithms, Reinforcement Learning, etc. The ultimate goal of such systems is to increase their contribution to the Intelligent Transport Systems (ITS) of the future.

Several intersection assistance systems have been conceptualized to assist the driver in the task of maneuvering intersections. The main reason behind the establishment of such systems was to reduce the number of mishaps occurring at intersections and to keep the driver away from the distractions while maneuvering intersections since the complex infotainment systems are gaining high importance today in the modern day vehicle cockpits. One of the solutions to counter these distractions and cognitive overload has been envisioned in the field of Human Machine Interface (HMI). Since the visual channels have been already used up through the other driver assistance systems, the auditory channel has been thought of one of the viable solutions [Hec+17]. The work 'Development of a personalized intersection assistant' provides insights into such proposed intersection assistance systems developed by IPG Automotive that are speech – based systems which warn the driver about the whereabouts of the vehicles approaching from the other directions [Hec+17]. Companies researching in the field of mobility such as dSPACE are validating such intersection assistance systems for enhancing safety while maneuvering intersections [dSPnd]. These systems include scanning the region around the vehicle which may not be cognized by the human driver and taking up necessary control such as warning or emergency braking. Other systems include situation assessment at intersections regarding the traffic situations and planning the trajectory of the vehicle or suggesting alternative lanes to the onboard navigation system. The turning assistant developed by Daimler for its commercial vehicles makes it possible to avoid accidents involving pedestrians and cyclists. Through a person recognition function, the systems primarily protect the most vulnerable road users [Daind]. The turn assist systems developed by Continental supports the driver in turning situations where oncoming traffic has to be considered and it warns the driver or brakes automatically [Connd].

## 2.1 Intersection Management in the context of autonomous vehicles

With the advent of ITS, multiple research projects are being carried out across the globe to ensure smooth and safe traffic flows. Intersection management is one of the challenges in maintaining a smooth and safe traffic flow. This section introduces them and illustrates a few concepts that can be used in the field of intersection management for autonomous vehicles. Even though the course of an intersection maneuver is relatively a very small part of the entire trajectory a car traverses, however majority of the disruptions in the traffic flow are caused by intersections and their inefficient management. According to CARE – Community road accident database from the EU, during the last decade i.e. from 2001 – 2010, more than 20 % of the total fatalities were intersection related fatalities [Bro+13]. The National Highway Transportation Administration (NHTSA), an agency of the American government and part of the department of transportation provides a nationwide census related to motor vehicle fatalities and crashes under the name 'Fatality Analysis Reporting System' (FARS). According to it, around 40 % of the crashes and almost 21.5 % of the traffic fatalities were intersection related [NHTnd].

Majority of the traffic lights that control the flow of the traffic at intersections in developed countries are adapted to suite the situational requirement. However, not all of them are dynamically adapted to the real time traffic and inefficient operation of signal times can hamper the safety and efficiency of the entire system. Congestion versus safety of the vehicles has been a topic of interest for many traffic engineers. Some believe that congestion levels do not impact the safety of the transportation system whereas some believe that accident frequencies increase with congestion levels at intersections as well as motorways [MW10]. However, crashes at intersections with lower congestion levels and thereby higher average velocities result in severe fatalities due to head on crashes or the involvement of Vulnerable Road Users (VRUs) such as pedestrians.

Lately one of the upcoming area of research in the field of ITS has been of 'Cooperative Intersection Management' to confront the vital issues related to traffic safety and efficiency discussed above through the progress made in the field of wireless communications. With the rise of inter vehicular communication technologies such as V2V and vehicle – infrastructure communication technologies such as V2I, the cooperation between the participants of the intersection management system would be beneficial also for the conventional signalized intersections along with non – signalized intersections [CE16]. Based on the communication received from several vehicles approaching the intersection, the traffic light can adapt the traffic light switching phases for an optimized traffic flow. This optimization would be carried out by a virtual 'Agent' so to say that would be a part of the traffic light infrastructure. It can be a conventional optimization algorithm or a highly sophisticated neural network that drives the adaptive phase changes of the traffic light system [Cas17].

A non – signalized intersection does not have stop signs or a phase indicator. With such systems, the approaching vehicles convey their dynamic information such as speed, position, intended direction of travel to the other vehicles or an 'Agent' that supervises this intersection. The agent may comprise of conventional optimization algorithms or state of the art neural networks that output actions to be undertaken by the participating vehicles. These actions might be diverse depending upon the complexity of the optimization problem. Typically it may include the 'Stop' and 'Go' commands or 'Increase speed' or 'Decrease speed' commands. A combination of these actions may also be a viable option for ensuring higher levels of efficiency. The agent here would take up the role of a virtual traffic light that implicitly communicates the actions or decisions to the individual participants rather than explicit communication such as signalized intersections through traffic lights [CE16]. Thus, this cooperative intersection management would allow vehicles to maneuver the intersection without human intervention.

As mentioned in the section 1.2.1 on levels of automation in vehicles, such non – signalized intersection management systems should be ideally implemented only for autonomous vehicles since the failure to undertake the communicated action from the agent or the controller by the human driver might result in complete disruption of the system. For e.g. if the participating vehicle has been informed to maintain a level of speed and maneuver the intersection, the execution responsibility within a pre – determined time frame would rest with the human driver of the system. A failure on his side would result in interruption of the working of the system and hamper the efficiency of the entire system. In critical situations or situations pertaining to highest level of optimization where the vehicles have been provided

with a very narrow time slot to maneuver the intersection, it might even result in severe fatalities. The figure 2.1 gives an overview of the different approaches with which a cooperative intersection management system can be realized.

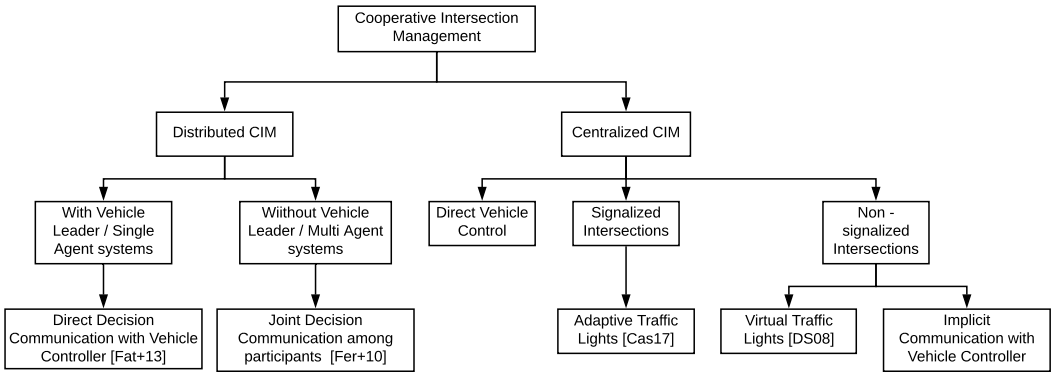


FIGURE 2.1: Classification Tree for approaches in realizing CIM

This thesis is driven by the efficiency gains such systems would offer. It aims at attempting to solve this optimization problem for the 'Agent' using the state of the art Artificial Intelligence (AI) techniques. It aims to develop a system of systems level control for driving such optimizations at the intersection. The infrastructure mounted 'Agent' or the master receives information through a suitable communication channel about the participating systems i.e. the vehicles approaching the intersection. Based on this information, it then controls the maneuver of each participating vehicle based on the strategy that it has learned during its training tenures using reinforcement learning in order to ensure a successful crossing of the intersection. This system can be thought of analogous to the generic master slave architecture where the slave raises a service request and the master responds to it issuing corresponding commands and suitable actions.

To ensure better levels of optimization along with maintenance of the safety levels would be a too complex conventional algorithm to program. Also, due to the random nature of traffic, ensuring optimization and efficient traffic flows against majority of the scenarios may not be realized using conventional algorithmic architectures. The modern day AI techniques such as machine learning provide us with the possibility to develop a robust and intelligent system to realize autonomous intersection negotiation by training it for thousands of hours against a diversity of scenarios.

## 2.2 Related Work

Intersections have been one of the bottle necks for the efficient movement of vehicles within a city. A lot of research has already been steered in this direction in order to improve the traffic flow in the context of intersections. This section aims to put forward the research that has been carried out in the field of intersection management using reinforcement learning, its drawbacks and which aspects of the work inspire this project on system of systems level intersection management using reinforcement learning. With the recent advancements in the communication technologies, V2X communication would soon be a reality and with this continuous real – time

information exchange between the moving vehicles, infrastructure and the control centers, a cooperative intelligent transport system can be realized.

An IEEE Transaction on intelligent transportation systems, volume 17, number 2 named as 'Cooperative Intersection Management: A Survey' serves as a highly comprehensive survey in this field for accelerating the advancement of automated and cooperative intersections. Issues pertaining to vehicles out – of – sight of the driver can be solved through joint optimization and cooperation between the vehicles and the infrastructure eventually leading to a fully cooperative intersection management (CIM) system resulting in improved traffic safety and efficiency [CE16]. Such systems can help the vehicles with a global view of the intersection for better decision making. The suggested propositions involve modeling of the intersections as a space and time discrete resource allocation and optimization problem. In such a state space, the geographical area or space and time slots would be allocated to moving vehicles with an underlying purpose of maximizing the efficiency [CE16]. Modeling of the vehicular trajectories for optimizing the vehicle control parameters such as velocity, acceleration / deceleration and collision region modeling in order to ensure safe resource reservation are the extensions of the space and time allocation model.

The work surveys two approaches – a centralized CIM where the coordination unit collects information relevant to the movement of the vehicles and makes central decisions or a distributed CIM where vehicles communicate with each other and form a VANET and decisions regarding maneuvering the intersections are made locally. The work focuses on several ongoing projects all across the world in the field of CIM along with several concepts such as centralized cooperative resource reservation and centralized cooperative resource reservation with an economic incentive. The later allows the vehicles to trade reserved resources by paying a price for maneuvering the intersection for emergency situations such as users rushing to the airport. The figure 2.2 below indicates this sequence diagram for communications between the vehicle agent (VA) and the reservation agent (RA) for the former approach and the communication between the vehicle agents and the exchange agent (EA) for a time slot exchange (TSE) [CE16]. This work proves to be a primary starting point for our project to get an overview of the concept of cooperative intersection management, already carried out research in this field and a few proposed ideas in order to realize such CIM systems.

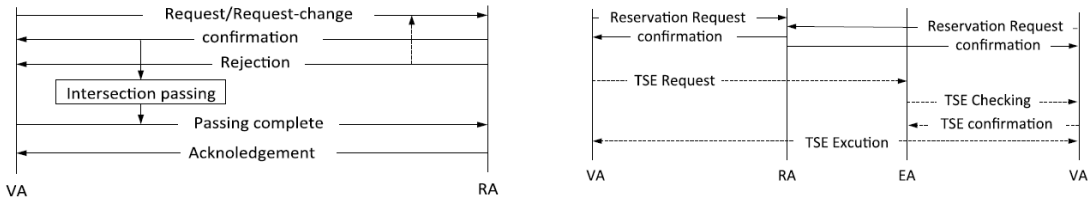


FIGURE 2.2: Sequence diagram for communication between vehicle and agent for a CIM [CE16]

The main focus of our research is in the field of centralized CIM and trying to optimize it for non – signalized intersections using reinforcement learning. Majority of the work in this field has been done in optimization of traffic light timings since

the conventional fixed – cycle traffic lights result in large waiting times for vehicles at the junction even when there are no vehicles passing on the opposite side of the junction. Such optimization problems can be one of the applications of reinforcement learning.

One of the earliest approaches to the field of traffic control using reinforcement learning has been carried out by Thorpe and Anderson [TA96]. The state of the system has been characterized by the number and position of vehicles in the north, south, east and west lanes approaching the junction. The actions consist of either allowing the vehicles on the north – south route to pass or the vehicles along the east – west route to pass. The SARSA algorithm in reinforcement learning has been used to optimize the traffic light control time [TA96]. The agent in the context of this project deals with limited complexity by handling traffic only in two possible directions and also takes into account only the number of vehicles waiting in each direction. Hence it ultimately regulates a system level intersection control by taking into account only the number of participants in each direction of travel.

The common four – way junction with single lanes on all routes has been formalized as a MDP in the work 'Application of Markov Decision Processes to the Control of a Traffic Intersection' [Min09]. The state space has been characterized by the number of vehicles waiting along the north – south and the east – west routes along with the current state of the traffic light. The action space consists of two actions namely 'Stay the same' or 'Change'. The intersection problem is modeled as a MDP and the performance is checked for three different strategies – the conventional fixed time cycle where the phase changes every 30 seconds, the adaptive mechanism where the phase changes when the number of vehicles waiting in the opposite lane are twice in the lane from which the vehicles are being released and the reinforcement learning strategy using value iteration algorithm. The reinforcement learning strategy outperforms the fixed – cycle one, but fails to match the adaptive mechanism [Min09]. This work again deals with limited complexity by accounting only for two directions of travel and no turning involved. Also, it attempts to optimize traffic flow by training the agent to regulate the phase changes for a traffic light. Similarly [LLW16] studied the application of deep reinforcement learning to optimize traffic lights at a single cross – shaped intersection with two – lanes per direction and no turning allowed (like in the former work). However, these works do not involve the training of an agent through collisions to manage an intersection with a holistic perspective. More or less, they are dependent on the number of vehicles waiting or the traffic demand in each direction.

One of the extensive works in the field of traffic light control using reinforcement learning is 'Deep Deterministic Policy Gradient for Urban Traffic Light Control' [Cas17]. With fixed pre – timed traffic light controls, the duration of the phases i.e. red, yellow and green are fixed and leave no scope for optimization. Adaptive traffic lights change their phase in accordance with the real – time traffic demand in order to ensure minimum waiting times at the junctions and increasing the traffic flow efficiencies. Such adaptive lights detect the real – time traffic distribution using traffic detectors. However, one of the major drawbacks of such systems is that they take into account only the traffic situation local to a single junction [Cas17].

This work tries to create coordination between the traffic light cycles within an entire region or over a large expanse of area in order to ensure higher level of traffic

flow efficiencies. One of the major reasons for lack of work in this research area was the need of an algorithm that can be used to optimize the traffic light cycle durations with a holistic view due to the poor scalabilities in case of the earlier algorithms. With advancements in the field of deep Q – networks and subsequent successes in the approach, deep deterministic policy gradient capable of handling very large and continuous state and action spaces, this traffic light optimization problem with a holistic view was attempted to be solved [Cas17]. The state space was characterized with the metric 'speed score' for each detector where speed score was the minimum of 1.0 and average speed / maximum speed. The action space consisted of phase durations for all the traffic lights under control [Cas17]. This work was a motivation for our project in context of development of a system of systems level intersection management, with the difference that the above stated work develops a system that regulates traffic lights for different intersections (participating systems) within a certain area, whereas our project develops a system that concerns with the regulation of intersection maneuvers for different vehicles or participating systems directly and not implicitly using traffic lights.

All the above works focus on more or less optimization of the traffic lights using reinforcement learning in order to enhance the traffic flow efficiencies. Our project work revolves around the concept of non – signalized intersections or virtual traffic lights (only a 'Stop' or a 'Go' command issued individually to each participant of the system) based intersection management for autonomous vehicles in order to achieve better flow efficiencies. One of the works slightly inclined in this direction is 'Flow: Architecture and Benchmarking for Reinforcement Learning in Traffic Control' [Wu+17]. The 'Flow' framework is created to bridge the gap between machine learning and complex traffic control problems. Flow framework has been used to control a vehicle simulation for different traffic scenarios using the SUMO simulator and a reinforcement library 'rllab'. The road networks used within the scope of this work include roundabouts, figure – of – eight loops and cross intersections. The main difference within the scope of this work is that the reinforcement learning model is trained to control the lateral and longitudinal behavior of vehicles and not necessarily intersection control. Also, the training episodes are terminated in case collisions are encountered [Wu+17]. However, in case of our project, the agent learns an effective management strategy through the collisions that occur.

The only work that quite closely resembles our project is 'Navigating Occluded Intersections with Autonomous Vehicles using Deep Reinforcement Learning'. It focuses on navigating non – signalized intersections using reinforcement learning [Ise+17]. However, the main difference lies in the fact that this work revolves around laying an optimal strategy for navigating an intersection only for an ego vehicle after taking into consideration the surrounding traffic scenarios. The action space consists of either a wait or a go command and sequential actions such as acceleration or deceleration only in the context of the ego vehicle. Since the topic revolves around the control of a single vehicle, the state space characterization includes discretization of the area around the ego vehicle in the form of grids through Cartesian coordinates. The intersection scenarios realized during the experimentation include taking a right turn at an intersection, taking a left turn at an intersection and going straight through an intersection. The authors make use of deep reinforcement learning using DQNs in order to train the ego vehicle controller to navigate the intersections without colliding with the surrounding traffic. The metrics that have been calculated

include time to collision (TTC), average braking time, average time taken to maneuver the intersection and the number of collisions occurring during the movement. The randomness of the traffic has been simulated by varying the traffic distribution [Ise+17]. Our project attempts to train the agent based on similar metrics, but not only concerning the ego – vehicle, but all the participants at an intersection delivering holistic management. The above mentioned project also aborts the episode during the training in case collisions have been encountered unlike our approach, where the collisions or mishaps train the agent to map states in which particular actions are to be avoided.

# Chapter 3

# The proposed solution

## 3.1 SUMO – Simulation of Urban Mobility

A suitable traffic simulator is required for the fruition of our concept of cooperative intersection control using reinforcement learning. The training of the agent in order to learn a policy using reinforcement learning to control an intersection needs a suitable environment, which communicates the input state space to the agent i.e. information pertaining to the vehicles and renders a feedback to the agent for the actions it has taken. This section elucidates the viable options, the selection of the simulator and justification for this selection.

The traffic simulators can be divided into three broad classes – microscopic, macroscopic and mesoscopic simulators [Cas17]. The microscopic simulators compute the positions of individual vehicles at every simulation step. The dynamics of the vehicles are also governed by pre – defined models which drive the behavior under different conditions. In contrast, macroscopic simulators work with a holistic view, managing the traffic conditions like in a flow network in fluid dynamics. Mesoscopic simulators refer to the variations between microscopic and macroscopic simulators. In context of our project, in order to control the intersection behavior of vehicles, we need information pertaining to the individual vehicles and thus we focus on micro simulators. Quadstone Paramics, VISSIM, AIMSUN, MATSIM, POLARIS and SUMO are the various commonly used traffic micro simulators. The former ones are commercial softwares whereas the latter two are open source software capable of handling large – scale traffic simulations. SUMO was started as an open project in the year 2001 [Kra+12]. It stands for Simulation of Urban Mobility and is mainly developed by employees of the Deutsches Zentrum für Luft und Raumfahrt (DLR).

SUMO is very widely used in the research community. It is an extensible simulator that discretizes time and advances the simulation for a user – defined timestep. The central idea behind its development was to provide the traffic research community with a common platform to test and compare traffic light optimization problems. Since SUMO was constructed as a basic package, it can be modified and extended according to the needs of the user. SUMO was initially designed to be highly portable and run the simulations as fast as possible and hence the initial versions could be run only from the command line without a graphical interface. The GUISIM was later added as a component for simulation with a graphical user interface. It has several other components such as NETGEN, NETEDIT, NETCONVERT, etc. and this allows for easy extension of the applications. SUMO is implemented in C++ and uses portable libraries [DLRndb]. SUMO includes a Python API called

TraCI (Traffic Control Interface). TraCI provides the user with the extension functionality through querying and modifying the state of the simulation at each time – step. This gives the user a direct access to the simulation in order to control its behavior. The API also makes retrieval of information about the vehicles running in the simulation and sets various parameters by issuing precise commands possible for the user. TraCI uses a TCP (Transmission Control Protocol) based client – server architecture in order to provide access to SUMO wherein SUMO acts as the server and the Python script as the client. The majority of the machine learning community across the world uses Python as one of the most preferred programming language for implementation of machine learning methods for a diversity of applications. In this context, several pre – written machine learning packages and frameworks can be simply imported and directly used in Python. The Python program together with the TraCI toolbox can be developed to serve as an interface between the SUMO simulation and reinforcement learning. Hence the SUMO simulator along with its TraCI toolbox is a promising option in demonstrating this concept of cooperative intersection control using reinforcement learning.

Creation of a simulation in SUMO consists of a series of steps. The first step is the generation of a network that can be done using existing .xml files and adapting them to pass the requirements or by manually creating a network using the NETGEN module. In both the approaches, the nodes (.nod.xml), the edges (.edg.xml), the connections (.con.xml) have to be specified either in the files or explicitly during the manual modelling. A configuration file (.netccfg) along with these files then creates a network file (.net.xml). The route file (.rou.xml) that specifies the paths to be followed by the vehicles along with the vehicle specifications can be written either directly as a .xml file or can be written through the Python TraCI API so as to generate it real – time. The later approach has been defined in the subsequent sections. The network file (.net.xml), route file (.rou.xml) and the additional files (if any) (.additionals.xml) that may consist of information pertaining to the detectors, etc. together with the SUMO configuration file (.sumocfg) finally create a SUMO simulation. For simulations using TraCI, the Python script connects with the SUMO server using a specific port. A route generation process is carried out using the Python script using a loop function that terminates after a pre – defined number of steps in order to generate the vehicles and update the route file. The advance of a simulation step can also be issued through the Python script as a command and a simulation loop is established. After the simulation expires, the communication with the SUMO server via the port is closed.

One of the biggest advantages of the SUMO simulator is that it is frequently updated by DLR according to the modern day traffic infrastructure enhancements. It is functionally extended by an active research community as well as by its developers at DLR. The official GitHub repository for the SUMO simulator is maintained by the developers who are extremely responsive to raised issues and questions pertaining to the simulator. Adaptive traffic lights switch phases in correspondence to the real – time traffic situation have been already implemented in a few metropolitan cities across the world. Such systems use traffic detectors that are generally embedded within the road and assess the passing vehicles and report it to the traffic light control system. There are generally three types of detectors offered by SUMO – E1 detector or Induction Loops, E2 detector or lane – based detector and E3 detector or multi – origin / multi – destination detector [Min09]. The E1 detector or commonly known as induction loop is actually a slice plane that is embedded through a road

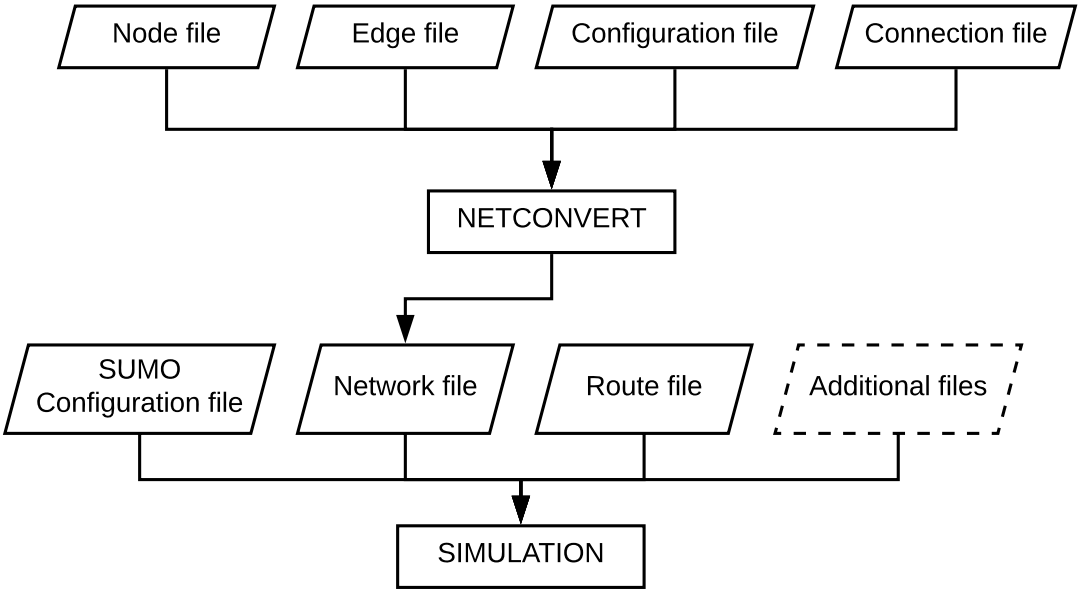


FIGURE 3.1: Flowchart for simulation pre – processing in SUMO

lane and assesses the information in context of vehicles passing over it. The E2 detector or lane – based detector is a sensor along the complete lane or a part of the lane that asses only the vehicle currently on it. The multi – origin / multi – destination detector or E3 detector monitors vehicles moving between a set of entry points and a set of exit points. These detectors are essential in context of our project to communicate the vehicle presence in a certain lane along the route to the infrastructure mounted agent or management system.

## 3.2 Reinforcement learning based system of systems intersection management

In reinforcement learning, the agent is not explicitly instructed about the actions it must take, but must look out for actions so as to eventually yield a maximum reward [SB17]. Reinforcement learning offers two primary advantages from the system level that make it one of the most suitable machine learning methods in the context of cooperative intersection control for autonomous vehicles – the approach can be realized as a model free development method and it delivers high robustness against unseen scenarios during its entire training tenures. Supervised learning cannot be used in the context of this project, since it requires test data to be generated. This test data generation would in turn require an existing algorithm for efficient control of the intersection and then a neural network can be trained with this training data using supervised learning. However, the trained network would only be as good as the existing algorithm itself and arriving up to such an efficient algorithm for every type of intersection and for every traffic distribution is not a viable option. Using reinforcement learning, we let the system itself arrive upto an efficient algorithm that we cannot come up ourselves. It is almost impossible to simulate every single scenario that would be encountered in reality for every type of intersection within the simulator. A traffic distribution function or model can only approximate the actual traffic and never entirely represent every encountered scenario. Hence, the agent

that regulates vehicle flows at intersections should offer high robustness against scenarios it may not have been trained for by at least trying to approximate the control strategy. Supervised learning offers limited capabilities of predicting the outputs against the inputs it hasnt been trained for. The reinforcement learning algorithms, because of an overall combination of implicit and explicit learning eventually tend to impart natural intelligence to the system that may be capable of handling unseen or untrained scenarios. As seen in the literature section, SL for detection and RL is often used for control problems.

The SUMO simulator has been used in the context of this thesis to demonstrate the concept of cooperative intersection control for autonomous vehicles using reinforcement learning. The SUMO simulator has been chosen as a training environment for the reinforcement learning based intersection management system. Several environments have been developed that aid in the research of reinforcement learning. A bachelor thesis had been previously carried out at the chair of software engineering: Dependability at the Technische Universität Kaiserslautern, with an aim to examine existing simulation environments with regard to their usage in reinforcement learning for autonomous vehicles [Ger18]. The environments such as Udacity – self driving car, Microsoft AirSim, TORCS and CARLA have been evaluated and analyzed based on the fulfillment of a few requirements that had been laid before such as retrieval of diverse sensor data for evaluation of ego vehicle state. However, these environments have been developed concerning control of only the ego vehicle. A holistic or system level control over several participating vehicles at an intersection cannot be implemented in these environments and hence they are inappropriate for our project based on system of systems level intersection control. The other reinforcement learning environments at our disposal such as 'Gym' from open AI deals with highly preliminary control problems using reinforcement learning and cannot be used for our project. Within these environments that have been specifically developed for promoting reinforcement learning, upon execution of a single simulation step, the next state that has been reached by the agent and the subsequent reward for reaching that particular state are received as a feedback from the environment directly. However, the SUMO environment has not been primarily developed for such a purpose and hence provides the possibility of receiving such feedbacks but through explicit requests using the TraCI Python API.

This project develops this robust interface between the SUMO simulator and reinforcement learning in context of intersection control via the TraCI Python API. Since no prior research has been found in this area, the initial concept development has been demonstrated through a highly fundamental intersection model with a simplistic state space characterization. During the later stages, the state space characterization goes through two more iterations in order to improve the performance of the reinforcement learning agent. So this section has been split into four main approaches as per the complexity of the state space characterization and for a real – life scenario as per the table 3.1.

### 3.2.1 Reinforcement learning in context of a simplistic single lane model

The simplistic model is a single lane four – way intersection model with a highly basic state space representation. This section 3.2.1 puts forward this simplistic single lane four – way model and its basic simulator setup, the basic algorithm structure and functionalities, the state space characterization, the results and the limitations

TABLE 3.1: Different iterations in context of system of systems level intersection management based on RL

| Intersection Model | State – space characterization | Section |
|---|---|---|
| Single lane four – way | Simplistic | 3.2.1 |
| Single lane four – way | Split state | 3.2.2 |
| Single lane four – way | Advanced | 3.2.3 |
| Triple lane four – way | Advanced | 3.2.4 |

of the chosen state space characterization. The main purpose of this model was to demonstrate the frictionless execution of the reinforcement learning algorithms in context of cooperative intersection management and resolve multiple simulator related errors. It also aids in detailed understanding of the simulator framework and its inherent nature.

As described in the above sections, the main aim of our project is to optimize and evaluate the performance of reinforcement learning in context of traffic intersections. A junction consisting of intersection of two roads has been modeled in SUMO. Each road is composed of one lane for each direction of travel. The agent or the controller in our case is our Python script that communicates with the SUMO simulator through a client – server architecture. In reality, it can be assumed that this agent would be a part of the traffic junction infrastructure and each incoming and outgoing vehicle would communicate with this agent aiding in a system of systems level control for the intersection. In case of our experiments, this communication to the agent regarding the incoming and outgoing vehicles has been established using detectors placed along the incoming and outgoing lanes in each direction. The detectors on the incoming lanes are placed a finite distance away from the junction to allow the agent to take appropriate actions on the approaching vehicles. The detectors on the outgoing lanes are placed very close to the junction along the outgoing lane to aid in speedy communication regarding the successful maneuver of a vehicle across the junction.

#### 3.2.1.1 Simulator setup for the simplistic single lane model

This section explains the setup of the simulator and the subsequent settings for simulating collisions in SUMO along with the basic functionalities required to start the simulation. The figure 3.2 shows the simplistic scenario of a single lane four – way intersection with induction loop type detectors for incoming and outgoing vehicles along the lanes. The green phases for all lanes across the junction have been explicitly set and have been explained subsequently. The XML files for modeling such an intersection scenario along with the SUMO configuration file and additional files can be found on the Github repository for this project on intersection control under the following URL –

https://github.com/rgugale/Cooperative-Intersection-Control-using-Reinforcement-Learning.git

The entire system runs with a discrete time step of 1 second, i.e. the agent outputs control actions at a time interval of 1 second. The selection of the time step of 1

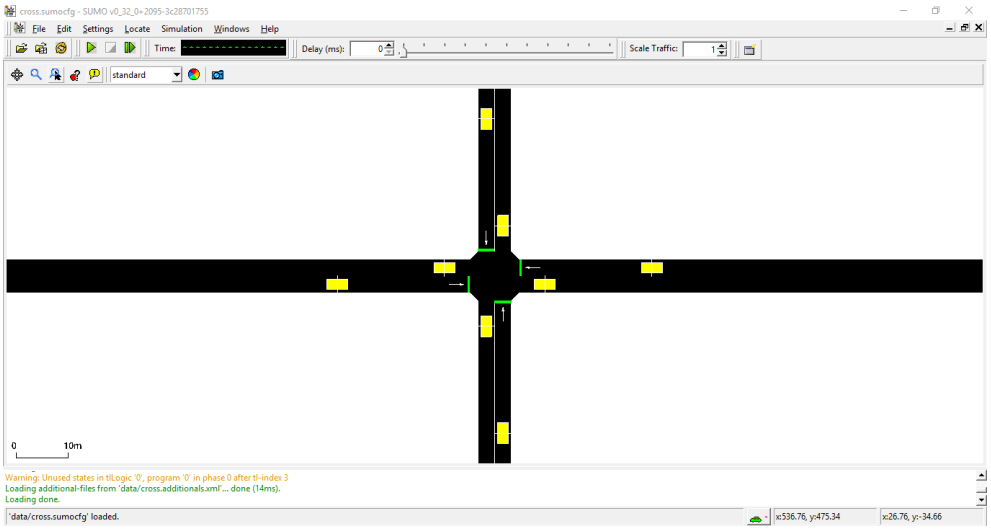


FIGURE 3.2: Single lane four – way intersection model in SUMO

second has been justified in the chapter 4 of the document. The performance evaluation of reinforcement learning in the context of intersection control without traffic lights can be performed by assigning rewards in case of intersection maneuvers by the vehicles and negative rewards or punishments in case of vehicular collisions. During the initial training phases when the agent has no knowledge about the environment and a rudimentary policy that is under development, collisions are ought to occur and a negative reward for the actions causing the collisions helps the agent or the controller comprehend that it should figure out a policy that avoids these actions. However, since the inherent behavior of SUMO is to avoid collisions, a few important settings have to be performed in context of the set of network XML files and the SUMO configuration file.

The connection file defines or specifies which incoming edge to the junction and which corresponding outgoing edge from the junction are along the flow route of the vehicles. In simple words, it links incoming lane with the outgoing lane through a junction. Along with the normal attributes such as such as 'from' and 'to', an additional attribute has to be appended to impart the capability of monitoring junction collisions: 'pass = true'. Also, the collisions occurring at the junction cannot be monitored if the junction node is of the type – unregulated or traffic_light_unregulated. The junction node type in the node file of the network must be explicitly declared as 'traffic_light' to be able to monitor junction collisions. Since the inherent nature of the SUMO simulator is to avoid collisions, it implicitly brakes or stops the vehicles when possible, in order to avoid a collision. However, this can disrupt the complete functioning of our reinforcement learning approach, since we are training our model based on the number of successful intersection maneuvers and collisions occurring. In order to avoid such an implicit behavior of the simulator, the speed of the vehicle has to be set using the 'Set Speed Mode' and 'Set Speed' methods defined in the 'traci.vehicle' class.

These methods violate the internal vehicle dynamics model of SUMO and have been explained in detail under the section of 'Modifications of the SUMO Simulator' i.e. chapter 4. However, in order to simulate the collisions, these methods have to be

called during the execution of the simulation using TraCI.

```
traci.vehicle.setSpeedMode(vehicles_waiting_1[0], 0)
traci.vehicle.setSpeed(vehicles_waiting_1[0], 10)
```

After the required files for the network have been written appropriately, the NETCONVERT module is used to generate a network. Since the junction type has been explicitly declared as 'traffic_light', a default traffic light switching duration and phase logic is programmed in the 'net.xml' file. This default logic has to be changed to an 'All – Green' phase logic in order to simulate the collisions occurring at the junctions by controlling the vehicle maneuvers through the reinforcement learning model. This can be done by editing the 'duration' and 'state' attributes of the 'tlLogic' parameter of the generated 'net.xml' file. This setting results in a green phase for the entire duration of the simulation across the junction in all directions of travel.

The route file that defines the trajectories of the vehicles and their types is generated by a function 'generate_routefile( )' that has been declared and defined in the Python script. The declaration and definition for this function can be found in the Python script that controls the entire reinforcement learning intersection control approach. The reproducibility of the tests is an important factor for enabling debugging of the entire approach. Hence, the random number generator has been seeded to produce the same results during every execution. It is important to note that seeding the generator does not imply that the agent is trained only against certain scenarios. The traffic density is varied for each episode using the 'random.randint' function within a pre – determined range. However, seeding the generator ensures that this random number generation for certain defined number of episodes would be the same enabling in debugging the entire system in case of failures or errors. Thus, variations in the number of episodes train the agent on a multitude of scenarios. Upon execution of the Python script, the function 'generate_routefile( )' writes the route file for the entire simulation run. This function has been defined below for reference.

```
def generate_routefile():
    N = 5000 # Number of time steps
    # Demand per second from different directions changing per Epsiode
    traffic_density = [random.randint(-3, 3) for i in range(0, 4)]
    pWE = 1. / (8 - traffic_density[0])
    pEW = 1. / (8 - traffic_density[1])
    pNS = 1. / (8 - traffic_density[2])
    pSN = 1. / (8 - traffic_density[3])

    with open("data/cross.rou.xml", "w") as routes:
        print("""<routes>
        <vType id="typeWE" accel="0.4" decel="6.5" sigma="0.5" length="
                                                5"
        minGap="1.5" maxSpeed="10" guiShape="passenger"/>

        <vType id="typeNS" accel="0.4" decel="6.5" sigma="0.5" length="
                                                5"
        minGap="1.5" maxSpeed="10" guiShape="passenger"/>

        <route id="right" edges="51o 1i 2o 52i" />
        <route id="left" edges="52o 2i 1o 51i" />
        <route id ="up" edges="53o 3i 4o 54i" />
        <route id="down" edges="54o 4i 3o 53i" />""", file=routes)
        lastVeh = 0
        vehNr = 0
        for i in range(N):
```

```
            if random.uniform(0, 1) < pWE:
                print('<vehicle id="right_%i" type="typeWE"
                route="right" depart="%i"/>' % (vehNr, i), file=routes)
                vehNr += 1
                lastVeh = i
                continue

            if random.uniform(0, 1) < pEW:
                print('<vehicle id="left_%i" type="typeWE"
                route="left" depart="%i"/>' % (vehNr, i), file=routes)
                vehNr += 1
                lastVeh = i
                continue

            if random.uniform(0, 1) < pSN:
                print('<vehicle id="up_%i" type="typeNS" route="up"
                depart="%i" color="1,0,0"/>' % (vehNr, i), file=routes)
                vehNr += 1
                lastVeh = i
                continue

            if random.uniform(0, 1) < pNS:
                print('<vehicle id="down_%i" type="typeNS" route="down"
                depart="%i" color="1,0,0"/>' % (vehNr, i), file=routes)
                vehNr += 1
                lastVeh = i
                continue
        print("</routes>", file=routes)
```

This route file – 'rou.xml' generated prior to the simulation, the network file 'net.xml' and the other additional files are specified in the SUMO configuration file '.sumocfg' that ultimately outputs the simulation itself with or without a GUI (as per specification). A few other command line parsers have to be passed to monitor collisions occurring at the intersections. In our case, since SUMO GUI is invoked through the Python script itself, these command line parsers have to be programmed in the Python file. The main parsers in order to monitor junction collisions are '–collision.check-junctions' and '–collision.action' as 'warn'. The former monitors the collisions occurring at the junctions while the latter raises a warning on the SUMO console (default being crashing of the simulation). These command line parsers have been hard coded in the Python script as follows

```
traci.start([sumoBinary, "-c", "data/cross.sumocfg",
             "--tripinfo-output", "tripinfo.xml", "--additional-files",
             "data/cross.additionals.xml",
             "--collision.check-junctions",
             "--collision.mingap-factor", "1",
             "--collision.action", "warn",
             "--step-length", "1",
             "--error-log", "error.txt"])
```

#### 3.2.1.2 Characterization of the State Space for the simplistic single lane model

Since these are the first set of experiments, the characterization of the state space has been kept highly simplistic. This section elucidates this simplistic state space characterization. Each vehicle approaching the junction is by default requested to stop before the junction and given a 'Go' signal by the agent depending upon the strategy developed by the reinforcement learning model and these viable actions are a part of the action space. The states are characterized as a four element single row vector,

with each element either a '0' or a '1', either indicating the non – presence or presence of a vehicle respectively within a lane. The first element indicates a vehicle non – presence or presence in the 1st Lane, i.e. incoming vehicle from the left intended to travel to the right i.e. along the route west to east. The subsequent elements represent the subsequent clockwise lanes. For e.g. the state [0, 1, 1, 0] indicates that there is no vehicle in the left lane along the route left to right, there is a vehicle in the bottom lane along the route south to north and right lane along the route east to west and there is no vehicle in the top lane along the route north to south. The state space is retrieved by the agent using the detectors placed along the travel paths. The state space is actually the presence or non – presence of the vehicle between the incoming and the corresponding outgoing detector along each lane. Thus, with such a state space characterization, 16 different states are possible.

It is important to note here that a model free approach has been used for this experiment. Since the state space characterization is highly simplistic, a model with its transition probabilities into subsequent states after execution of particular actions can also be built. With a precise model of the entire system dynamics along with its transition probabilities, optimization techniques such as dynamic programming can also be used. However, the state space characterization has been kept simple only for the first few experiments. Later on, due to improper representation of the problem and optimization limitations with such a characterization, the state space representation has been made much more complicated and consists of integer values. Using non model – free approaches such as dynamic programming would not be feasible in such contexts due to millions of possible states.

The agent depending upon the current state can take 4 different actions namely – 'Lane1', 'Lane2', 'Lane3' or 'Lane4' i.e. release the vehicle from the leftmost lane i.e. along the route west to east (Lane 1) or the bottom lane i.e. along the route south to north (Lane 2) or the right lane i.e. along the route east to west (Lane 3) or the top lane i.e. along the route north to south (Lane 4). The Q – algorithm has been used to realize the entire intersection control problem using reinforcement learning. Since the characterization of state space is highly simplistic and the number of actions that can be performed by the agent are limited, a Q – table can be constructed using the 'Data Frame' provided by the Python 'pandas' library.

```
def build_q_table(n_states, actions): # n_states = no. of States
# actions = set of all viable actions
    table = pd.DataFrame(np.zeros((n_states, len(actions))), index=
                                        STATES, columns=actions)
    return table
```

The reward signal has been quantized with a value of '+1' for a crossing and a punishment or a negative reward signal approximately around '–1.5' is assigned when vehicles encounter a collision at the junction. The rewards and the punishment values have been determined after thorough experimentation and this has been explained in the subsequent section 3.2.4. The two broad classes of reinforcement learning algorithms – Monte Carlo and Temporal Difference algorithms have been explained in detail in the above sections. The usage of the popular Q – algorithm in context of our project on cooperative intersection control has also been justified in the above sections.

Since the characterization of the state space is kept highly simple for this experiment, a conventional Q – table can be generated with 16 rows as the different

states and 4 columns representing the 4 different viable actions in each state. The reinforcement learning algorithms are actually an amalgamation of implicit and explicit learning which is one of the biggest advantages of reinforcement learning in contrast with the other machine learning methods. Since the characterization of the state space is kept simple, the exploration versus exploitation dilemma of reinforcement learning is easy to tackle. Since each state can be visited during a simulation run, the exploration is set to almost zero. An Epsilon greedy policy has been pursued with the parameter Epsilon set as 0.99. Thus, the probability of choosing a random policy is almost negligible. Setting the Epsilon value to 0.99 and not 1 is a reinforcement learning practice to minimize the possibility of overfitting during validation or during training after Epsilon value reaches 0.99 [Cle18]. The code below describes this epsilon greedy policy for choosing an action.

```
def choose_action(state, q_table): # with EPSILON = 0.99 Greedy Policy
    state_actions = q_table.loc[state, :]
    if (np.random.uniform() > EPSILON) or ((state_actions == 0).all()):
        action_name = np.random.choice(ACTIONS)
    else:
        action_name = state_actions.argmax()
    return action_name
```

With the current characterization of the state space, the vehicles in each lane are characterized by their presence or non – presence between the section from the input detector to the output detector along the lane.

#### 3.2.1.3 Reinforcement learning with the SUMO Simulator

This section aims to put forward a few functionalities that are a part of the reinforcement learning interface with the SUMO simulator. The default lane follow model – 'Krauß' model in case of SUMO halts vehicles one behind the other with a finite distance between each vehicle [DLRndb]. Thus, it avoids the rear end collisions occurring with the vehicles within the same lane. In order to simulate collisions in SUMO, a few of its default models have to be violated and one of them is the car following model. Without this violation, SUMO implicitly brakes or stops the vehicles in order to avoid a collision and this is an undesirable behavior in context of our project. This behavior has been explained in further details in the chapter 4 of the document. Since we are violating the model in order to simulate collisions, the vehicles have to be explicitly made to halt one behind the other. Failure to do so results in rear end collisions, that would be monitored by the reinforcement learning model as its decision failures and hamper the overall learning of the model. The code below describes the logic to halt vehicles systematically before a junction one behind the other with a finite distance separating them.

```
for vehicle1 in add_vehicles_1:
# Vehicles detected by Lane1 Input Detector
    if vehicle1 not in vehicles_in_loop_1: # Vehicles_in_loop_1
        # consists of all vehicles between the input and output
        # detectors along the lane 1
        vehicles_in_loop_1.append(vehicle1)
        if vehicle1 not in vehicles_waiting_1: # Vehicles_waiting_1
            # consists of the vehicles waiting or approaching the
            # junction before the intersection for a 'Go' signal
            vehicles_waiting_1.append(vehicle1)
            if position1: # Loop to halt vehicles one behind other
                # at a finite distance
                i = len(position1)
```

```
                tmp_pos1 = position1[i-1]
                position1.append(tmp_pos1 - 3)
                traci.vehicle.setStop(vehicle1, "1i",
                position1[i])
            else:
                for i in range(0, len(vehicles_waiting_1)):
                    position1.append(495 - 3 * i)
                    traci.vehicle.setStop(vehicle1, "1i",
                    position1[i])
```

Similarly, after the first vehicle waiting before a junction has been released, SUMO explicitly moves the subsequent vehicles to the start of the junction, i.e. the vehicle behind the released vehicle moves forward and occupies its position. Thus, all the vehicles waiting within the lane move to occupy the previous position of the preceding vehicles. However, with the violation of the default lane follow model, this action has to be explicitly executed by the algorithm. Failure to do so results in the vehicles continuing to wait in their earlier positions, even when the vehicle ahead has been released. This causes the simulator to crash the simulation in case of new vehicles being added to the lane with an error 'distance too short to brake'. The code below describes how this functionality can be implemented within the algorithm.

```
if 495 not in position1: # The start of the junction is 495
    if vehicles_waiting_1:
    # For subsequent vehicles behind the released one
        pos1 = 495
        i1 = 0
        change_position1 = []
        for vehicle1 in vehicles_waiting_1:
            traci.vehicle.setStop(vehicle1, "1i", position1[i1], 0, 0)
            # Release the vehicle from its earlier position
            traci.vehicle.setStop(vehicle1, "1i", pos1)
            # Halt vehicle at previous position of preceding vehicle
            change_position1.append(pos1)
            pos1 -= 3
            i1 += 1
            position1 = change_position1
            # Change the position matrix for the vehicles
```

The Q – table gets updated in case of a reward or a punishment (in case of collisions) i.e. a feedback has been received due to the prior actions. In context of this project, a reward or a punishment is not received at every time step but has a sparse distribution. The Q – algorithm has been devised in such a way, that in case of no rewards (positive or negative), it updates the Q – table negligibly until convergence has been achieved. In case of a feedback, the Q – table is updated as per the algorithm for the action that has resulted in the reward. The algorithm has been described below.

```
q_predict = q_table.loc[str(S), A] # The original Q value
# in the table for the release action and the given state

if step == MAX_STEPS: # If last step of the simulation
    S1 = 'terminal'
else:
    S1 = 'nonterminal'

if S1 == 'nonterminal': # If not the last step of the simulation
    q_target = R + GAMMA * q_table.loc[str(S_), :].max()() # The
    # target Q value after the reward & the discounted
    # maximum future state value
else:
```

```
    q_target = R # Target Q Value in case of last simulation step

q_table.loc[str(S), A] += ALPHA * (q_target - q_predict) # Update in
# the Q table for the unsafe release action in the given state
```

In case of a collision, the vehicle that has been released at a later time stamp should be penalized for causing the collision. However, in order to determine the vehicle that has been released at a later time stamp and the corresponding release action selection by the reinforcement learning model that resulted in the collision can be arrived upon through a logical deduction. The list of the colliding vehicles can be retrieved using the 'getCollidingVehicleIDList( )' method. The vehicle that caused the collision and the corresponding release action that caused this collision can be deduced using 'regular expressions' in Python. This algorithm to retrieve the action that resulted in the collisions has been demonstrated in the section 3.2.3.2. concerning the single lane four – way model with advanced state space characterization.

#### 3.2.1.4 Results for the simplistic single lane model

This section aims to present the results achieved through the simplistic single lane four – way model. After tuning of several other parameters and required settings within the simulator, simulation runs have been executed and results are derived. One of the major hyper parameter 'Epsilon' has not been changed throughout all the simulation runs since each state can be easily explored due to the limited diversity of the states. Several iterations are carried out for varying number of simulation steps and varying number of episodes. Since the number of diverse states are limited, the Q – values in the Q – table converge around 6000 simulation steps. The learned action – value function Q directly approximates q*, the optimal action – value and enables early convergence proofs. Under the assumption that all state – action pairs are visited and updated, Q has been shown to converge with a probability 1 to q* [SB17]. The simulation steps are divided in 3 episodes, so that with the 'generate_routefile( )' function, traffic at different distributions can be generated in order to simulate randomness. The learning rate and the discount factor are also iterated in order to derive the best possible results. After running a few simulations, it has been realized that the characterization of the state space has to be enhanced due to certain limitations that have been illustrated in the next section. The figure 3.3 indicates the course of the training process as it attempts to approximate the policy by converging around 6000 steps. The figure is indicative of the fact that the learning is converging and its performance has been evaluated using metrics subsequently. The following update takes place in the Q – table at each time step in case of a reward or a punishment.

```
q_update = ALPHA * (q_target - q_predict)
```

The first update after receiving a positive reward is '+1 * Alpha' during the simulation run and then it subsequently converges to zero with oscillations in between. The spikes in between the convergence curve are due to sudden unexpected positive or negative rewards encountered during the simulation run resulting in a large update within the Q – values.

The figure 3.4 shows the subsequent frequency of vehicular collisions occurring during the entire simulation run. The scatter plot below clearly shows that the number of collisions do not reduce over a period of time and thus indicate the models inability to learn effectively. The primary reason for this non – learning can be pointed out

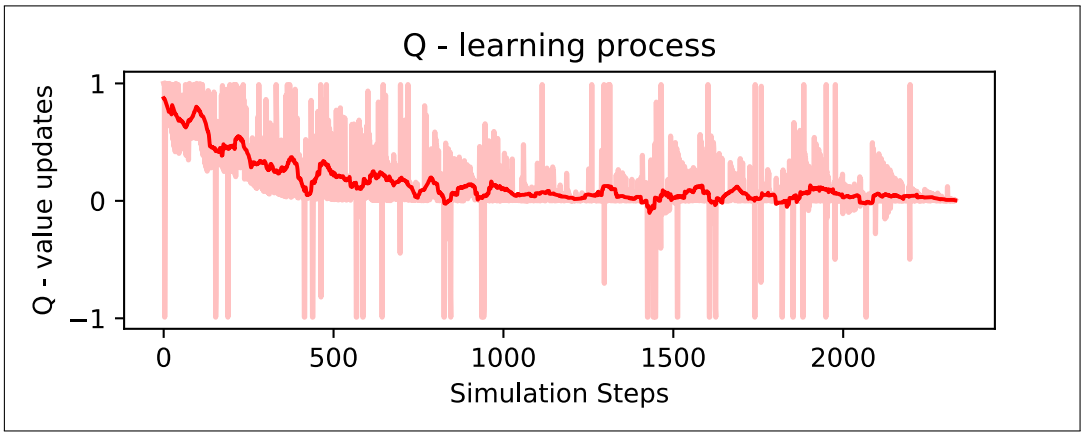


FIGURE 3.3: Convergence of the Q – value updates for the simplistic single lane model

towards the inappropriate characterization of the state space.

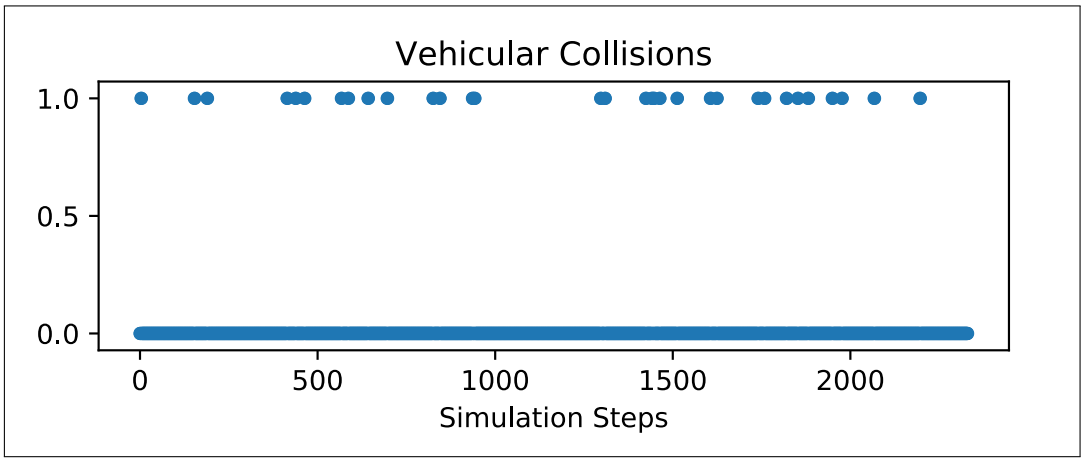


FIGURE 3.4: Vehicular Collisions versus update simulation steps for the simplistic single lane model

The process chain of a typical machine learning algorithm involves the training and testing phase on the data. The model is trained on the training data set and then the performance of this trained model is evaluated on the test data set. In context of our experiments, the training and the test data sets are equivalent to training and testing the reinforcement learning model on random traffic scenarios. The generate_routefile( ) function changes the frequency or the distribution of the traffic in all the directions in each episode in order to simulate the random nature of traffic. Thus, the last episode during the experiment has a complete different traffic distribution against which the model has not been trained for. Thus, it can also be used as the testing data set or the performance evaluation data set along with continuing the further training of the model.

This concept is popularly known as cross – validation in machine learning. It is basically a model validation technique that can be used for assessing how the results or performance of a statistical analysis will generalize to an independent data set.

So the training data itself is divided into several small bifurcations known as 'folds'. Each time during the training of the model, one of the bifurcations of the training data set or one of the 'folds' is chosen as a validation fold. The training process is carried out on the rest of the folds and validated against the validation fold. This is repeated by selecting each of the folds as a validation fold and repeating the entire process for all of them and ultimately averaging the parameters at the end over the entire process [Kie18]. A similar concept has also been used in the context of this project. Thus, reduced number of collisions towards the end of the simulation are indicative of the learning performance of the reinforcement learning model. Since the distribution of the traffic for this last fold or the last episode within the entire training process has been kept random, it in no ways influences or biases the training procedure for the reinforcement learning model and can be used as a measure of learning effectiveness of the model.

As per the software engineering practices, the piece of code after completion is deployed in the environment and tested, validated and its performance is evaluated based on a set of metrics. Similarly, in case of our reinforcement learning simplistic single lane four – way model, the training process has been aborted after 6000 i.e. 3 episodes after it displays convergence. The Q – table has been initialized to '0' for all states and actions before the start of the training process. The Q – table 3.2 indicates the updates that have been performed during the course of the training process and it is a representation of the policy that has been derived during the learning process.

TABLE 3.2: Q – table after the training process for the simplistic single lane four – way model

| | **Lane1** | **Lane2** | **Lane3** | **Lane4** |
|---|---|---|---|---|
| **[0, 0, 0, 0]** | 2.056447 | 2.176807 | 1.414723 | 0.951485 |
| **[1, 0, 0, 0]** | 2.118837 | 1.666555 | 0.924533 | 0.110208 |
| **[0, 1, 0, 0]** | 0.582967 | 2.843301 | 0.267946 | 1.882246 |
| **[0, 0, 1, 0]** | 1.184544 | 0.308302 | 1.442771 | 0.566609 |
| **[0, 0, 0, 1]** | 1.667920 | 0.133540 | 0.916955 | 2.015824 |
| **[1, 1, 0, 0]** | 2.013790 | 1.262026 | 0.399267 | 0.629122 |
| **[1, 0, 1, 0]** | 1.234327 | 0.148606 | 0.719825 | 0.596413 |
| **[1, 0, 0, 1]** | 1.144240 | 0.785555 | 0.027924 | 1.500938 |
| **[0, 1, 1, 0]** | 0.366564 | 1.083195 | 1.836695 | 0.490418 |
| **[0, 1, 0, 1]** | 0.152913 | 2.157283 | 0.415693 | 1.263058 |
| **[0, 0, 1, 1]** | 0.130742 | 0.307809 | 1.886341 | 0.000000 |
| **[1, 1, 1, 0]** | 1.658660 | 0.932352 | 0.112175 | 0.078849 |
| **[1, 0, 1, 1]** | - 0.114000 | 0.085324 | 0.154195 | 0.269015 |
| **[1, 1, 0, 1]** | 1.817107 | 1.759962 | 0.055256 | 1.158867 |
| **[0, 1, 1, 1]** | 0.765822 | 1.118318 | 1.062091 | 0.186209 |
| **[1, 1, 1, 1]** | 0.286146 | 0.522258 | 0.912417 | - 0.083280 |

The Q – table 3.2 has to be interpreted to understand the strategy that has been developed during the training process. For e.g. for the state as [1, 1, 0, 0], i.e. with a vehicle in the left lane and the bottom lane, the action – values are maximum for the action 'Lane1' with the value 2.013790. It signifies that during the deployment of the model for a test scenario, it will choose to release the vehicle from the left lane when it encounters the state [1, 1, 0, 0].

The model has been trained for three episodes of 2000 steps each. After the training process, the trained reinforcement learning model has been subjected to a test episode of 2000 steps and a random traffic distribution (varied within certain pre – defined limits), which has not been a part of the training process. The results of the entire process have been summarized in the table 3.3.

TABLE 3.3: Summary of the results for the simplistic single lane four – way model

| Parameters | Episode | Number of Collisions | Average number of Collisions | Waiting Time | Average Waiting Time |
|---|---|---|---|---|---|
| **Learning Rate = 0.1 Discount Factor = 0.9 Epsilon = 0.99** | 1 | 20 | 21 | 253 s | 180.3 s |
| | 2 | 22 | | 163 s | |
| | 3 | 22 | | 125 s | |
| | Test | 22 | - | 193 s | - |

Three iterations of the enhancement within the characterization of the state space have been carried out in total to ensure a better performing model. In order to establish a relationship between the results derived from these three iterations, the number of simulation steps have been restricted to 2000 per episode and the number of episodes to 3 for the benchmarking process. This is basically to be able to establish uniformity within the results in order to arrive at better conclusions. Thus, a direct comparison between the models, as well as a uniform benchmarking process with similar number of episodes and steps has been carried out.

#### 3.2.1.5 Limitations of the simplistic single lane model

The characterization of the state space is highly inappropriate for ensuring a descent level of optimization and reliability of the reinforcement learning approach with respect to intersection control. As justified above, the main purpose of such a simplistic characterization is just to establish an interface between reinforcement learning and the SUMO simulator via the TraCI API in order to get the algorithm running. The inappropriateness of the state space has been elucidated through actual examples below. As can be seen in the image 3.5, two vehicles are approaching from the right and the bottom lane. A per our characterization, the state space would be retrieved by the algorithm as [0, 1, 1, 0]. Since the vehicle approaching from the right has been detected at an earlier time step by the right lane detector with state as [1, 0, 0, 0], it is allowed to maneuver the intersection directly without a prior stopping action. At the next time step, the vehicle from the bottom lane approaches the intersection and is detected by the bottom lane detector. This changes the state space definition to [0, 1, 1, 0].

However, at this point of time, it is unknown if the vehicle approaching from the right is in section between the incoming detector and the junction or in the section between the junction and the outgoing detector along the right lane. Only the presence of the vehicle between incoming and the outgoing detector is taken into account while setting the state. During the training period of the model, it might have learned to halt the vehicle approaching from the bottom lane just before the junction in case of a vehicle moving in the right lane due to a previous collision where it might have received a negative reward signal. For e.g. with respect to the table 3.2, the action – value is maximum for the action 'Lane3' for the state [0, 1, 1,

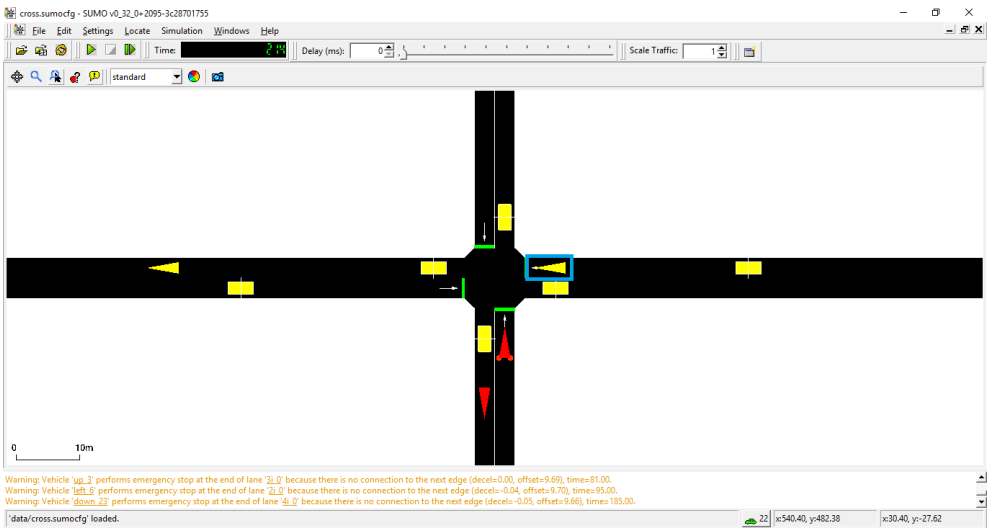


FIGURE 3.5: Limitations of the simplistic single lane model (1)

0]. Hence, as it can be seen in the image below, the vehicle in the bottom lane is requested to wait just before the junction in order to avoid a possible collision. Thus, even though there is no possibility of a collision even if the vehicle from the bottom lane is released, at the current time step, it is kept halted until the vehicle in the right lane completely maneuvers the intersection and is detected by the outgoing detector on the right lane and the state changes to [0, 1, 0, 0]. However, ensuring highly optimum intersection efficiency is one of the aims of our project.

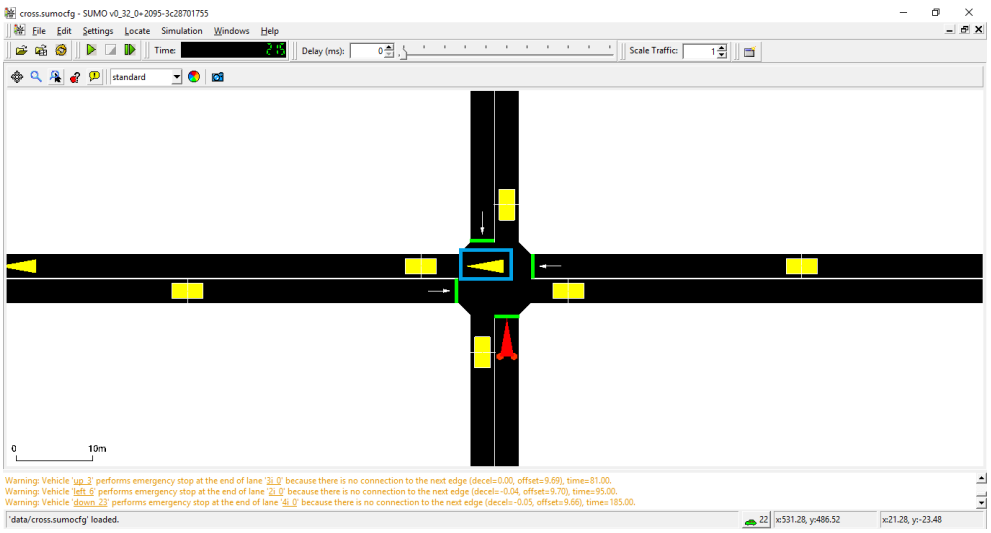


FIGURE 3.6: Limitations of the simplistic single lane model (2)

### 3.2.2 Reinforcement learning for a split state single lane model

The state space characterization has to be improved to overcome the limitations of the previous one. This section puts forward a new characterization of the state space and the subsequent results and its limitations.

#### 3.2.2.1 Characterization of the state space for a split state single lane model

Unraveling the limitations described in the earlier section through extension of the state space characterization may lead to a better and optimal solution. The presence or non – presence of a vehicle within the two different sections along the same lane has also been accounted for in this extended state space characterization. The lane has been split into two different sections – first section consisting of the lane portion between the incoming detector and the junction and the second section comprising of the lane portion between the junction and the outgoing detector.

The states are characterized as an eight element single row vector, with each element either a '0' or a '1', either indicating the non – presence or presence of a vehicle respectively within a section along the lane. The first element indicates a vehicle non – presence or presence in the first section of the first lane i.e. incoming vehicle from the left intended to travel to the right i.e. along the route west to east between the incoming detector along the left lane and the junction. The subsequent elements represented the subsequent clockwise lanes and sections. For e.g. the state [0, 1, 1, 0, 0, 0, 0, 0] indicates that there is a vehicle in the left lane i.e. along the route west to east in the second section between the junction and the outgoing detector along the left lane and there is a vehicle in the bottom lane i.e. along the route south to north in the first section between the bottom lane detector and the junction and there are no vehicles in the right and top lanes i.e. along the routes from east to west and north to south respectively. With such a state space representation, 256 different states are thus possible. The actions the agent can take in each state and the reward signals or punishments it receives are the same as in the earlier simplistic model. An ordinary data – frame based Q – table can be generated with 256 rows representing the possible states and 4 columns representing the 4 different viable actions that can be chosen from each state. The algorithmic structure and the simulator setup are the same as described for the simplistic single lane model.

#### 3.2.2.2 Results for the split state single lane model

Several simulation runs have been experimented out for the split state single lane model. The hyper parameter 'Epsilon' is not changed even in this case since only 256 diverse states are possible. Thus an 'Epsilon Greedy' policy is followed to tackle the Exploration versus Exploitation dilemma as explained in the section 3.2.1.2. Multiple experiments have been executed by varying the number of simulation steps per episode and the number of episodes. The final experimentation has been carried out consisting of 7500 training steps distributed over 3 episodes. The figure 3.7 indicates the progression of the training steps as it approximates a policy by converging around 7500 steps i.e. the updates in the Q – values become almost negligible. The developed policy has been evaluated using a set a metrics that are safety i.e. the number of collisions and throughput i.e. waiting time of the vehicles

The figure 3.8 displays the frequency of the vehicular collisions occurring during the entire training process. The scatter plot is indicative of the performance of the model since the number of collisions do not decrease as the training process progresses. One of the reasons for it is the inappropriate characterization of the state space that has been used. It indicates the ineffectiveness of the model to comprehend the erroneousness of the release actions that may have caused the collisions i.e.

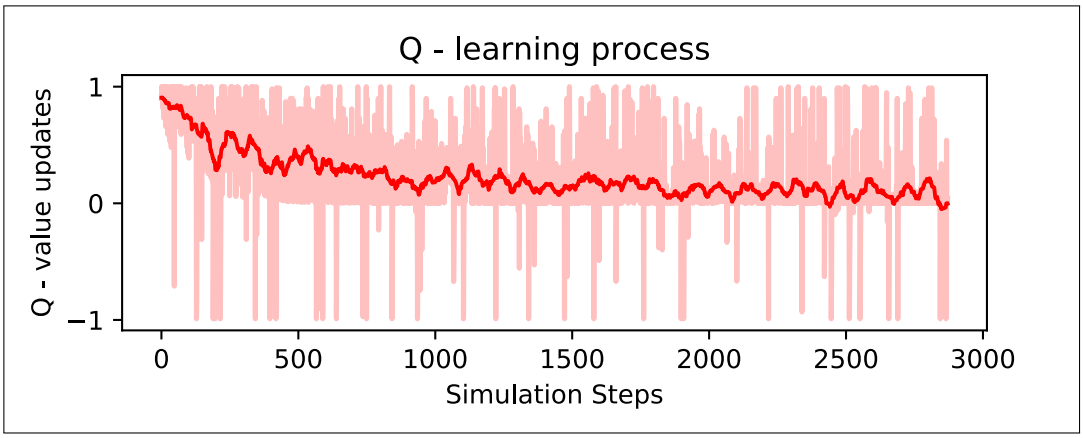


FIGURE 3.7: Convergence of the Q – value updates for the split state single lane model

it is unable to comprehend about the wrong actions that are causing the collisions and hence it cannot reduce them subsequently.

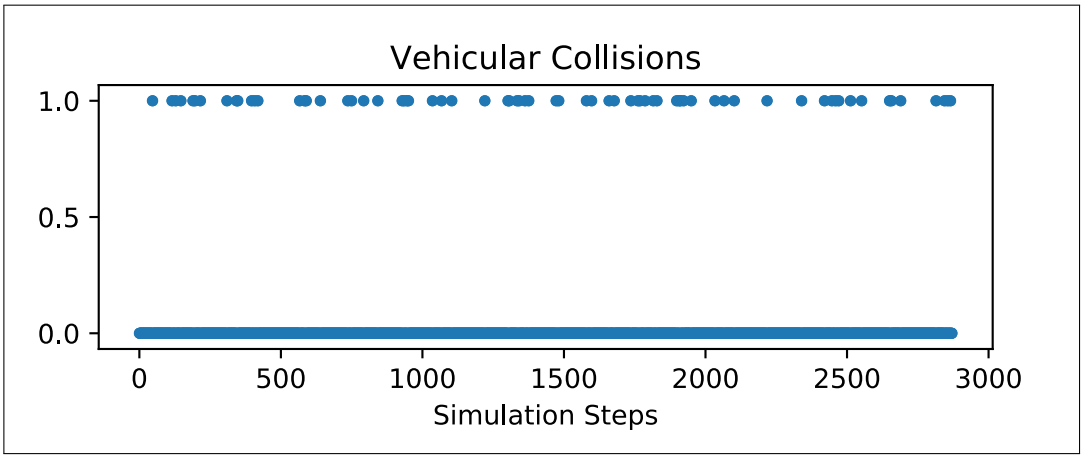


FIGURE 3.8: Vehicular Collisions versus update simulation steps for the split state single lane model

It is important to note that the reward achieved by the system of systems level intersection controller or the agent cannot be used as a performance metric in case of our project. The vehicles maneuver the intersections even in case of collisions and receive a reward which is later penalized. However, more or less the cumulative rewards at the end of the episode are dependent on the traffic distribution which is varied for each episode to impart robustness to the agent. Each vehicle that starts from the point of generation traverses its complete route and reaches the desired target point unless the episode has terminated. Hence, the cumulative reward received for each episode cannot be a performance metric of the agent and this concept has been explained in detail in the section 3.2.4.3. concerning the reward characterization for the system of systems level intersection controller.

The model has been trained for three episodes of 2500 steps each. After the training process, the trained reinforcement learning model has been subjected to a test episode of 2500 steps and a random traffic distribution which has not been a part of

TABLE 3.4: Summary of the results for the split state single lane four – way model

| Parameters | Episode | Number of Collisions | Average number of Collisions | Waiting Time | Average Waiting Time |
|---|---|---|---|---|---|
| **Learning Rate = 0.1 Discount Factor = 0.9 Epsilon = 0.99** | 1 | 50 | 48 | 22 s | 11 s |
| | 2 | 55 | | 4 s | |
| | 3 | 40 | | 7 s | |
| | Test | 56 | - | 3 s | - |

the training process. The results of the entire process have been summarized in the table 3.4. The performance of the model throughput – wise or in context of efficiency is better in comparison with the previous model, since the average waiting time is only 11 seconds. However, safety wise, the model indicates a huge deterioration as the average number of collisions are large i.e. 48.

#### 3.2.2.3 Limitations of the split state single lane model

Even with such a characterization of the state space, all the dynamics of the system cannot be represented for a better performing reinforcement learning model. This section aims to address the limitations of this split type state space characterization. The incompleteness of the state space representation has been demonstrated through an example below.

The image 3.9 shows two vehicles approaching from the left and the bottom lane. As per our representation, the state space would be [1, 0, 1, 0, 0, 0, 0, 0] since there is a vehicle in the left lane in the first section between the junction and the incoming detector along the left lane and there is a vehicle in the bottom lane in the first section between the bottom lane incoming detector and the junction. Since the vehicle approaching from the left has been detected at an earlier time step by the left lane detector with the state representation as [1, 0, 0, 0, 0, 0, 0, 0], it is released directly without a preceding stopping action. At the subsequent time step, the vehicle from the bottom lane approaching the junction passes over the bottom lane detector. This changes the state space to [1, 0, 1, 0, 0, 0, 0, 0].

The vehicle from the bottom lane is made to stop near the junction and the vehicle from the left lane is allowed to maneuver the junction. The state space representation changes to [0, 1, 1, 0, 0, 0, 0, 0] after the vehicle from the left lane enters the second section of the left lane i.e. between the junction and the outgoing detector along the left lane. The vehicle crosses the junction in two time steps until it is again detected by the outgoing detector. As seen in the image 3.10, after one step, the vehicle from the bottom lane can be possibly released without a collision with the vehicle from the left lane. However during the model training, it might have possibly learned to halt the vehicle approaching from the bottom lane before the junction when the vehicle from the left lane is in the 2nd section due to a previous collision where it might have received a punishment. Thus, even though there is no possibility of a collision even if the vehicle from the bottom lane is released, it is kept halted due to indefinite knowledge about the precise position of the vehicle in the second section along the left lane. It is released after the vehicle in the right lane completely maneuvers the

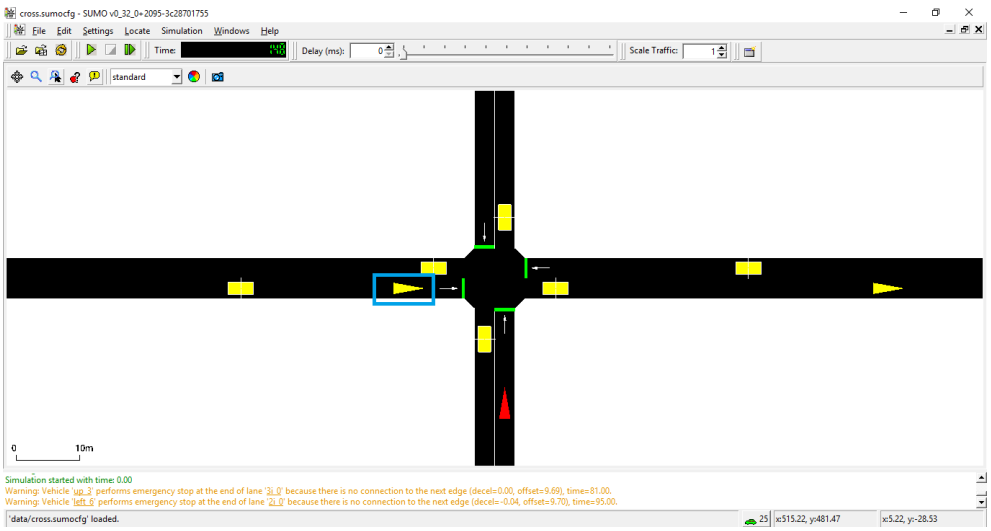


FIGURE 3.9: Limitations of the split state single lane model (1)

intersection and is detected by the outgoing detector on the right lane and the state changes to [0, 0, 1, 0, 0, 0, 0, 0].

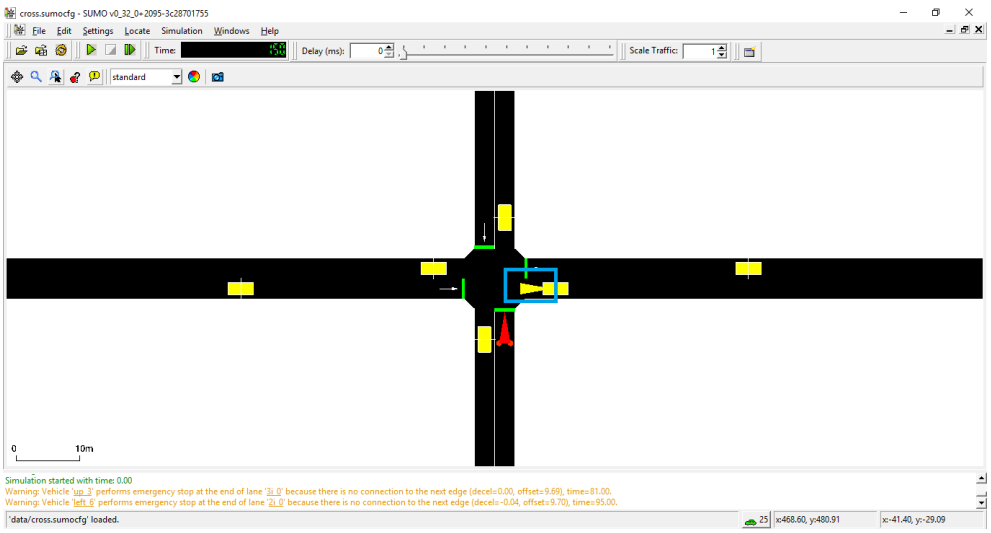


FIGURE 3.10: Limitations of the split state single lane model (2)

Thus even with this extended representation of the state space, ensuring a high level of traffic flow optimization is not possible. This leads us to the conclusion that the characterization of the state space should be further revised for a better performing model.

### 3.2.3 Reinforcement learning for an advanced single lane model

The inadequacy in the representation of the state space for the previous two models motivates us to look for a further better version. This section points out a much better and precise state space characterization in contrast with the previous models.

#### 3.2.3.1 Characterization of the state space for an advanced single lane model

All the earlier state space characterizations did not convey the exact position of the travelling vehicle along the lane. Without this information, ensuring a high level of optimization would not be possible. Hence, the state space was extended to also account for the exact position of the travelling vehicle along the lane in conjunction with information regarding the presence or non – presence of the vehicle in a lane.

The SUMO simulator splits the lane into two different sections – first section consisting of the lane portion between the incoming lane and the junction and the second section comprising of the lane portion between the junction and the outgoing lane. The state space representation thus consists of an eight element single row vector, with the $1^{st}$, $3^{rd}$, $5^{th}$ and the $7^{th}$ element either a '0' or a '1', indicating the non – presence or presence of a vehicle respectively within a section along the lane. The $2^{nd}$, $4^{th}$, $6^{th}$ and the $8^{th}$ element represent the corresponding position of the vehicle along the lane in case of its presence within the left, bottom, right and top lane respectively. For e.g. the state [0, 0, 1, 468, 1, 4, 0, 0] represents that there is a vehicle in the bottom lane at a relative distance of 468 with respect to the junction start i.e. between the incoming detector and the junction along the lane and there is a vehicle in the right lane at a relative distance of 4 with respect to the junction start i.e. between the junction start and the outgoing detector along the lane and there are no vehicles in the left and top lanes. With numerical values for the relative position along the lane, a very huge amount of diverse states are possible – around **65,610,000 different states** i.e. approximately around 65 million states. The agent can take in each state the same four actions that have been described in the earlier experiments. It is important to note that the position of a vehicle can be retrieved using the following method from the 'traci.vehicle' class.

```
plc1 = traci.vehicle.getLanePosition(a[0])
```

The method returns float values however; they are used in the state space representation by rounding them off to the nearest integer. One of the primary reasons for this is to limit the diversity within the possible amount of states. With a float type continuous state space representation, almost infinite different states are possible. Also, the rounding off does not affect the performance of the model, but definitely limits the complexity. An ordinary data frame based Q – table cannot suffice the requirements in case of such a state space characterization and hence neural networks have to be used. The advantages of neural networks in context of non – linear function approximations have been already clarified in detail in the sections above.

It is important to note that the state space characterization can be simplified by just appending the position of the vehicle along the lane and thus the presence of a position within the state space itself represents the presence or non – presence of a vehicle within a particular lane. For e.g. in case of the previous example, the state space could have been characterized as [0, 468, 4, 0] that indicates there is no vehicle in the left lane, a vehicle at a relative position of 468 from the junction along the bottom lane, a vehicle at a relative distance of 4 from the junction in the right lane and no vehicle in the top lane. However, in case of SUMO, the positions along the lane are retrieved with respect to the starting point of the junction along the lane till the junction center. The start of the junction has a position 0 and distances along the outgoing portion of the lane increase with respect to this start of the junction position, in our case 0 to 9, since the outgoing detector is placed at 5 units with respect to

the junction center. Also the distances from the incoming detector (in our case 460) increase upto the position just before the start of the junction (in our case 495). Thus vehicles can have the relative distance along the lane within a range of values from 460 to 495 (approaching the junction) or exactly 0 (precisely at the start or end of the junction) or from 1 to 9 (crossing the junction and moving into the outgoing lane). Thus a certain release action at a certain time step may result in the vehicle being exactly at the start or end of the junction thus having this relative distance as 0 and then it may again increase along the subsequent time steps. This may indicate that the vehicle has successfully maneuvered the intersection and passed over the outgoing detector. However, at the next time step it is again characterized by a distance between 1 to 9. The network may not be able to comprehend such a transition of states and may result in deceitful or improper training. Hence, for each lane, a '0' or a '1' along with the corresponding relative distance represents the presence or non – presence of a vehicle much better.

#### 3.2.3.2 Approach for an advanced single lane model

As the number of diverse states are too high, a traditional Q – table cannot not be used in the context of this state space characterization. A neural network is a viable option to predict the Q – values and ultimately pick up a suitable action from the action space. This section aids in the understanding of the basic concepts defined for reinforcement learning in context of Q – networks. The neural network receives the environment state as an input and generates the Q – values for each of the possible actions using a loss function that has been defined below. The subsequent weight updates take place in correspondence to the gradient.

$$\mathrm{L}(\theta) = \mathbb{E}[(y - Q(s,\ a;\ \theta))^2]$$

$$\nabla_\theta \mathrm{L}(\theta) = \mathbb{E}\left[\left(r + \gamma max_{a'} Q\left(s',\ a';\ \theta\right) - Q(s,\ a;\ \theta)\right) \nabla_\theta Q(s,\ a;\ \theta)\right]$$

*where* $\theta$ indicates the network parameters and $\nabla_\theta \mathrm{L}(\theta)$ *indicates the gradient of the loss function*

Reward clipping is one of the important concepts in order to scale and clip the values of the rewards within the range [-1, +1]. It prevents the weights from boosting during back propagation and has been proved in several applications of Q – networks [Mni+13][Mni+15]. Also in a similar work where the Q – network has been used to evaluate the action values for a traffic light control problems, the reward function is scaled by a factor $\alpha$ to clip it in the above defined range for the stability of the network [Cas17]. This topic has been explained in detail in section 3.2.4.3. about reward function characterization for the system of systems level intersection controller.

The assignment of the punishment signal in case of collisions to the action causing the collision is an important part of the entire learning process. The vehicle released at the later time stamp has to be penalized for causing the collision. This cannot be directly obtained in case of SUMO and several lines of python script has to be programed to impart this functionality. The list of the colliding vehicles is retrieved using the 'getCollidingVehicleIDList( )' method and the vehicle that has caused the collision and the corresponding release action that causes this collision has been deduced using 'regular expressions' in Python. The code snippet below

demonstrates this functionality and the subsequent Q – algorithm updates for the network.

```
if R_: # If a collision has taken place
    collision_division_factor = 2.5
  collision_factor = -1
  R_ = R_ * collision_reward / collision_division_factor
  # This is a hyperparameter explained in the section 3.2.4.3

  c = len(collided_vehicles) # ID's of the vehicles involved
  # in collision e.g. collided_vehicles = [up_32, left_41]
  d1 = collided_vehicles[c - 1]
  # E.g. [left_41] causing the collision
  d2 = collided_vehicles[c - 2]
  # E.g. [up_32] causing the collision
  m = ['right', 'up', 'left', 'down'] # To arrive upon the
  # release action that resulted in the collision

  for n in range(len(m)):
    m[n] = re.compile(m[n])
    p = m[n].match(d1)
    # Here n would be 2 since left_41 would match the
    # third element in the List m with index 2
    if p:
      if n == 0:
        pos1 = S_old[0, 1]
        n1 = 0
        break
      if n == 1:
        pos1 = S_old[0, 3]
        n1 = 1
        break
      if n == 2:
        pos1 = S_old[0, 5] # Store old position of the left_41
        n1 = 2
        break
      if n == 3:
        pos1 = S_old[0, 7]
        n1 = 3
        break

  for n in range(len(m)):
    m[n] = re.compile(m[n])
    p = m[n].match(d2)
    # Here n would be 1 since up_32 would match the
      # second element in the List m with index 1
    if p:
      if n == 0:
        pos2 = S_old[0, 1]
        n2 = 0
        break
      if n == 1:
        pos2 = S_old[0, 3] # Store old position of the up_32
        n2 = 1
        break
      if n == 2:
        pos2 = S_old[0, 5]
        n2 = 2
        break
      if n == 3:
        pos2 = S_old[0, 7]
        n2 = 3
        break
```

```
# The old positions are compared to find out the vehicle released at
# the later time step that caused the collision
if pos1 > 440 and pos2 > 440: # pos1 = 490 and # pos2 = 495
    if pos1 > pos2:
      col_action = n2
  else:
    col_action = n1
    # Culprit Action is late release of left_41
if pos1 > 440 and pos2 < 440:
  col_action = n1
if pos1 < 440 and pos2 > 440:
    col_action = n2
if pos1 < 440 and pos2 < 440:
  if pos1 > pos2:
    col_action = n2
  else:
    col_action = n1

if col_action == 0:
  A_old = 'Lane1' # The release action causing collision
  B_old[0] = 0

if col_action == 1:
  A_old = 'Lane2'
  B_old[0] = 1

if col_action == 2:
  A_old = 'Lane3'
  B_old[0] = 2

if col_action == 3:
  A_old = 'Lane4'
  B_old[0] = 3


allQ = sess.run(Qout, feed_dict={input_NN:S_old})
# The retireval of Q values for the old state space
Q1 = sess.run(Qout, feed_dict={input_NN:S_})
# The retireval of Q values for the current state that was the
# futuristic stae for the old one
maxQ1 = np.max(Q1)
# The maximum Q value for the state
targetQ = allQ

targetQ[0, B_old[0]] = R_ + GAMMA * maxQ1
# The update for the Q - values

_, W1, W2, W3, LOSS = sess.run([updateModel, weights1, weights2,
                                weights3, loss], feed_dict={
                                input_NN:S_old, nextQ:targetQ})
# The network update for the new Q values

episodeBuffer.add(np.reshape(np.array([S_old, B_old[0], R_, S_]),[1,
                                4]))
# Storing experience in replay buffer for stabiliy of net
```

#### 3.2.3.3 The network for the advanced single lane model

This section provides insights into the network architecture for the single lane four – way model with advanced state space characterization. A very large number of experiments have been carried out with respect to the network architecture and the

hyper parameters to achieve a good performing agent i.e. a neural network or an agent that shows good performance based on the metrics used. In order to establish a relationship with the previous approaches, experiments have also been carried out with 2000 simulation steps per episode and number of episodes restricted to 3 like the previous versions of the state space characterization. The network consists of a simple architecture with 8 neurons at the input, a hidden layer with 8 neurons and 4 neurons at the output.

The selection of the eights neurons at the input and the four neurons at the output is justified by the eight element single row vector at the input and the clipped Q – values for the fours actions at the output. There is no deterministic rule in context of selection of activation functions; however, over a period of time, researchers have laid a few thumb rules to aid in a better design of the network. The step activation function fires the neuron if the 'Y' value is greater than a certain pre – determined threshold and deactivates it otherwise. However, the major limitation of such a binary activator is that it cannot be used in classification problems. In context of our project, activation of more than 1 output neuron may result in complication of the entire action selection process. The performance might be much better if the neurons output intermediate or analog values rather than binary. A linear function may also not be a promising solution where the activation is directly proportional to the input. The gradient for a linear function is a constant and has no dependencies with the input. Hence during back – propagation of such networks, the changes are always constant and do not depend upon the change in the input. This may result in ineffectiveness to predict the Q – values and thereby affect the action selection. A network with multiple layers using the linear activation function can be replaced principally with a single layer that is a linear function of the first layer input. The rectified linear activation function popularly known as 'ReLu' is actually linear along the positive X – axis; however combinations of ReLu are non – linear since it clips the negative input to 0. The range of ReLu is thus between [0, inf]. The biggest advantage of this activation function is the sparsity of activations in comparison with sigmoid or tanh activation functions that cause almost all neurons to fire (dense activation) that increases the requirement on the computational resources. The network above uses 'ReLu' as the activation function for the hidden layer with 8 neurons. However, since ReLu clips the negative input to 0, the gradient can go towards 0, thus resulting in no updates for the weights during descent. This may result eventually into non – responding neurons and may make a significant part of the network passive. The leaky ReLu solves this problem by using an insignificant non – linear function for negative inputs.

The above described activation functions that can take values upto infinity cannot be used for the output neurons within our framework of reinforcement learning for cooperative intersection control using the SUMO simulator. The main reasons for this conclusion are the characterization of our state space, our actions space, our reward strategy and the entire SUMO simulator framework. During the initial phases of training of the network model, it is unable to well approximate the output action value functions due to sparsely generated test data. It receives a high reward for a few initial actions and updates the weights quite high in order to minimize the loss function i.e. difference between the target and predicted action value functions and improve its approximation. With a multilayer network, the matrix multiplications with high weights result in very high approximations for particular action value functions. Over a period of time during the training period, these numbers reach

infinity due to the exponential nature of the multiplication and do not update the weights anymore (multiplication of very large numbers in tensorflow resulting in NaN).

The main reasons for this are the sparsely received rewards initially for actions taken at each time step. In order to avoid such a training behavior, nonlinear or analog activation functions along with limited output range [0, 1] or [-1, 1] such as sigmoid or tanh activation functions are preferred. It does not blow up the activations and maintains them in a certain range. The sigmoid function is one of the most widely used activation function that has a very smooth and non – linear gradient. The gradient is steep initially which causes even minuscular changes in the input values to change the output significantly and the output values tend to be unresponsive to the changes in the input towards the end due to flattening of the gradient. The tanh function is actually just a scaled up version of the sigmoid activation function. Because of the scaling up, the gradients are much steeper for tanh function as compared to sigmoid functions. Due to this vanishing gradients problem, the 'softmax' activation function has been used at the output layer. Several different error and loss functions have been tested out for the above given architecture along with the hyper parameters and the results have been summarized in the section below. The squared error has been used in the final version of the network with the gradient descent optimizer.

#### 3.2.3.4 Results for the advanced single lane model

This section summarizes the results for the single lane four – way model with the advanced state space characterization. The agent has been trained for a total of 10,000 steps split amongst 5 episodes. The convergence of the updates in the Q – values and the heavy oscillations throughout the progression of the training process have been attempted to be countered in the section 3.2.4.7.

Advanced concepts of Q – networks such as Experience Replay have also been used to provide stability to the network. The basic concept behind using it is that by storing an agent's experiences, and then randomly sampling it by drawing batches from this storage to train the network, the model can learn more robustly and it imparts stability to the network. By storing the experiences and drawing them randomly, we prevent the network from only learning about what it perceives in the environment currently. We allow it to learn from its diverse past experiences where each of the experience is stored as a tuple of the typical MDP transition. The experience replay buffer stores these experiences continuously and removes the old ones. After a certain pre – determined number of steps, the network is trained based on a randomly drawn uniform batch of experiences from the storage. A separate class has been written for it that is used by our Python TraCI script during the training process of the network or the agent. For the above experiment, the batch size is chosen as 32 and the update frequency has been set to 2000 steps after 4000 initial steps. The code snippet below describes the new class – experience_buffer( ).

```
class experience_buffer():
    def __init__(self, buffer_size=5000):
        self.buffer = []
        self.buffer_size = buffer_size

    def add(self, experience):
        if len(self.buffer) + len(experience) >= self.buffer_size:
```

```
            self.buffer[0:(len(experience) + len(self.buffer)) -
            self.buffer_size] = []
        self.buffer.extend(experience)

    def sample(self, size):
        return np.reshape(np.array(random.sample(self.buffer, size)),
        [size, 4])
```

The network is trained on the sampled experiences after pre – defined number of steps (pre – training steps) of simulation have elapsed. The exploration versus exploitation dilemma has also been confronted in this approach by setting the initial Epsilon value as 0.6 (Choosing 40 random actions every 100 selections). This Epsilon value has been marginally increased after each simulation step to reach a maximum value of 0.99 after 6000 simulation steps (Epsilon Greedy policy). The following function is used to ensure these marginal increases in the value of Epsilon per simulation step –

```
EPSILON_Orig = 0.6
MAX_EXPLORATION_STEPS = 6000
EPSILON = min(EPSILON + (1 - EPSILON_Orig) / (MAX_EXPLORATION_STEPS), 0
                                    .99)
```

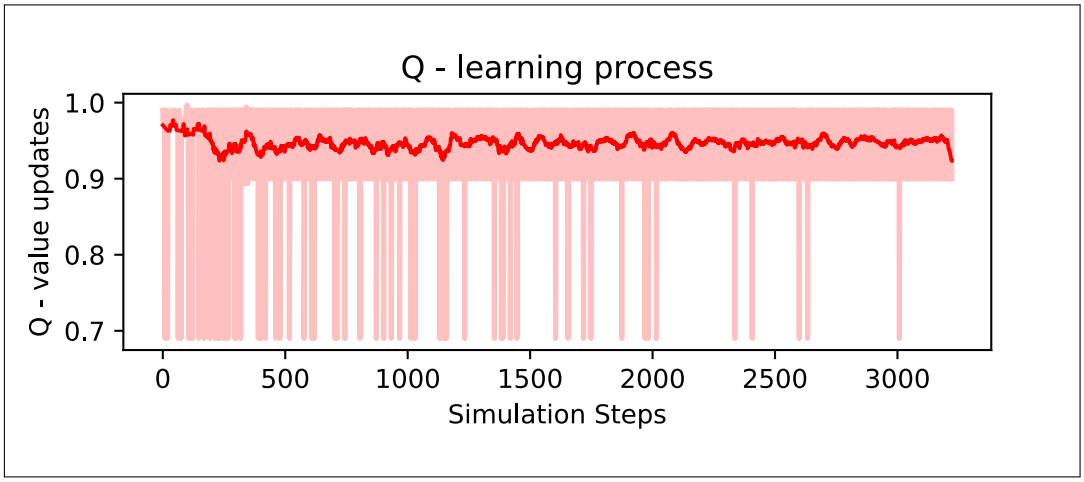


FIGURE 3.11: Convergence of the Q – value updates for the advanced single lane model

The importance of the exploration has been explained in detail in the section 3.2.4.4. concerning the exploration versus exploitation dilemma for the triple lane four – way model. The figure 3.11 indicates the course of the training process for the single lane four – way model. The disruptive spikes during the training process indicate the sudden rewards or punishments received during the training process resulting in heavy updates of the network. The replay buffer and the reducing number of collisions stabilize the too heavy updates over the course of training process.

The figure 3.13 shows the frequency of vehicular collisions occurring over the simulation run with 5 episodes and a total of 10000 steps. The meager number of collisions towards the end indicate the effectiveness of the model and its learning ability in comparison with the previous ones. The figure 3.12 displays the gradual increase in the epsilon value and a constant value of 0.99 after 6000 simulation steps.

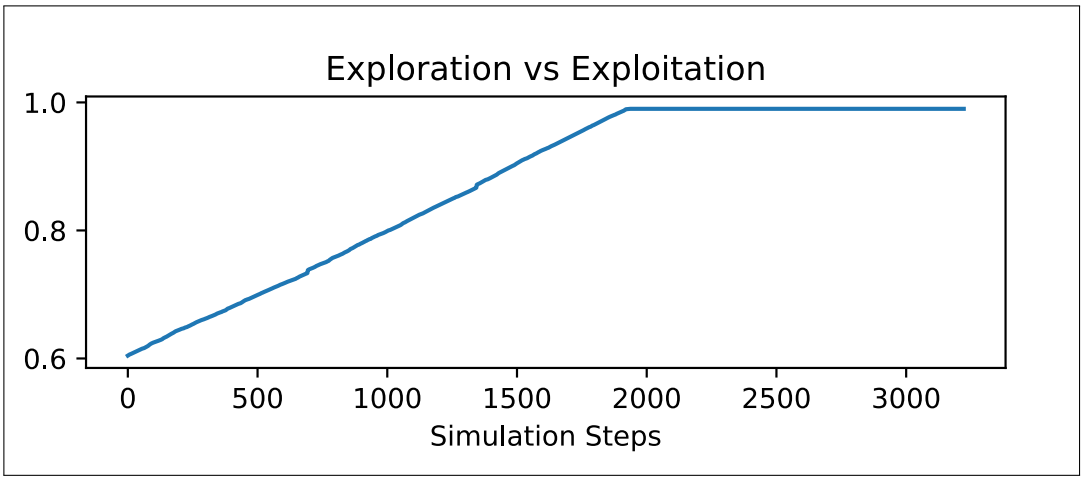


FIGURE 3.12: Exploration versus Exploitation against update simulation steps

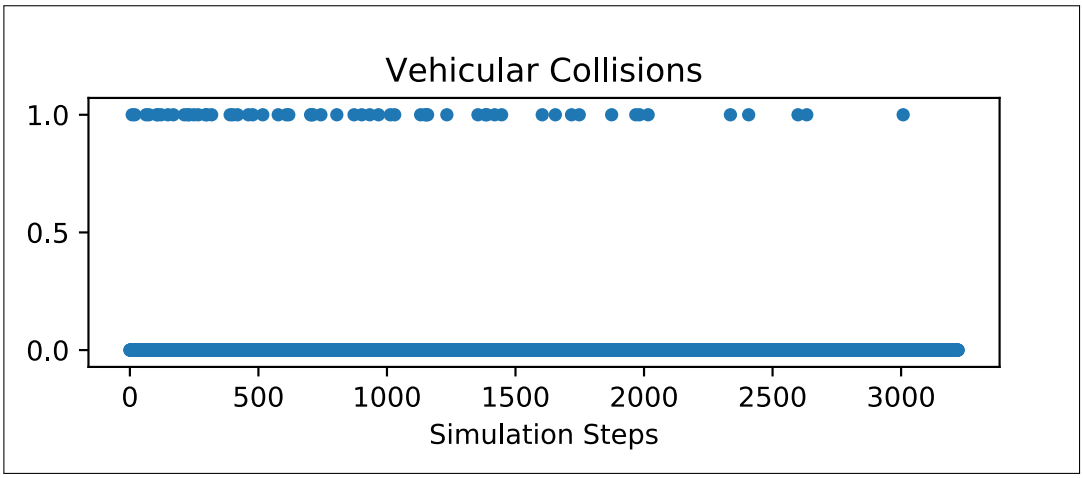


FIGURE 3.13: Vehicular collisions versus update simulation steps for the advanced single lane model

Even though this approach of state space characterization proves to be effective, it largely depends upon the network architecture, set of hyper parameters, number of episodes, traffic distribution during each episode, etc. This concept has been explained in detail in the section 3.2.4.7. for the triple lane four – way model. With the enhanced state space characterization, the model learns effectively and this can be observed from the reducing number of collisions over a period of time. The model has been trained for five episodes of 2000 steps each and the training process has been aborted after 10,000 steps after the lowest value of updates in the Q – values approximately around 0.9 is reached. After the training process, the trained reinforcement learning model has been subjected to a test episode of 2000 steps and a random traffic distribution which has not been a part of the training process. The results of the entire process have been summarized in the table 3.5.

The experiments with uniform parameters have also been carried out to establish the relationship between the different state space characterizations. For the benchmarking process, the single lane four – way models with the three different state

TABLE 3.5: Summary of the results for the advanced single lane four – way model

| Parameters | Episode | Number of Collisions | Average number of Collisions | Waiting Time | Average Waiting Time |
|---|---|---|---|---|---|
| Learning Rate = 0.01 | 1 | 52 | 24 | 27 s | 128.8 s |
| Discount Factor = 0.9 | 2 | 33 | | 45 s | |
| Epsilon = 0.6 – 0.99 | 3 | 22 | | 95 s | |
| Max exploration steps = 6000 | 4 | 8 | | 187 s | |
| Buffer batch size = 32 | 5 | 6 | | 290 s | |
| Update frequency = 2000 | Test | 6 | - | 150 s | - |

space characterizations have been subjected to training and the cross – validation process consisting of a total of 6000 steps divided into 3 episodes. The results for the same have been tabularized in table 3.6 to aid in a concise understanding of the same.

TABLE 3.6: Performance comparison between the state space characterizations

| State Space depiction | Episode | Number of Collisions | Average number of Collisions | Waiting Time | Average Waiting Time |
|---|---|---|---|---|---|
| **Simplistic Model** | 1 | 20 | 21 | 253 s | 180.3 s |
| | 2 | 22 | | 163 s | |
| | 3 | 22 | | 125 s | |
| **Split State Model** | 1 | 38 | 39 | 22 s | 11.3 s |
| | 2 | 34 | | 7 s | |
| | 3 | 46 | | 5 s | |
| **Advanced Model** | 1 | 21 | 11 | 136 s | 232 s |
| | 2 | 10 | | 141 s | |
| | 3 | 2 | | 419 s | |

The figure 3.14 depicts the tabular information in a graphical form. It aids in a better understanding of the overall trend of the different versions over several episodes.

#### 3.2.3.5 Advantages of the advanced single lane model

The results achieved for the advanced single lane four – way model proves the effectiveness of the state space characterization in contrast with the other approaches. This section explicates these advantages in detail through examples. The image 3.15 shows two vehicles in the left and the bottom lane where the vehicle in the bottom lane detected at an earlier time step has been already released and allowed to cross the intersection. The state space representation for the indicated scenario would be [1, 489, 1, 2, 0, 0, 0, 0] indicating a vehicle at a relative position of 489 from the junction along the left lane, a vehicle at a relative distance of 2 from the junction in the bottom lane and no vehicles in the top lane and the right lane.

However, since the network has been trained to account for the positions of the vehicle along the bottom lane and the left lane, it comprehends / learns over time that it may not collide with the vehicle from the bottom lane even if the vehicle from

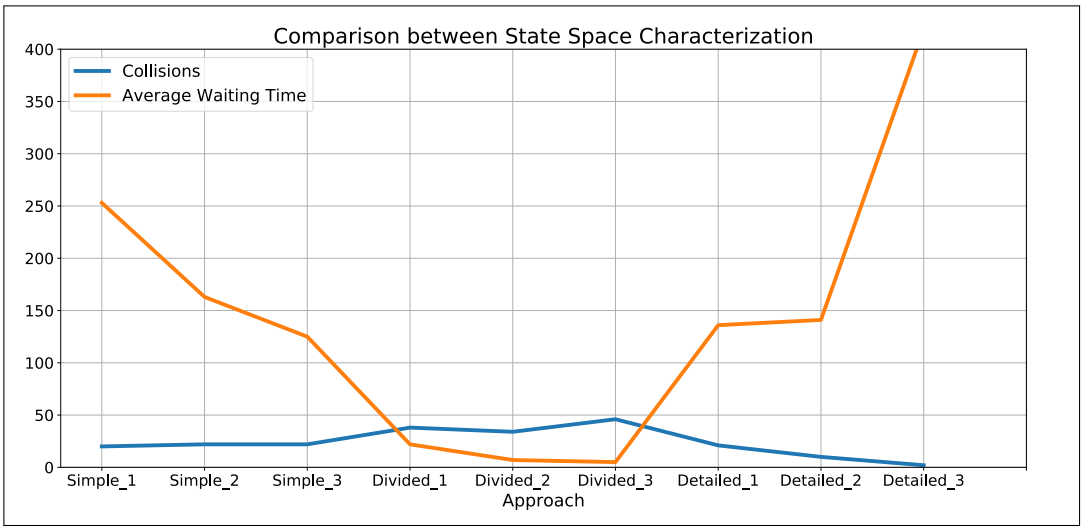


FIGURE 3.14: Performance comparison between the state space characterizations

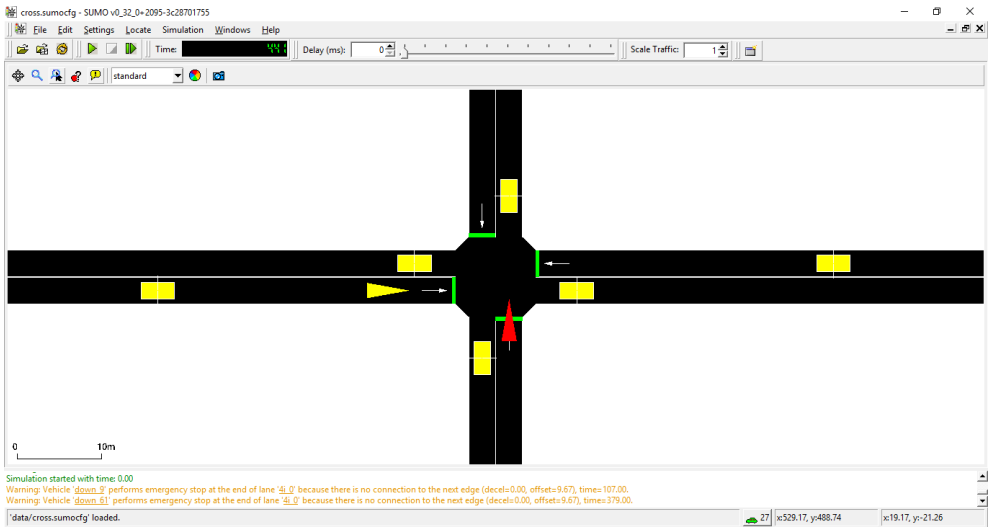


FIGURE 3.15: Advantages of the advanced single lane model (1)

the left lane has been released; and hence it releases the vehicle from the left lane. This can be witnessed from the figure 3.16, where the left and the bottom lane vehicles are concurrently in the junction area. Thus with this extended representation of the state space, ensuring a high level of traffic flow optimization is possible and yields much better results in comparison with the previous approaches.

It is important to note that this state space characterization also has a finite amount of limitations. These limitations have been explained in the section 3.2.4.7. for the triple lane four – way model for intersection control using reinforcement learning along with the proposed viable solutions.

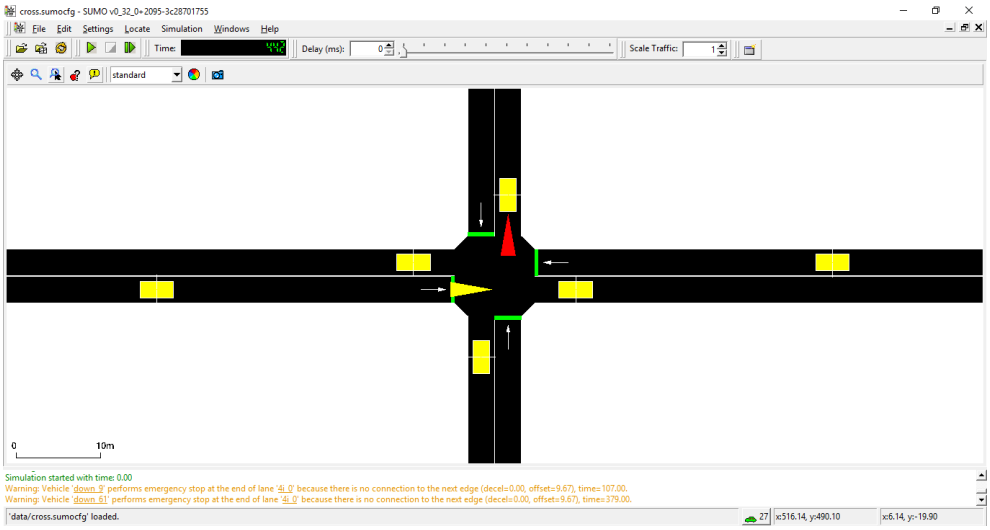


FIGURE 3.16: Advantages of the advanced single lane model (2)

### 3.2.4 Reinforcement learning in context of a triple lane model

The earlier sections demonstrate a simplistic intersection model to facilitate a robust interface between the SUMO simulator and reinforcement learning via the TraCI API and ensure its frictionless functioning. This section describes the simulator setup for a triple lane four – way model, its state space and reward characterization, importance of exploration and aims to demonstrate the ability of the proposed reinforcement Q – learning based model to handle a complex intersection scenario.

The intersection scenario modeled in the previous sections can be associated with real world scenarios but with lower likelihoods due to their limited complexity. In contrast, the intersection scenario modeled in this section can be often encountered in the real world. In order to replicate such an intersection, a junction consisting of two roads has been modeled in SUMO. Each road consists of three lanes indicating the three possible directions of travel namely – travelling to the left, moving straight along the road and turning right. A similar traffic distribution algorithm that has been utilized in the previous sections has been used for this intersection type. The trained agent or the controller in reality would be a part of the traffic junction infrastructure and the incoming and outgoing vehicles would communicate about their positions, velocities, intended directions of travel, etc. with this controller aiding in a system of systems level intersection control. In scope of our experiments, the communication with the agents regarding the incoming and outgoing vehicles is established by using detectors placed in all three lanes along the incoming and outgoing routes. The control of the vehicles by the system of systems level intersection controller starts and ends with the passing of the detectors.

#### 3.2.4.1 Simulator setup for the triple lane four – way intersection model

The figure 3.17 shows the modeled scenario of the triple lane four – way intersection with the detectors for the incoming and outgoing vehicles along each lane. The simulator related settings such as the green phases for all lanes across the junction or the type of junction as 'traffic_light' are similar to the previous models. The set

of settings necessary to simulate collisions in SUMO also are similar to the previous models. The SUMO configuration file and additional file including information about the detector locations along with the XML files for modeling the intersection can be found on the Github repository for this project on intersection control using reinforcement learning.

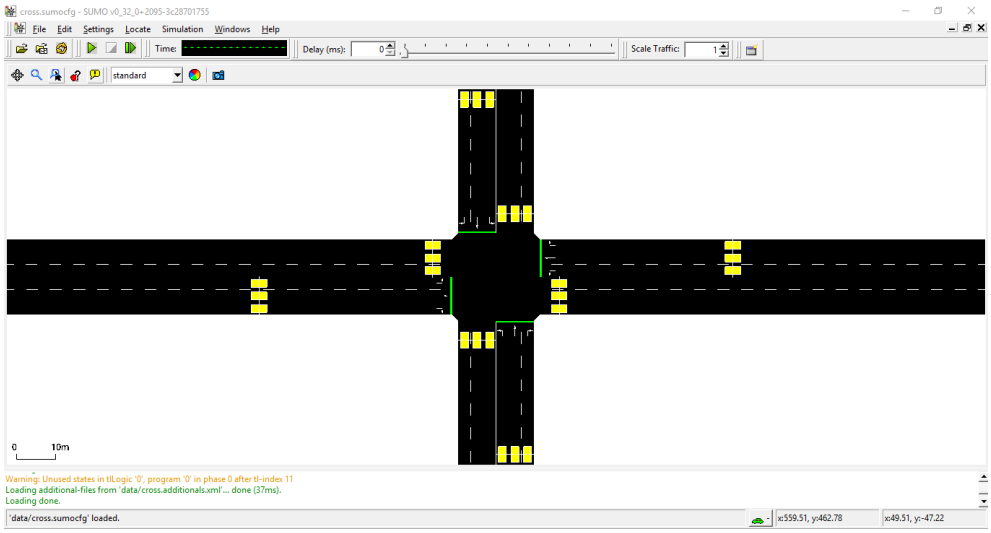


FIGURE 3.17: Triple lane four – way intersection model in SUMO

Since the number of vehicles approaching the junction in this model are much more in contrast with the earlier models, the detectors along the incoming routes are placed further away from the junction to aid in the decision making process of the agent (relative distance of 435 m – 450 m from the junction start). The detectors on the outgoing lanes are placed immediately after the junction to facilitate faster communication regarding the successful maneuver of the vehicle across the junction and remove it from the state space characterization.

One of the modeling aspects of the SUMO is the lane – changing model for multi – lane roads. The lane – changing model determines the lane choice on multi – lane roads and speed related adjustments with respect to lane changing. The major motivations of changing lanes during the maneuver are strategic lane changes to reach the next edge on the route, cooperative lane changing with the sole purpose of helping other vehicles with lane changing towards their target edge, tactical lane changing to avoid following a slow leader vehicle and remote controlled lane changing using the TraCI during the simulation. The lane changing model accomplishes two main purposes that are computing the change decisions of a vehicle based on the route of the vehicle and the current and historical traffic conditions surrounding the vehicle [Erd14]. With the default lane change model active, SUMO makes these lane changes implicitly based on the motivations stated above. However, in context of our project, the intended direction of travel of a vehicle is communicated to the agent through the incoming detector along the lane. For example a vehicle in the left most lane along the route west to east is detected by the incoming detector and communicates to the agent via the state space characterization that the intended direction of travel is to take a left turn and travel along the route to the north. Hence, the vehicles moving in the particular lanes along the course of the road should not be allowed to change their lanes as it might result in detection by a different lane

detector communicating a false intended direction of travel, but an altogether different actual route of travel. The default active lane changing model in SUMO can be deactivated using the TraCI during the simulation by explicitly disabling it for each vehicle as follows

```
running_vehicle_ids = traci.vehicle.getIDList()
for vehicleID in running_vehicle_ids:
if vehicleID not in lane_change_disabled_vehicles:
        traci.vehicle.setLaneChangeMode(vehicleID, 0b0100000000)
```

The remaining structure of the algorithm remains the same for the triple lane four – way model just scaled up to facilitate control over this complex intersection scenario.

FIGURE 3.18: A triple lane four – way intersection, Schwanseestraße, Munich

The picture 3.18 shot by us shows a real world triple lane four – way intersection scenario shot near the Giesing train station in the German city of Munich. The figure 3.19 shows the OpenStreetMap view of the same intersection for clear understanding.

#### 3.2.4.2 State space characterization for the triple lane four – way model

The state space characterization of the triple lane four – way model is similar to the advanced single lane four – way model from the previous sections. The state space characterization includes the exact position of the vehicles along each lane for a particular route in conjunction with the information regarding the presence or non – presence of the vehicle within a lane. Each route consists of three lanes namely – leftmost lane for vehicles intended to travel to the left, middle lane for vehicles to continue moving along the same path and the rightmost lane for turning right and travelling along the route to the right. The vehicles travelling along the rightmost lane for all routes do not intercept the junction area that is shared by all other vehicles during their entire course of trajectory. Since they do not intervene between the trajectories of any other vehicles during their maneuver, it is unnecessary for the intersection regulator or the agent to exercise control over vehicles in these lanes. The figure 3.20 below indicates the vehicles travelling along the rightmost lanes along

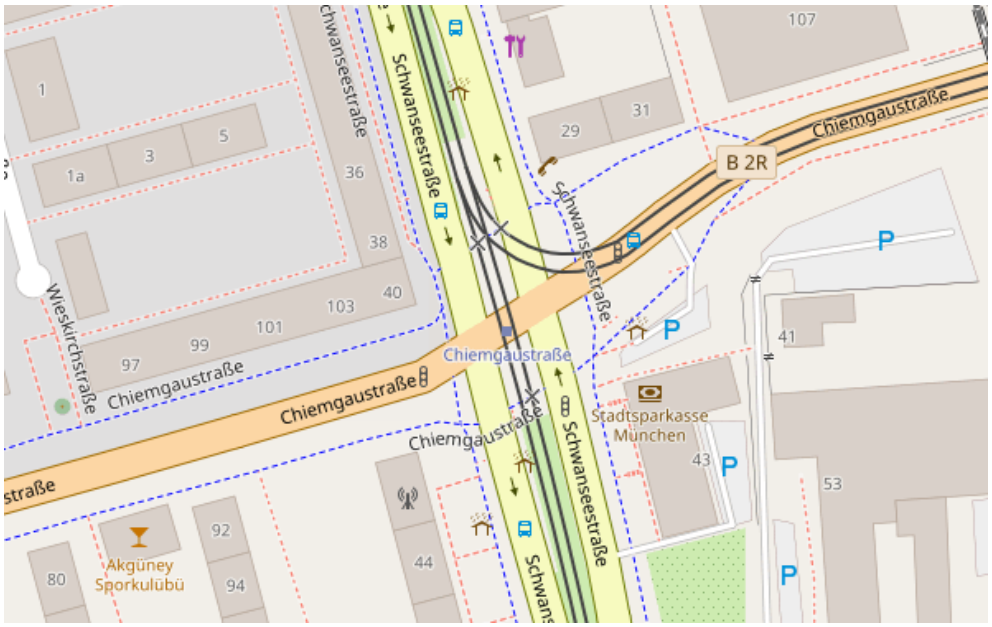


FIGURE 3.19: OpenStreetMap view of the intersection

the different routes without intercepting the trajectories of any other vehicle during the course of their entire maneuver. Thus the vehicles moving in the rightmost lanes for all the routes have been excluded from the state space representation.

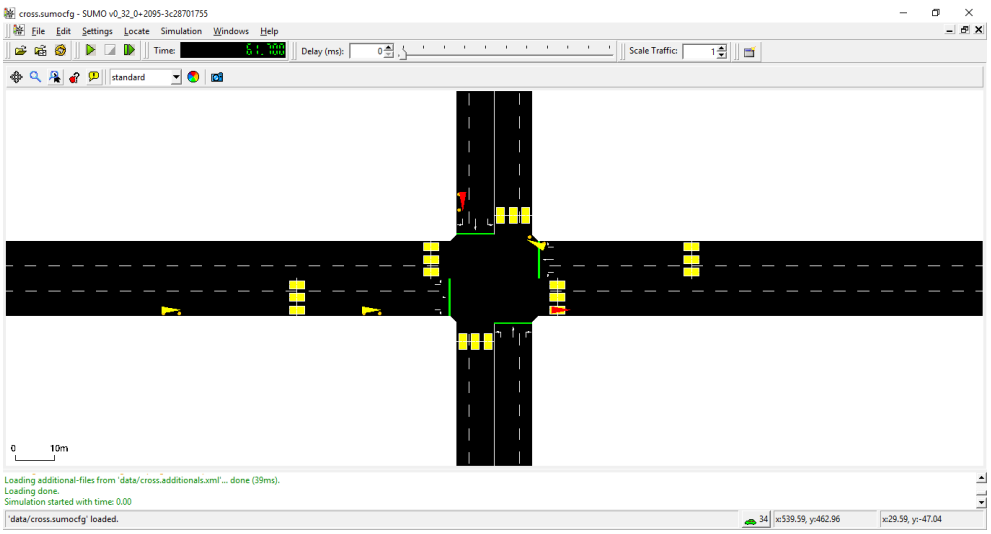


FIGURE 3.20: Vehicles travelling along the rightmost lanes for different routes

The state space representation thus consists of a sixteen element single row vector with the odd position elements starting from 1 indicating the presence or non – presence of a vehicle through a '0' or a '1' respectively within a lane along the route and the even position elements starting from 0 indicating the corresponding position of the vehicle along the lane in case of its presence within a lane for a particular route i.e. west to east, south to north, east to west and north to south respectively.

For e.g. the state [1, 442, 0, 0, 1, 468, 1, 4, 0, 0, 1, 485, 1, 453, 0, 0] represents that there is a vehicle in the leftmost lane along the route west to east at a relative distance of 442 with respect to the junction start i.e. between the incoming detector and the junction along the lane, there is no vehicle in the middle lane along the same route, there is a vehicle in the leftmost lane along the route south to north at a distance of 468 with respect to the junction start between the incoming detector and the junction, there is a vehicle in the middle lane along the same route at a relative distance of 4 from the junction start i.e. between the junction and the outgoing detector along the lane, there is no vehicle in the leftmost lane along the route east to west but one in the middle lane waiting just before the start of the junction at a distance of 485 (the first vehicle in each lane is requested to wait at a distance of 485 relative to the junction start before it receives a 'Go' command from the intersection agent), there is a vehicle in the leftmost lane at a distance of 453 along the route north to south and no vehicle in the middle lane along the same route. With such a state space characterization, **1.0395 x $10^{17}$ states** are possible i.e. **almost infinite diverse states** are possible and the agent can take eight different actions to release vehicles from the leftmost or the middle lanes along each route. Similar to the advanced single lane four – way model, the state space characterization could have been simplified by just appending the position of the vehicle along the lane resulting a smaller eight element single row vector, but the position as '0' exactly at the junction start for a vehicle can disrupt the comprehension of the state by the agent as explained in the section 3.2.3.1 and hence the presence or non – presence of a vehicle within a lane characterized by a '1' or a '0' respectively has to be included in the representation.

With infinitely possible diverse states, the Q – table cannot suffice and a neural network has been used to exercise control over the intersection as an agent as the action – values have to be approximated rather then an explicit Q – value. The neural network receives the current state i.e. the sixteen element single row vector as an input and generates the Q – values for the possible actions corresponding to the input state. Depending upon the exploration and exploitation criterion, the reinforcement learning model selects and executes the respective action.

#### 3.2.4.3 Reward function characterization for the system of systems level intersection control using reinforcement learning

One of the distinctive features of reinforcement learning is the usage of a reward signal to formalize the idea of a goal and this section rationalizes the selection of this reward and the punishment signal for our agent i.e. system of systems level intersection controller using reinforcement learning.

The reward or the punishment is a way of communicating to the agent what is to be achieved but not how is it to be achieved. The selection of the reward and the punishment are one of the vital factors that determine the learning progress of the agent. The reward setup is an explicit indicator of what is to be achieved. Hence, in context of our project, the agent receives a reward for an intersection maneuver of a vehicle and receives a punishment for collisions encountered during this maneuver. The quantization of these reward and punishment signals has been carried out through experimentation. During experimentation it has been perceived that the difference between the reward and the punishment cannot be too large and needs to be in a subtle range with respect to each other. During the initial learning phase

where multiple collisions are expected to occur, a too high punishment signal because of a particular action results in too large updates within the network or the Q – table. The agent tries to avoid taking the action that resulted in the punishment in the near future. The figure 3.21 below shows an example of such a behavior for the triple lane four – way intersection model. The negative reward or punishment has been chosen to be approximately 65 % more than the reward signal during the particular experiment. During the initial training phase, release of the vehicle from the middle lane along the route north to south resulted in a collision. Since the punishment signal is quite large in contrast with the reward signal and the cumulative rewards achieved because of this release action are not quite large, the large weight updates during the training phase within the network circumvent the selection of this action. It results in a long accumulated queue of vehicles within this middle lane along the route north to south and hampers the progress of the overall training process.

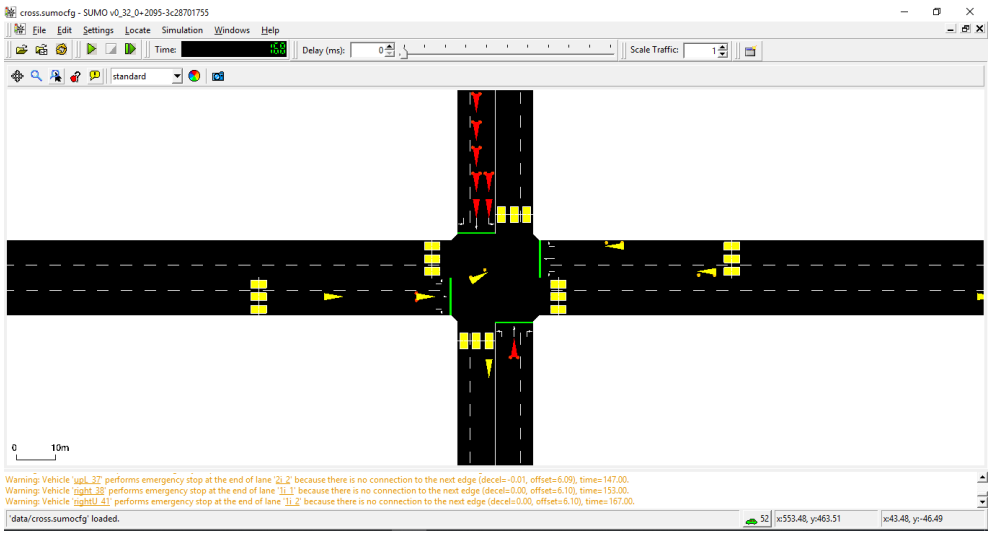


FIGURE 3.21: Long vehicle queues due to improper punishment signal characterization

Similarly, a too high reward signal in comparison with the punishment results in increased number of collisions, since the weights within the network are updated with a bias towards releasing the vehicles to increase the cumulative reward. After extensive experimentation it has been realized that for the reinforcement learning model in context of our project with the SUMO simulator, the difference between the reward and the punishment signals should be in the range of 50 – 60 % with respect to each other. The punishment signal has been characterized approximately 50 % more than the reward signal since the number of collisions encountered are quite less in comparison with the successful intersection maneuvers during a typical simulation run. Literature survey within the context of reward function selection also points out that the rewards should be more or less clipped within the range of [–1, +1] to control the scale of the resulting gradients during the network updates [Mnih+13][Cas17][Min09]. Experimental trials in context of our setup points out that a high reward selection but within the recommended range of the punishment signal results in blowing up of the weights of the network and proves to satisfy no particular purpose. Abiding to these guidelines, the reward signal is clipped to +1 and the punishment signal has been iterated within the specified range (50 – 60 %)

like a hyper parameter for different simulation runs i.e. between the range –1.5 to –1.6.

The SUMO simulator has a remarkable characteristic in context of monitoring vehicular collisions that aids in the training process of the agent in context of our project. The code snippet shown below is used to retrieve the number of vehicles involved in the collision and the IDs of the vehicles involved in the collision.

```
R_ = traci.simulation.getCollidingVehiclesNumber()
R_List = traci.simulation.getCollidingVehiclesIDList()
```

In case of a collision the getCollidingVehiclesNumber( ) method retrieves a different value depending upon the type of collision encountered. A direct intersecting collision between vehicles returns a higher value than a normal brush – up collision between vehicles. With respect to the minimum gap factor between vehicles, SUMO reports a collision if vehicles are within this minimum gap of each other, however with a lower value. The figure 3.22 indicates the different type of collisions occurring during the simulation run returning different values through the retrieval method.

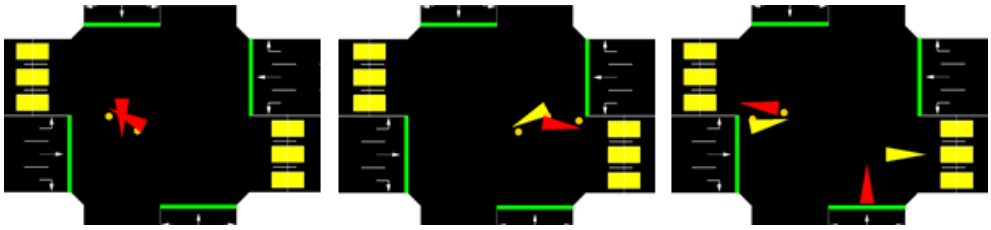

FIGURE 3.22: Intersecting, brush – up or collisions because of the minimum gap factor

In case of the intersecting collision shown in the first figure from figure 3.22, the method returns a value of 12, in case of the brush – up collision shown in the second figure, the method retrieves a value of 8 and the last figure from where SUMO reports a collision since the vehicles are within the minimum gap of each other, the method returns a value of 6. This retrieved value is in the range between 6 to 12 and is used to calculate the final punishment signal for the Q – algorithm that aids in the update of the weights of the network. The code snippet below indicates this calculation of the punishment signal for the update.

```
R_ = traci.simulation.getCollidingVehiclesNumber()
collision_division_factor = 3.5
collision_factor = -1
R_ = R_ * collision_factor / collision_division_factor
```

The 'collision_division_factor' is a hyper parameter that is tuned to suit individually for each simulation and the traffic distribution for the simulation run. The model thus receives a punishment depending upon the severity of the collision and primarily learns to minimize the severe collisions.

In context of the cumulative reward earned, it is important to note that the total reward earned during an episode of the simulation run cannot be used as a metric to gauge the performance of the reinforcement learning agent. One of the main reasons being that the vehicles maneuver the intersection inspite of the collisions or the type of collisions and receive a reward and hence the cumulative reward received at the

end of each episode is approximately within a range. Also, the cumulative reward depends upon the number of vehicles running during the simulation run that is characterized by the traffic distribution. This traffic distribution is modified within a pre – defined range after every episode to train the agent against diverse scenarios. Thus the learning progress of the agent or the performance of the agent cannot be determined on the basis of the cumulative reward at the end of each episode and may only provide a rudimentary idea.

#### 3.2.4.4 Exploration versus Exploitation in context of the reinforcement learning agent for intersection control

The frequent updates by the Q – algorithm during the training phase aims to maintain estimates of the action values and aids in the selection of at least one action whose estimated value is the maximum in contrast with the others. The selection of these actions is termed as greedy actions and the agent exploits its knowledge of the action values. If the agent selects one of the non – greedy actions, then it is said to explore, since the action value of the selected non – greedy action was not known before and the agent tries to improve its estimate. This section clarifies the importance of exploration in contrast with exploitation in the context of our system of systems level intersection control. In order to maximize the reward for the current step, exploitation is a viable option but exploration may produce greater total reward in the long run [Sil15]. This condition arises since the agent certainly has the knowledge about the greedy action, but numerous other actions might be estimated to be nearly as good but with a substantial amount of uncertainty within it. In the longer run probably one of these uncertain actions might result in a higher reward and hence striking a balance between exploration and exploitation conflict is one of the important characteristics of the reinforcement learning process.

In the context of the single lane four – way model with a much simpler state space characterization, i.e. the agents trained for the simplistic state space characterizations described in sections 3.2.1. and 3.2.2. since the diversity within the states is limited, the $\epsilon$ greedy action selection can be used. The greedy actions are chosen with a probability of $1 - \epsilon$ and random actions or actions with a purpose of exploring are chosen with a probability of $\epsilon$ where the exploration factor 'Epsilon' is generally set in the range 0.01 to 0.15. The value of the exploration factor remains constant throughout the entire training process.

However with complex state space characterizations such as the advanced state space characterization that accounts for the exact positions of the vehicles, exploration is highly critical and has been used for agents trained and described in the sections 3.2.3. and 3.2.4. In case of the single lane four – way model with advanced state space characterization or triple lane four – way model, the neural network determines the selection of the actions by providing the action value estimates at the output. The network is initialized with very small weights and without the elapse of a finite training period; they are unable to aid in the selection of an action. Exploration proves to be extremely useful during this initial training phase and helps the network accumulate finite amount of diverse experiences that might be rewarding or punishing because of collisions. These experiences are also saved in the replay buffer and might be retrieved at later stages during the training to aid in the stability of the network. For example during the initial training period, with a state space as – [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 485, 0, 0, 0, 0] the reinforcement learning model receives a

reward for the release action from the middle lane along the route from east to west (Lane3S) and updates the weights to output high Q – values for the release action. With the subsequent state as [1, 462, 0, 0, 0, 0, 0, 0, 0, 0, 1, 5, 0, 0, 0, 0] the Q – values may be high in favor of the same release action because of the tuned weights i.e. releasing the vehicle from the middle lane along the route from east to west during the initial training phase. With exploration, the choice of the release action from the leftmost lane along the route west to east (Lane1L) helps the network cognize that the release action 'Lane1L' with a vehicle positioned at 462 is later rewarding and does not result in a collision with a vehicle positioned at 5 i.e. junction area ahead of the middle lane along the route east to west.

The exploration factor has been maintained quite high and reduced over time as the training phase progresses. With a value of 0.6 for EPSILON at the start of the training i.e. selection of 40 random actions every 100 actions, it is increased marginally at each time step as the simulation advances. During each simulation step, the 'numpy' uniform random function is used to generate a random float value between 1 and 100. If the random number is greater than the value of EPSILON at the particular time step, a random release action is selected at the particular time step. In order to improve the effectiveness and utilize exploration opportunities at their fullest; a combination of while and if – else conditions is used to select a random release action for lanes which are characterized by presence of vehicles. Without this function, random actions are often selected for lanes that do not have vehicles in it and result in a loss of the exploration opportunity that has been provided to the model. With the additional function in use, the number of accumulated experiences during the initial training phase increase approximately by 20 %, thus aiding in faster training of the network. The code snippet below shows the implementation for this functionality. The exit through the while loop is only possible by randomly selecting a release action for a lane in which vehicles are or have been requested to wait, thus avoiding the possibility of selection of a random release action for an empty lane.

```
if np.random.uniform() > EPSILON:
  if (vehicles_waiting_1_S or vehicles_waiting_1_L or
      vehicles_waiting_2_L or vehicles_waiting_2_S or
      vehicles_waiting_3_S or vehicles_waiting_3_L or
      vehicles_waiting_4_S or vehicles_waiting_4_L):
      while(1):
        ran_int = np.random.randint(0, 8)

          if ran_int == 0:
            if vehicles_waiting_1_L:
              A = 'Lane1L'
                break

          . . . . . .

       else:
      A = np.random.choice(ACTIONS)
```

#### 3.2.4.5 Modified state space characterization for the triple lane four – way model

The characterization of the state space for the triple lane four – way model explained in the section 3.2.4.2 above cannot ensure a good performing model and the state

space characterization has to be modified. This section attempts to justify the reasons for the poor performance of the model with the state space characterization put forward in the previous section.

A neural network has been used to control the intersection for the triple lane four – way model due to the diversity among the states. Multiple simulation runs and extensive experimentation has been carried out to explore a good performing model displaying the ability to learn from the experiences (trend of reducing number of collisions over time). The reinforcement learning model setup in conjunction with the network architecture involves approximately around 34 hyper parameters that affect the simulation and the training process. With the state space characterization with the even elements starting from 0 indicating presence or non – presence of the vehicle within the lane and the odd elements indicating the precise positions shows learning inability and no signs of convergence. Multiple architectures and activation functions have been tested together for different sets of hyper parameters. The use of the replay buffer also does not stabilize the network. A few insights regarding the inability of the network to perform have been gained during debugging of the training sessions. The state space characterization for the triple lane four – way model is a sixteen element single row vector. The characterization of the state space is one of the reasons for the instability of the weights in the network during back – propagation. The position of the incoming detector along the lane is a hyper parameter that is individually tuned for each simulation run. The position of the incoming detector is varied between the range 435 – 450 relative to the start of the junction. For example with the state space as follows – [1, 442, 0, 0, 1, 468, 0, 0, 0, 0, 1, 485, 0, 0, 0, 0]; the even positions have either a '0' or a '1' and the odd positions may have an integer value between 440 m (position of the incoming detector) to 485 m (for vehicle between the incoming detector and the start of the junction) or between 0 m to 15 m (for vehicle between the start of the junction and the outgoing detector). Such a state space with a huge difference between the possible values along with a sudden disruption because of the '1' or '0' leads to indifferent weight updates and no signs of convergence of the loss function for the weights for several different architectures. The network is unable to consistently approximate the action – values for such an input, since the number of individual weights to be tuned are too high and the state space has a sudden disruption within its representation. The similar state space characterization with a single row eight element vector delivered good performance for the single lane four – way model, however with the sixteen element single row vector for the triple lane four – way model it is unable to yield performance.

The state space characterization thus has to be modified to ensure a better model in contrast with the previous one. The state space has been reduced to a single row eight element vector by accounting only for the positions of the vehicles, if present, in the respective lanes. The presence or non – presence of the vehicle within the respective lanes is not represented through a '1' or '0' and the presence of the position is itself indicative of it. Such a characterization introduces a few drawbacks which have been explained in the section 3.2.3.1.

The experiments with the modified state space characterization yields better results in comparison with the previous state space representation and have been summarized subsequently. The modified state space is indicative of the position as well

as the presence or non – presence as for example – [442, 0, 468, 0, 0, 4, 0, 0] indicates a vehicle positioned at 442 in the leftmost lane along the route west to east, a vehicle positioned at 468 in the leftmost lane along the route south to north and a vehicle between the junction and the outgoing detector along the middle lane along the route east to west. Depending upon the network architecture, the state space also has been divided by an equal factor before it is received by the network as an input, since probabilities are required at the output i.e. a number between 0 and 1. With a small network, the weights tend towards 0 and hence the dividing factor tries to prevent this phenomenon. After several experiments with the network architecture and the approximately 34 hyper parameters, a descent performing model has been attained. The model has been trained for total of 60,000 simulation steps with 20 episodes of 3,000 steps each. During each episode, the traffic distribution has been varied within certain pre – defined limits. The network architecture consists of 8 neurons at the input, one hidden layer with 16 neurons and 8 neurons at the output. The linear activation function has been used for the intermediate layers and the softmax function has been used at the output layer, indicative of the release action with the highest probability of being selected. The selection of the activation functions has been elucidated in detail in the section 3.2.3.3.

The initial Epsilon value has been set to 0.6 and marginally increased after each simulation step upto 0.99 similar to the single lane four – way model illustrated in section 3.2.3.4. The learning rate and the discount factor are 0.001 and 0.9 respectively. The loss for the experiment has been defined as the updates for the model for the output Q – values i.e. the difference between the target and the predicted Q – values that takes place during the training process. The loss fails to converge to a 0 value and reaches approximately around 0.4 towards the end of the simulation consisting of 60,000 steps. This is the lowest value the loss function attained towards the end even for increased number of episodes and hence the training has been stopped after 60,000 steps. The training process attains negligible or zero loss for several steps during the simulation, however fails to converge. The subsequent section 3.2.4.7 tries to identify a few probable reasons for the non – convergence. The experience replay stored the experiences as they have been collected along the training process and the agent or the network has been trained by randomly sampling them in batches to provide stability to the network. This stability can be seen in the figure 3.23 where the number of peaks or very high amplitude oscillations reduce towards the end of the training process. The hyper parameters pertaining to the replay buffer and the training intervals also have been tuned individually to suit the simulation setup. For this experiment, the batch size is 64 and the update frequency has been set to 5000 steps after 20000 initial or pre – training steps. The figure 3.24 displays the gradual increase in the epsilon value and a constant value of 0.99 as the training process progresses.

The time step of the simulation for all the experiments has been set to 1 second. SUMO has a few issues which limit its stability in context of collisions since it has a collision free inherent nature and the methods to retrieve collisions also have been a part of the development version and not the release version indicative of the fact that they are still under development [Wu+17]. Debugging the errors is also not possible in a few cases since the errors are not reproducible. This point has been explained in detail in the section on modifications of the SUMO simulator. Also, SUMO's car following models are calibrated for a time step of 1 second and their behavior for smaller time steps produces unnatural behaviors [Wu+17][DLRnda]. The maximum

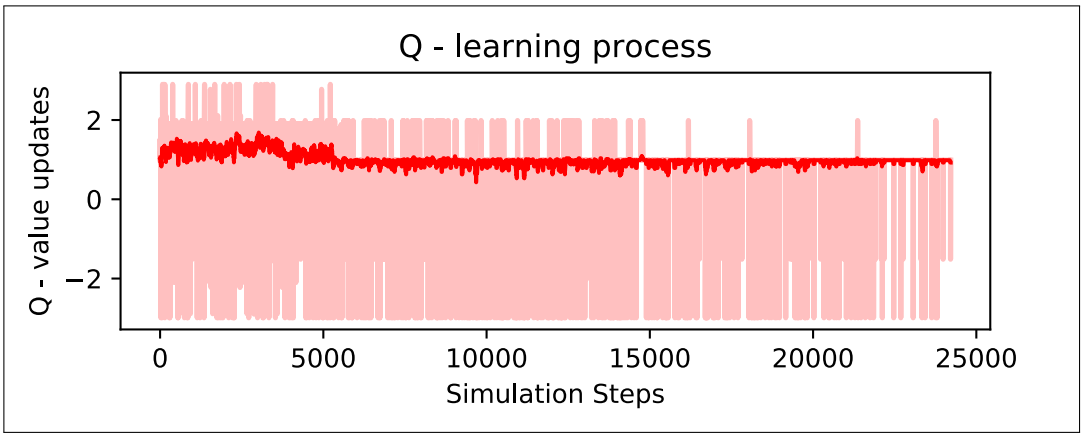


FIGURE 3.23: Convergence of the Q – value updates for the triple lane model

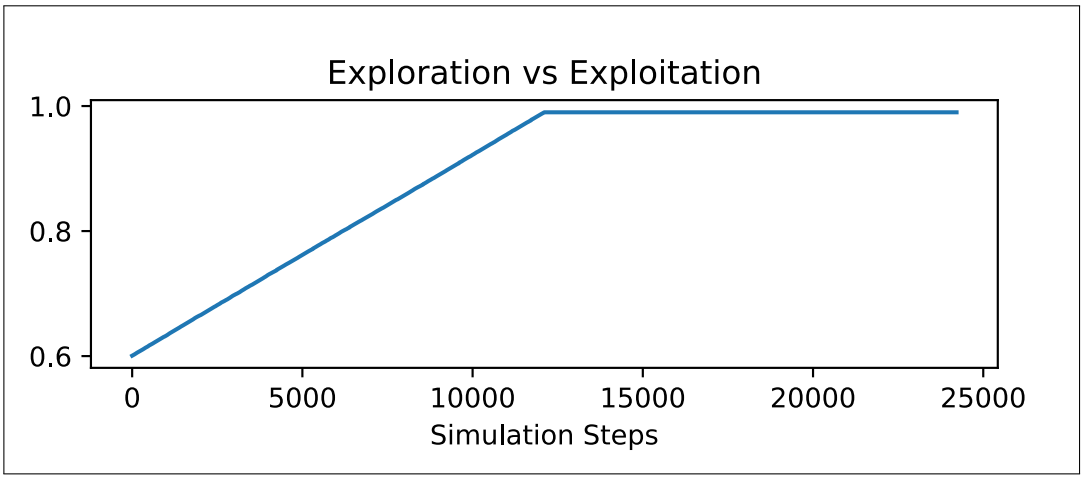


FIGURE 3.24: Exploration versus Exploitation against update simulation steps

number of collisions (91) have been encountered during the 7th episode and minimum number of collisions (30) have been encountered during the 16th episode. The model initially releases the vehicles in order to increase the cumulative reward but the occurring collisions help it comprehend the erroneousness of such actions and the waiting time for the vehicles goes on increasing as the training progresses. The waiting times for the vehicles goes on increasing from 942 seconds for the first episode to a maximum of 1405 seconds during the 12th episode. The CO2 emission decreases from 21.57 kilograms during the first episode to a minimum of 15.76 kilograms during the 5th episode. After the training process, the network has been subjected to a test episode of 3000 steps with a random traffic distribution varied between the pre – defined limits like during the training phase, i.e. a completely new traffic distribution. The validation or the test episode encounters 26 collisions. These performance metrics for the experiment have been benchmarked in the next section against the conventional approach.

#### 3.2.4.6 Comparison between the reinforcement learning and conventional traffic light approach for the triple lane four – way model

Efficient management of traffic is still a challenge and many researchers and engineers are trying to approach the problem in diverse ways. The effectiveness of these approaches can be demonstrated by benchmarking them against the conventional or in – use approaches. Within the scope of this project, we try to improve the traffic flow using reinforcement learning. This section compares the system of systems level intersection management based on reinforcement learning approach to control intersections with the conventional traffic lights control for the triple lane four – way model.

The single lane four – way model has not been compared with its conventional traffic light control counterpart since, such intersection scenarios are found with a limited likelihood. However, the triple lane four – way approach is a very common intersection scenario and hence has been benchmarked against its conventional counterpart. The intersection shown in the figure 3.18 in the above sections is an example of such a typical scenario. The algorithm and the phase durations during the peak traffic hours for controlling the intersection shown above using the conventional traffic lights have been recorded and attempted to be modeled and replicated in SUMO. For the intersection referenced above, we tried to approximate the algorithm by recording the phases and their corresponding durations for different directions. During the peak traffic hours, the green and the red phase durations are approximately equal to 45 seconds for each direction. SUMO offers multiple functionalities in context of intersection control. With the traffic light based control, a basic algorithm is available by default that can be extended and parameterized to adapt to application specific scenarios. The control algorithm has to be programmed in the SUMO .net.xml file and the snippet below exhibits the control logic attempted to be replicated.

```
<tlLogic id="0" type="static" programID="0" offset="0">
        <phase duration="45" state="GGrGrrGGrGrr"/>
        <phase duration="5"  state="GyrGrrGyrGrr"/>
        <phase duration="45"  state="GrGGrrGrGGrr"/>
        <phase duration="5"  state="GryGrrGryGrr"/>
        <phase duration="45" state="GrrGGrGrrGGr"/>
        <phase duration="5"  state="GrrGyrGrrGyr"/>
        <phase duration="45"  state="GrrGrGGrrGrG"/>
        <phase duration="5"  state="GrrGryGrrGry"/>
</tlLogic>
```

Vehicles in the diagonally opposite lanes intended to travel in the similar directions with respect to the lane are given a green signal at the same time. With respect to the nomenclature established in the scope of our project, for e.g. vehicles in the leftmost lane intended to turn left along the routes west to east and also east to west are given a green signal at the same time. The vehicles in the rightmost lane intended to turn right along all the routes are given a green signal during the benchmarking process and also subsequently programmed in the .net.xml file for SUMO. However, in reality they are dependent on the phases of the pedestrian signals which have not been accounted within the scope of our project.

The validation episode or the test episode for the reinforcement learning model consists of 3000 simulation steps. It has been benchmarked with the conventional

traffic light scenario for same simulation duration and for the same traffic distribution. The figure 3.25 is a screenshot during the simulation for the modeled intersection scenario using conventional traffic lights. It is important to note that the incoming and the outgoing detectors in case of the conventional traffic light simulation have been used to retrieve the values of the waiting times and emissions for the vehicles.

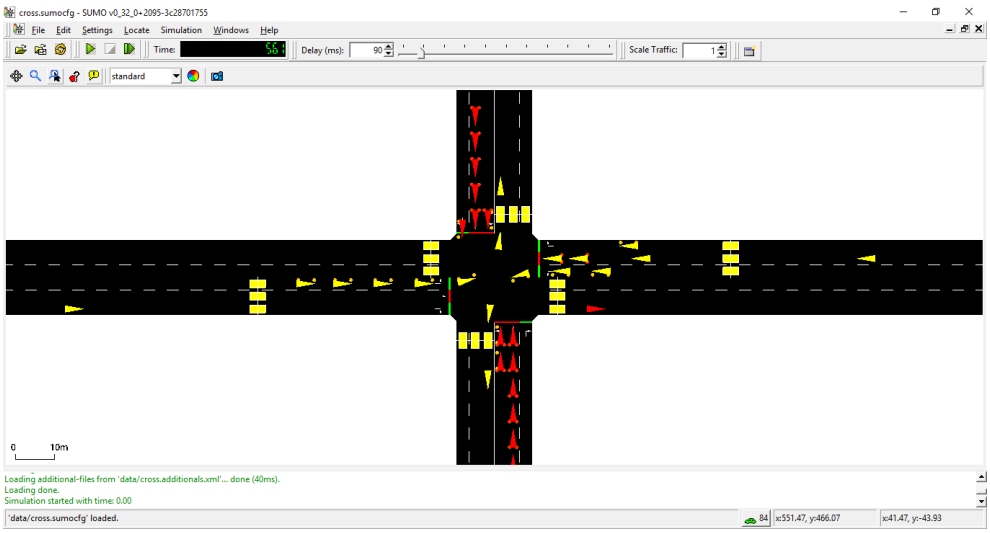


FIGURE 3.25: Triple lane four – way model with conventional traffic lights for intersection control

The fixed – cycle traffic lights, also referred as pre – timed control have fixed durations of the red, yellow and green phases. This may lead to long waiting times for vehicles at one side of the crossing even when there are no vehicles maneuvering the intersection on the other side of the crossing. The proposed reinforcement learning system of systems level intersection controller attempts to reduce this waiting time and thereby the $CO_2$ and other greenhouse gas emissions. The table 3.7 summarizes the performance of the reinforcement learning based approach and the conventional traffic controller.

TABLE 3.7: Comparison between the reinforcement learning and conventional traffic controller

| Approach | Waiting Time (s) | $CO_2$ Emission (kg) | Number of Collisions |
|---|---|---|---|
| **Conventional Traffic Lights** | 3383236.00 | 192.103 | 0 |
| **Reinforcement Learning Controller** | 613.00 | 19.280 | 26 |

The run – time parameters such as the waiting times for the vehicles or the greenhouse gas emissions for the vehicles can be retrieved during the simulation in SUMO through the TraCI interface. The methods getWaitingTime( ) and getCO2Emission( ) belonging to the 'traci.vehicle' class have been used to retrieve the values for all the vehicles between the incoming and the outgoing detectors during the simulation. The reinforcement learning based intersection controller reduces the waiting time by almost around 99.98 % thus improving the efficiency of the traffic flow. The reduced waiting time also thereby reduces the CO2 emissions by almost 89.96 % thus reducing the greenhouse gas emissions and ensuring a sustainable environment. However, the number of collisions occurring with the reinforcement learning based controller are a major drawback of the system. The safety and reliability of

artificial intelligence for applications concerning safety critical systems is still a developing concept and would require further years of research.

Since the agent has been trained to comprehend the imprecise release actions through the collisions occurring, it may not be assured that the number of collisions after the elapse of certain training period can be reduced to zero. One of the viable options would be to allow the agent to control the intersection until a collision is encountered. The moment a collision is encountered, the episode is terminated and a new one is started. The performance of the agent can be evaluated based on the time until which the agent could control the intersection without a collision. However, it may again depend on the traffic distribution for each episode and thus may only provide a rudimentary idea of the agent performance. Another method evaluation would be based on the number of vehicles controlled by the agent at the intersection without a collision. However, even this may not serve as a metric to evaluate the agent performance, since the number of vehicles generated during the simulation run also depend on the traffic distribution. Thus, further research and use of further complex methods to train the agent to ensure a collision free application scenario is needed in this context.

#### 3.2.4.7 Traffic distribution dependent network architecture and probable reasons for the non – convergence of the network policy

During the experimentation in context of the single and the triple lane four – way model, a few insights have been gained pertaining to the network architecture. This section describes these findings and also attempts to put forward a few probable reasons for the non – convergence of the loss for the networks that are representative of the optimal policy.

The network structure for a good performing model also depends upon the overall traffic distribution. Depending upon the overall distribution, a suitable network structure can be determined after thorough experimentation. This involves modifications within the network structure such as the number of neurons and the number of layers within the network or tuning the hyper parameters related to the network. As seen in the previous sections, very high amount of different states are possible for the state space characterization used for the single lane or the triple lane four – way model. But if the frequency of the traffic is high, vehicles are made to wait at the start of the junction before being issued a release action from the agent or the vehicles go on lining up behind one another in order to avoid collisions. Since the state space characterization is such that it represents the vehicle position along the lane in case of its presence; in case of piling up traffic (vehicles waiting at 485 i.e. start of the junction for the triple lane four – way model), the different number of possible states as an input to the network are restricted and overall limited number of diverse states are encountered. After the first vehicle crosses the outgoing detector, the subsequent waiting vehicle again represents itself in the state space with the similar positions. Eventually the network learns to handle randomness within the traffic distribution, since it is changed for each episode (within a pre – defined limit), but there is always a better network architecture and better set of hyper parameters that may perform better than the previous one. It is not the case that a particular architecture cannot handle some indifferent traffic distribution, but there is no best architecture and set of hyper parameters that are robust against all the traffic distributions and offer better performance. It has been observed that with a very high traffic distribution, a

smaller network is better performing since the number of diverse states are limited, however, then it performs well only for a certain type of traffic distributions due to the limited learning diversity. With a sparse traffic distribution, the opposite is true.

One of the viable solutions against this is to enhance the state space characterization by accounting for all the vehicles and their corresponding positions along a lane in the state space representation. But this results in dynamically changing state space dimensions. For e.g. if the lane can accommodate five vehicles in the space between the input detector and the start of the junction, these five vehicles along with their relative positions can be represented in the state space for this particular lane. However, if the first vehicle is released from the waiting line and a new vehicle joins the waiting line at the rear end, it may disrupt the complete state space representation. The knowledge about the vehicle positions in the common junction area is highly important to avoid collisions. If we decide to account for only the first five vehicles within the section between the input and the output detectors along the lane, the remaining vehicles along the lane might result in the same issue that was encountered with the previous state space characterization. If we decide to characterize the state space in such a way that it accounts for the maximum number of vehicles that can be accommodated between the input and the output detectors along a lane, it may not be possible. Since if 10 vehicles might fit in this section between the input and the output detector, there might be a case when even 11 vehicles might fit in this section – when the first and the last vehicle covers half area of the input and output detector (i.e. the vehicles are directly on the detector covering half area of the detector). This might change our state space dimensions dynamically depending upon the traffic situation. Since neural networks are incapable of handling varying input dimensions, a different solution has to be thought of for this particular issue. One of the viable solutions is the use of space reservation based state space characterization rather than vehicle based one. The entire course of the lane from the incoming detector to the outgoing detector is split into several sections and the state space representation accounts for the occupancies within these sections. The only drawback of such a system would be the necessity to have detectors across the entire course of the route that may pose a threat for its wide spread adoption from the economic perspective.

The single step off – policy temporal difference learning i.e. Q – learning has been used for the training of our intersection management system. As seen in the section on Q – learning above, the learned action – value function, Q, directly approximates q*, the optimal action – value function, independent of the policy being followed with the updates defined as

$$Q(S_t,\ A_t) \leftarrow Q(S_t,\ A_t) + \alpha\left[R_{t+1} + \gamma\, max_a Q(S_{t+1},\, a) - Q(S_t,\ A_t)\right]$$

The policy affects the training since it determines which state – action pairs are visited and updated. In order to ensure convergence, it is necessary that all pairs are continued to be updated and that majority of the state – action pairs are visited an infinite number of times. In case of our state space characterization along with the established interface between reinforcement learning and the SUMO simulator via the TraCI API, it may not be possible to visit even majority of the state – action pairs infinite number of times. It also depends upon the traffic distribution during the training phase. A reinforcement learning based value approximation approach involves a close – loop system in which the target value at step $t + 1$ is biased by the

training at steps $t$, which can amplify the drifts, thus ruining the learning as the distance between the desired value for Q and the obtained value differ more and more. In context of our networks, the early reduction in the updates size is indicative of the learning trait pursued by the agent, however, later on it does not drop down to a '0' value (constant at approximately around 0.4 for the triple lane four – way model). The reduction in the number of collisions over a period of time during the learning process, irrespective of the network architecture and the hyper parameter values is indicative of the fact that the model gets an overall mapping of the actions to be taken in a particular state. But a new state encountered every time does not drop down the magnitude of the updates during the learning process. Non – convergence of the update magnitudes or informally known as the Q – loss when the data keeps changing during the policy updates is a phenomenon often encountered [Fis16]. It is not similar to supervised learning where the data never changes and we can make multiple passes on the data to make sure that the weights are well – fitted with that data. In case of a similar reinforcement learning project in context of traffic flow optimization using the SUMO simulator, similar instability and non – convergence has been displayed by Q – learning because of the large state space diversity [Cas17]. The non – convergence because of lack of multiple visits for the action – value pairs might suggest that the algorithm needs more training time to converge. However, the norm of the gradient used to carry out the updates decreases over time and stagnates, thus indicative of the fact that the algorithm has reached a stable point and the results might not further improve. Further study is required in order to assess this topic and the subsequently needed convergence improvement techniques. One of the thought of solution is the use of policy gradient, rather than the Q – learning algorithm to improve the convergence, since policy gradient maps a state directly to an action to arrive upto a policy, i.e. parameterizes the policy. It has better convergence properties and is effective in high dimensional spaces [Sil15].

# Chapter 4

# Modifications of the SUMO Simulator in context of the project

The concept of system of systems level intersection management based on reinforcement learning has been attempted to be demonstrated through this project. The above sections aid in the comprehensive understanding of the concept starting from a fundamental approach involving the single lane four – way model along with the Q – tables to a complex triple lane four – way model regulated by a Q – network. However, a few limitations of the SUMO simulator have been faced within the context of this project and a few of them have been resolved by violating the underlying models of the SUMO simulator. These modifications in the SUMO simulator have been undertaken to facilitate the use of the SUMO simulator as an environment for reinforcement learning. This section illustrates these limitations, their corresponding reasons, the problems these limitations pose to our concept, the proposed solutions or modifications and the implications of these solutions or modifications.

## 4.1 Implicit and / or explicit braking of vehicles by the simulator to avoid collisions

The SUMO simulator has been designed principally to be a collision free simulator [DLRndb]. Hence the inherent nature of SUMO is to try and avoid collisions to the maximum extent by taking implicit and / or explicit and self – driven actions such as hard braking of vehicles. However, not all collisions are avoided by SUMO and some of them can be explicitly seen along with a warning in the SUMO console. The figure 4.1 indicates a collision occurring at the intersection due to a tandem unsafe release from the right i.e. east to west and the bottom lane i.e. along the route south to north. During this collision, the SUMO engine does not try to avoid collision by emergency braking of one of the vehicle. A collision warning helps the reinforcement learning algorithm to interpret the action as an unsafe action and trains itself accordingly. However, contradictory to the behavior displayed in the figure 4.1, in some cases, the SUMO simulator avoids collisions inspite of an unsafe release by TraCI by implicit emergency braking of one of the vehicle. Hence no collision warning is given out in the SUMO console. The figure 4.2 demonstrates this implicit emergency braking behavior for the vehicle approaching from the top lane i.e. along the route north to south to avoid a collision with the vehicle maneuvering the intersection from the right lane along the route east to west even when the vehicles have been explicitly released from their respective lanes by the reinforcement learning model via the TraCI API.

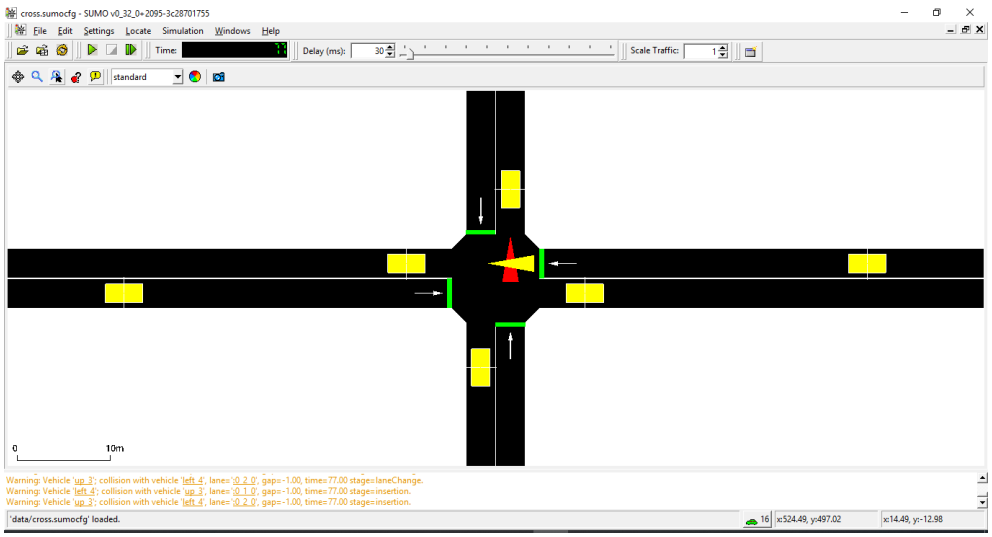


FIGURE 4.1: Vehicle collision monitored by SUMO and the subsequent warning notifications

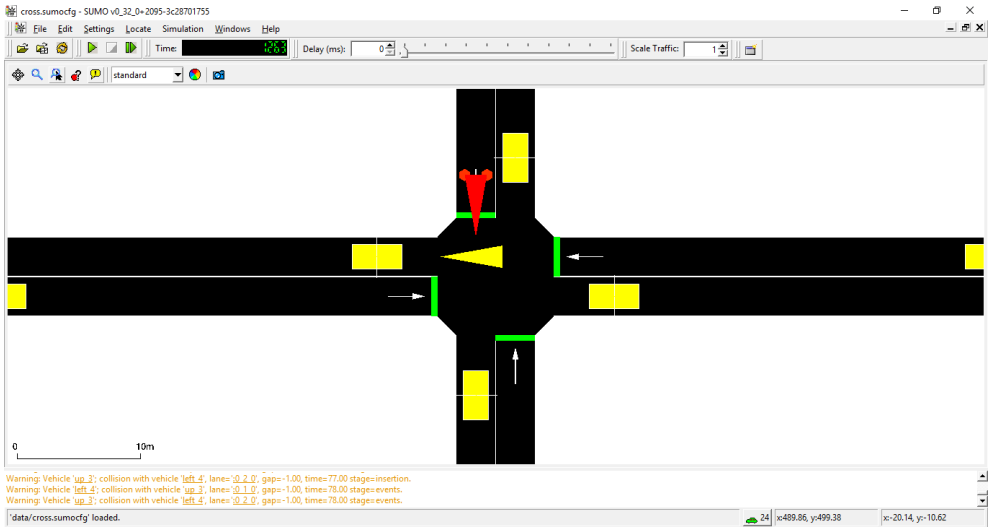


FIGURE 4.2: Implicit braking of the vehicle to avoid the collision

**Reason for Implicit and / or explicit braking of vehicles by the simulator to avoid collisions:**

The core idea behind development of such simulators such as SUMO is to optimize the traffic light switch times using sophisticated algorithms for mining out higher traffic flow efficiencies. However, the underlying assumption behind the development is that the algorithm at any point of time should separate conflicting streams of vehicles, and thus avoid collisions. The conventional traffic light system assures this separation between conflicting streams even with its highly simple functioning. The developers at DLR who created the SUMO simulator have not accounted for this vague behavior in case of collisions, since basic safety (prevention of vehicle collisions) can never be a performance criterion as it is quite trivial to ensure it [DLRndb]. Hence the simulator has been imparted an inherent nature to avoid collisions.

**Difficulties posed by the limitation in the context of the project:**

This is a grave issue in the context of our project pertaining to evaluation of cooperative intersection control for autonomous vehicles using reinforcement learning. We train our model or network based on the principal notion of assigning rewards for a successful maneuver of an intersection and a punishment or a negative reward in case of collisions caused by unsafe release actions. Inability to capture occurring collisions inspite of unsafe releases of vehicles may result in improper training of the entire model since the model fails to comprehend the release action as an unsafe one. This improper training might not ensure the results we want to extract from our concept a reinforcement learning based system of systems level intersection controller.

**Proposed modifications to confront the limitation:**

The implicit and / or explicit braking by the SUMO simulator in case of collisions can be avoided by setting the speed mode command in SUMO. The command has a 5 bit setting with each bit having a pre – defined function. Setting it to 0 disables all checks and overrides the basic behavior of the SUMO simulator i.e. prevention of collisions. However, in conjunction with the speed mode command, the speed of the vehicle also has to be set using the 'set speed' or 'slow down' commands, where the first one sets the desired speed of the vehicle instantaneously and the latter increases or decreases the speed of the vehicle gradually to the desired value and in the specified amount of time. The unwanted braking behavior can be avoided and the collisions can be registered only after these settings have been carried out.

**Implications of the proposed modifications to confront the limitation:**

In case of our project, the default velocity of all approaching and outgoing vehicles has been set to a constant value of 10 m/s. Using the 'slow down' command, the speed of the vehicle that has to be released from a preceding stop near the junction can be increased gradually to a pre – defined desired speed within a pre – defined amount of time. However, this often results into rear end collisions of the vehicles within the same lane. Using the speed mode and slow down commands in conjunction disables / violates the default car follow model of the vehicles resulting in rear end collisions within the same lane. Such a scenario is often encountered when a vehicle at a stop near the junction within a particular lane has been released and the subsequent approaching vehicle in the same lane also has been released at the next time step without a preceding stop action. The latter vehicle already has a velocity of 10 m/s and the leading vehicle increases its velocity gradually upto 10 m/s. Such same lane collisions are registered by SUMO and interpreted by the reinforcement learning model as an unsafe release action, thus training the entire model in an inappropriate manner. The figure 4.3 shows a typical same lane collision scenario arising out from these settings. The leading vehicle moving downwards from the top lane has been initially stopped near the junction. At a particular time step, the model takes the release action on it and its velocity is gradually increased to reach a desired velocity of 10 m/s. A subsequent approaching vehicle in the same lane that already has a velocity of 10 m/s was released at the next time step without a preceding stop action. This results in a same lane collision of the two vehicles since the default car follow model has been disabled. The release action by the model is appropriate at this time step for both the vehicles since no vehicles are present in the other lanes, however, the registered collision results in a punishment or a negative reward for

the model, since a collision has been monitored by SUMO. Thus, the model trains itself improperly and such scenarios should be avoided.

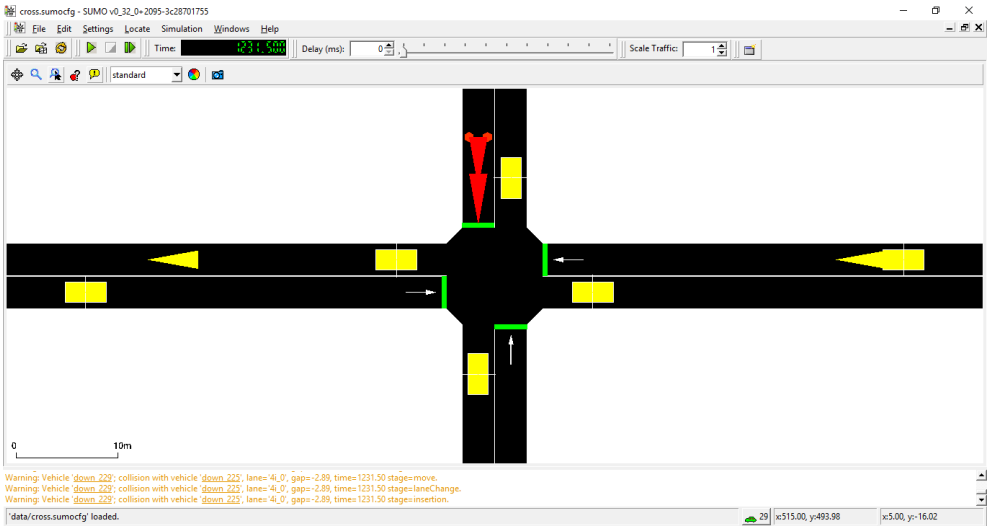


FIGURE 4.3: Rear end collisions in SUMO due to violation of the lane follow model

The solution to this lies in setting the velocity of the vehicle instantaneously to 10 m/s along with the release action using the 'set speed' command unlike previously where it has been increased gradually. This violates the basic vehicle dynamics model for a vehicle and can never be effected in the actual environment since instantaneous acceleration of a vehicle to a desired velocity is impossible. However, it is the necessary evil that helps our model train properly.

The future scope here lies in modifications within the SUMO framework to register collisions occurring at junctions inspite of violating its default car follow model. This functionality cannot be currently imparted to the SUMO simulator using its extensibility ability, since it violates the default models the simulator has been designed to operate on.

## 4.2 Discrete nature of the SUMO simulator

Inspite of disabling the checks as described above to register collisions occurring at the junctions, not all collisions can be registered by the simulator. The simulator is inherently a discrete simulator and basically functions as a series of time steps. Hence the vehicles do not actually continuously travel along the defined path but are placed at calculated locations along the path at each time step. These locations are calculated based on the velocity of the vehicle and the time step duration. The time step can be explicitly set using the TraCI API. At each time step, if the vehicles physically intersect each other or are within the pre – defined minimum gap between each other, a collision is registered on the SUMO console. Hence, seldom is the case that the vehicles are actually on intersecting paths; however, due to the discrete nature of the simulator, they are placed at non – intersecting locations along the path between two consecutive time steps. Thus, inspite of an unsafe release action by the model, they do not actually collide at the explicit time step. This phenomenon has been demonstrated below through an example.

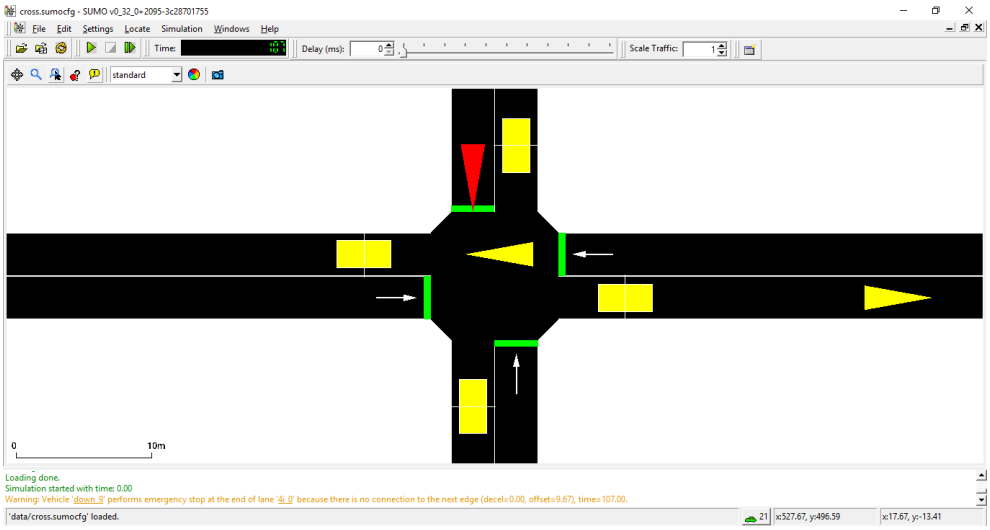


FIGURE 4.4: Vehicles released unsafe by the reinforcement learning algorithm that would definitely collide in reality

The figure 4.4 indicates two vehicles in the right lane and the top lane. The right lane vehicle has been released since it was detected at an earlier time step. However, the network also releases the vehicle from the top lane and no brakes lights can be seen. Due to the discrete nature of the SUMO simulator, the vehicles directly are placed at their new positions (right lane vehicle over the detector) at the next time step and they do not intersect at this time step. Hence the SUMO console does not notify of a possible collision. However, they might have definitely collided at a time between the duration of two time steps.

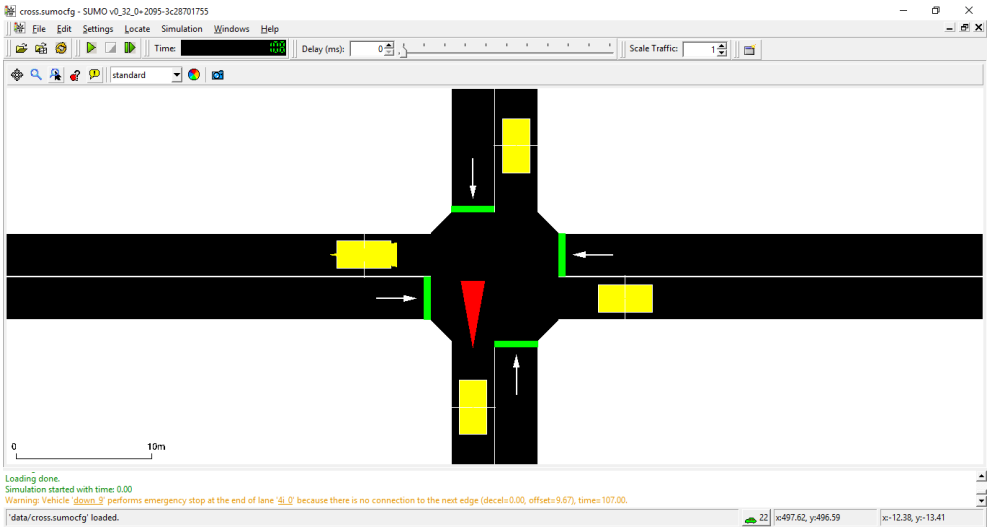


FIGURE 4.5: Collision unnoticed because of the discrete nature of the simulator

**Difficulties posed by the limitation in the context of the project:**

In reality, the above scenario would be an unsafe release and the algorithm should

have been penalized for such an action as in reality the vehicles would definitely encounter a collision. This introduces a finite amount of inaccuracy within our trained model; however, has no particular solution.

**Proposed modification or solution to confront the limitation:**

The only solution that can avoid such scenarios is by setting the time step extremely small such as 0.2 seconds, with which, every occurring collision can be monitored by the SUMO simulator. Also the minimum gap factor between the vehicles can be increased and thus the probability of such scenarios can be minimized.

**Implications of the proposed modification or solution to confront the limitation:**

The time step of the simulation in context of all our experiments has been set to 1 second. The primary reason for this being that SUMO has several current issues which limit its stability in context of collisions. This point has been explained in detail in the subsequent part. Also, SUMO's car following models are calibrated for a time step of 1 second and their behavior for smaller time steps produces unnatural behaviors [DLRnda][Wu+17]. Also, a smaller time step increases the computational effort of the simulator and reduces the overall training speed of the model. Neural Networks are already computationally expensive and this solution may only make it worse, thus affecting the overall training speeds. Also, with such a small time step and with frameworks such as the graphics card version (GPU version) of tensorflow that has been used in the context of our project, the simulation runs very slowly and seldom even crashes the entire system due to lack of computational resources. One of the viable solutions to this issue has been to increase the minimum gap factor to monitor a collision between the vehicles. If it has a relatively high value, the above described scenarios take place with very low probability and majority of the collisions can be monitored. The time step can also be maintained to a high value in order to avoid its effect on the speed of the simulation.

## 4.3 Unstable behavior of the SUMO simulator in case of collisions

As explained in the above sections, the inherent nature of the SUMO simulator is of a collision – free simulator. Even though after tweaking a few settings within the simulator and monitoring and simulating collisions within the simulator, it results in instability of the entire functioning of the simulator. This instability results in random behavior or undefined and unnatural behaviors [Wu+17]. Such behaviors also result in errors during the simulation which are not reproducible. The simulation may even crash in case of fatal errors. During the scope of our project as well, several such instances of undefined behavior have been experienced during the simulation runs. During the debugging of the resulting errors, it has been found that the errors have a complete undefined and random behavior and no particular cause behind them. For e.g. errors such as 'the defined path for the vehicle does not have the current downstream route' inspite of the vehicle movement along the correctly defined trajectory. Also, these errors are not reproducible even though all the generators have been seeded to exactly replicate the same simulation run. This affects the overall training process and demonstration of our concept of system of systems

level intersection management. However, the only solution here lies in imparting stability to the simulator and thereby its underlying models to monitor and handle collisions in the future releases.

# Chapter 5

# Future Scope of the Project and Conclusion

## 5.1 Future Scope of the Project

Several further extensions of the project can be carried out preserving the basic concept and the established interface between the SUMO simulator and reinforcement learning for intersection control of vehicles. The main idea behind these extensions would be further enhancements in context of the efficiencies achieved in case of traffic flows or increase in overall safety of the entire system.

### 5.1.1 Further optimization by expanding the Action space

As seen in the experiments performed in the scope of this thesis, the action space comprises of only the decisions regarding release of vehicles with respect to the lanes. For e.g. in case of the simplistic 4 lane experiment, the action space comprised of only 4 explicit actions – Release of vehicle from Lane – 1, Lane – 2, Lane – 3 or Lane – 4. However, further optimization can be ensured by extending the action space and thus the corresponding output decisions. For e.g. the reinforcement learning algorithm can also be trained to output actions such as to increase or decrease the velocity of the approaching vehicles so that the waiting time near the junction can be minimized. It is a well – known fact that the $CO_2$ and other emissions for vehicles with internal combustion engines are higher at idling revolutions. Hence this may prove to be beneficial in the future when the waiting times of the vehicles can be optimized by prior increase or decrease of approaching vehicle velocities rather than halting the vehicles completely before the junction. The extension of the action space could not be undertaken due to time constraints and this could be an extension of this thesis topic in the near future.

### 5.1.2 Further expansion of the State space

In order to expand the action space as defined above, the state space may also have to be better characterized for the velocities of the approaching vehicles. Since the velocities of the approaching vehicles also consist of continuous values, it may have increased the complexity of the overall number of different states several manifolds due to the curse of dimensionality that increases the dimension space exponentially. However, this would be a proposed extension of the thesis in future to ensure even higher levels of optimizations and alleviation of the problem pertaining to traffic flows. The project does not aim at reducing the waiting times explicitly with the current characterization of the state space and the reward function since this project was more of a feasibility study. However, this would be definitely a viable future

prospect of the work. It involves accounting for the current waiting duration of the vehicle in the state space representation. However, again with continuous values for the waiting times, the state space would have evolved exponentially. And even though if not explicitly, the project work tries to minimize the waiting times implicitly and can be witnessed from the results achieved. The system of systems level control in context of intersection control eventually tries to reduce the waiting times and allows the vehicles to maneuver the intersection with and / or without waiting before the junction through the eradication of the conventional traffic light systems.

Another extension of the project could be in the context of incentive or economics based reward function for the reinforcement learning model to minimize the waiting times explicitly for a certain class of vehicles or upon special request communicated by the vehicles. The underlying aim of such a system would be to release vehicles belonging to a certain class such as emergency vehicles, ambulances or police vehicles immediately without halting at the junction. The reinforcement learning model can be trained for such release actions through a suitable high reward characterization for the approaching vehicles and accounting for such special class of vehicles in the state space characterization. The concept can also be applied for normal vehicles who explicitly request for such quick releases in order to save time for e.g. in case of vehicles which need to rush to the airport. These requests can be served by the reinforcement learning model by accounting for these requests in the state space representation and subsequent tuning of the reward functions during the training phase. Such requests can be served for economic incentives as well in case of commercial applications.

The fuel consumption of commercial vehicles is quite high at idling revolutions in comparison with those of the ordinary passenger vehicles. The greenhouse gases thereby emitted by commercial vehicles are also quite high in contrast with passenger vehicles. Hence, priority release actions for the large vehicles such as trucks, buses, construction mobile equipments, etc. can result in overall reduction in the fossil fuel requirements and emission of greenhouse gases thereby contributing to the sustainability of the society. This could be one of the viable extensions of the project. The model can be trained against different classes of vehicles modeled in SUMO and represented suitably in their state space characterization. The subsequent tuning of the corresponding reward function can help in effective training of the reinforcement learning model to undertake such priority release actions. Several complicated intersection scenarios could have also been modeled and cooperative intersection control strategy could have been demonstrated on such intersections. This also could be an application oriented extension of this thesis topic in the near future since a friction free interface between reinforcement learning and the SUMO simulator via the TraCI API has been already established in the context of this project.

### 5.1.3 Safety Supervisor for safety critical applications of AI based systems

This project of intersection control aims at developing a concept of system of systems level intersection management based on reinforcement learning. Even though the technology is a promising one from the efficiency gains it provides as illustrated in the section 3.2.4.6 in contrast with the conventional traffic management solutions, traffic management is a safety critical application. The 26 collisions encountered for the triple lane four – way model cannot be ignored in reality and pose a serious threat

to human life. The advancements about the potentials of AI to transform transport have definitely brought excitement, however along with the concerns about safety, privacy, security, fairness, economic and military implications of such autonomous systems [Amo+16].

A future work in this context is the development of a safety supervisor (SSV) that governs and monitors the AI based intersection management system. Such a conceptual safety supervisor has been presented in the publication 'A conceptual Safety Supervisor Definition and Evaluation Framework for Autonomous Systems' by researchers at the Fraunhofer Institute for Experimental Software Engineering in Kaiserslautern, Germany [FSA17].

The concept presented in this work can be deployed in the context of our project on intersection control. The SSV would evaluate the safety of the situation based on the information from the Safety World Model and initiate a countermeasure from the set of Risk Reduction Strategies if one is needed [FSA17]. In our context, this Safety World Model would consist of the current Situation Description i.e. the current state space and information about the vehicles issued release requests by the AI system, the Situation Prediction and the Situation Risk Assessment. The Situation Prediction would consist of mathematical models of vehicles along with the model of the intersection and the environment dynamics that can predict how the current strategy by the agent for the given state space can evolve in the future. If intersecting trajectories, collisions or very close maneuvers are found out to occur in the future because of an improper decision of the AI system based on the prediction, the Situation Risk Assessment informs about the risks involved and the criticality of the situation through suitable metrics to the SSV [FSA17]. The safety supervisor takes over control and manages the intersection maneuver for the vehicles from the approaches defined in the Risk Reduction Strategy which may consist of conventional / traditional traffic light based approaches or a safety channel approach where it can direct the vehicles that may potentially collide in the future. The supervisor is thus implemented with traditional algorithms that can be verified. Every decision or release action issued by the AI based system is checked for safety by the supervisor system thus providing an additional runtime verification that focuses on the safety as the absence of unreasonable risks [FSA17]. The functioning of the AI system can be again resumed after the vehicles involving the ambiguous decisions by the AI system have maneuvered the intersection successfully under the guidance of the SSV.

With an AI based system to control the intersections, it may not be possible to ensure a collisions free approach. Since the nature of traffic distribution for every intersection scenario cannot be exactly represented, it may not be possible to train the agent using reinforcement learning against every scenario and for every traffic distribution. Thus, against an unseen setting, the agent would try to approximate the decision with a finite amount of ambiguity involved. In order to circumvent this ambiguity, a safety supervisor or a similar system is a must, to monitor the decisions outputted by the AI based system. With such a tandem operation of the systems, the number of collisions may be reduced to zero and safety may be guaranteed.

Another extension to this concept might be an individual vehicle level safety supervisor. These supervisors for the vehicles at an intersection would communicate with each other using V2V communication and review the release actions of the AI

based system if their trajectories would intersect in the future or might be too close with respect to each other. In case of ambiguity, they can request the infrastructure based safety supervisor to issue them safe and reliable release commands or the vehicle level supervisors can adapt the decisions themselves by taking suitable actions such as reducing or increasing the speed of one of the vehicle and avoiding a potential collision. The vehicle level and the infrastructure level SSV can also operate in tandem by double cross – check of the decisions of the AI system to guarantee even increased safety levels.

One more concept of dual agents can be proposed to assure safety during the intersection maneuver. Both the agents are installed at the infrastructure after undergoing through similar training procedures using reinforcement learning. One of the agent or network is used to actually control the intersection and the other is further trained during the functioning of the entire system and made to learn from real experiences. During this ongoing training process for one of the agent, the SSV can supervise this learning process and guarantee that no unsafe behavior has been learned [FSA17]. After the elapse of certain training period, the learned characteristics by the agent that is being further trained can be transferred to the agent that is actually involved in the intersection control process for e.g. in case of neural networks this implies transfer of updated weights and biases.

## 5.2 Conclusion

Management of traffic at intersections is one of the challenging problems for ensuring an efficient transport system. With increasing penetration levels of vehicular communications and automation, technologies such as cooperative intersection management are promising ones for the efficient management of traffic. This thesis lays down a concept of a reinforcement learning based system of systems level cooperative intersection management. It briefly introduces the advancements in the field of advanced driver assistance systems and their advancements towards realizing complete autonomy along with a brief overview to the field of Artificial Intelligence.

The main focus is on the development of a cooperative intersection management system trained using reinforcement learning techniques and facilitated by vehicle – to – infrastructure communication, mainly proposed for vehicles with level 4 and above automation levels. The concept has been demonstrated using the SUMO simulator as an environment to train the intersection controller through the establishment of a suitable interface between the simulator and reinforcement learning via the TraCI API. The fundamental concept has been tested through a highly simplistic single lane four – way model in conjunction with the one – step off – policy temporal difference learning i.e. Q – learning algorithm. The ineffectiveness and several limitations of the simplistic model drove the further enhancements within the state space characterization in order to attempt to achieve a better performing intersection management system. Inspite of the non – convergence of the action value updates during the learning process, the agent is capable of reducing the overall number of collisions over a period of time (reducing trend) independent of the network structure and the hyper parameters (varied between certain predefined limits) and thus demonstrates a learning ability and yields positive results to our feasibility study of development of an agent for intersection management using reinforcement learning.

The neural network based intersection management system in contrast with the conventional traffic lights system for the triple lane four – way model shows great future potential on the lines of efficiency improvement and improved environment sustainability. Limiting the state space in terms of sections i.e. dividing the path between the incoming and outgoing detector along the lane in sections of pre – determined lengths would drastically limit the diversity among different states. For e.g. a vehicle detected between a distance 455 to 460 would be characterized in the state space with a value of 457.5 and this would limit the total number of states. Multiple visits would be possible to a lower number of states during the training process and thus its convergence can also be ensured with the off – policy Q – learning. Such a state space characterization would result in a few limitations pertaining to efficiency as illustrated in the split state characterization approach from section 3.2.2. Thus the convergence of the action – values to arrive upon an optimal policy for the position based state space characterization using the already built robust interface between the SUMO simulator and reinforcement learning via the TraCI API can be another line of research in the near future. The usage of the policy gradient algorithm to impart convergence during the learning process is also a viable option since it directly maps states to actions rather than analyzing the value function. With the enhanced action space to account for the velocities of the vehicles i.e. not only issue the subsequent release requests from each lane, but also individually control the velocities of the vehicles for improved efficiency, algorithms such as DDPG can also be used to handle such continuous action spaces.The development of the proposed concept

of the safety supervisor to fortify the safety of such artificial intelligence based systems in context of safety critical applications such as intersection management can promote research in the field of machine learning that matters.

Several experiments have been carried out to train an agent using reinforcement learning to perform intersection control. The experimental results show the feasibility of such a concept along with its subsequent limitations and future extensions. This project also demonstrates that the SUMO simulator can be used as a training environment for reinforcement learning to train an agent to manage the vehicular traffic at an intersection. During this project, a few insights have been investigated and elaborated in the context of modifications and limitations of the SUMO simulator in context of our project of reinforcement learning based intersection control. An interface between the SUMO simulator and reinforcement learning can be established via the TraCI API to train an agent to manage intersections based on collisions and intersection maneuvers.

Along with seeking to maximize machine learning algorithm performance on isolated data sets, we must also develop solutions using the machine learning perspective that impact many people since these gains matter to the world outside of machine learning research [Wag12]. The concept of system of systems level intersection management introduced in this project can attempt to create such an impact through its wide spread adoption.

The concept of a system of systems level intersection management based on reinforcement learning introduced by us, to the best of our knowledge, has not been demonstrated before in the reinforcement learning literature as well as in context of applications of the SUMO simulator i.e. to train an agent to control intersections through the collisions occurring during the training tenure.

# Bibliography


[AFS16] Rasmus Adler, Patrik Feth, and Daniel Schneider. "Safety engineering for autonomous vehicles". In: *Dependable Systems and Networks Workshop, 2016 46th Annual IEEE/IFIP International Conference on*. IEEE. 2016, pp. 200–205.

[Amo+16] Dario Amodei et al. "Concrete problems in AI safety". In: *arXiv preprint arXiv:1606.06565* (2016).

[AO03] Masami Aga and Akio Okada. "Analysis of vehicle stability control (VSC)'s effectiveness from accident data". In: *Proceedings: International Technical Conference on the Enhanced Safety of Vehicles*. Vol. 2003. National Highway Traffic Safety Administration. 2003, 7–p.

[Ben+14] Klaus Bengler et al. "Three decades of driver assistance systems: Review and future perspectives". In: *IEEE Intelligent Transportation Systems Magazine* 6.4 (2014), pp. 6–22.

[Boj+16] Mariusz Bojarski et al. "End to end learning for self-driving cars". In: *arXiv preprint arXiv:1604.07316* (2016).

[Bro+13] Jeremy Broughton et al. "Traffic safety basic facts 2012: Junctions". In: (2013).

[BSR15] J Bonnefon, A Shariff, and I Rahwan. *Autonomous vehicles need experimental ethics: Are we ready for utilitarian cars? Computers and Society*. 2015.

[Cas17] Noe Casas. "Deep Deterministic Policy Gradient for Urban Traffic Light Control". In: *arXiv preprint arXiv:1703.09035* (2017).

[CE16] Lei Chen and Cristofer Englund. "Cooperative intersection management: a survey". In: *IEEE Transactions on Intelligent Transportation Systems* 17.2 (2016), pp. 570–586.

[Cle18] Andrew Lewis Cleland. "Bounding Box Improvement With Reinforcement Learning". PhD thesis. Portland State University, 2018.

[CP11] MC Jobin Christ and RMS Parvathi. "Segmentation of medical image using clustering and watershed algorithms". In: *American Journal of Applied Sciences* 8.12 (2011), p. 1349.

[DAR] Statistical Tools for Data Analysis and Visualization in R. *Partitioning cluster analysis: Quick start guide - Unsupervised Machine Learning*. Accessed: 2018-10-21. URL: `http://www.sthda.com/english/wiki/print.php?id=236`.

[DS08] Kurt Dresner and Peter Stone. "A multiagent approach to autonomous intersection management". In: *Journal of artificial intelligence research* 31 (2008), pp. 591–656.

[Erd14] Jakob Erdmann. "Lane-changing model in SUMO". In: *Proceedings of the SUMO2014 modeling mobility with open data* 24 (2014), pp. 77–88.

[Ert18] Wolfgang Ertel. *Introduction to artificial intelligence*. Springer, 2018.

[Fat+13] Negin Fathollahnejad et al. “On reliability analysis of leader election protocols for virtual traffic lights”. In: *Dependable Systems and Networks Workshop (DSN-W), 2013 43rd Annual IEEE/IFIP Conference on*. IEEE. 2013, pp. 1–12.

[Fer+10] Michel Ferreira et al. “Self-organized traffic control”. In: *Proceedings of the seventh ACM international workshop on VehiculAr InterNETworking*. ACM. 2010, pp. 85–90.

[Fet+18] Patrik Feth et al. “Multi-aspect Safety Engineering for Highly Automated Driving”. In: *International Conference on Computer Safety, Reliability, and Security*. Springer. 2018, pp. 59–72.

[FS12] Friedrich Fraundorfer and Davide Scaramuzza. “Visual odometry: Part ii: Matching, robustness, optimization, and applications”. In: *IEEE Robotics & Automation Magazine* 19.2 (2012), pp. 78–90.

[FSA17] Patrik Feth, Daniel Schneider, and Rasmus Adler. “A Conceptual Safety Supervisor Definition and Evaluation Framework for Autonomous Systems”. In: *International Conference on Computer Safety, Reliability, and Security*. Springer. 2017, pp. 135–148.

[Ger18] Pascal Gerber. “Zuverlässiges Verhalten autonomer Fahrzeuge durch Reinforcement Learning”. Bachelor Thesis. Technische Universität Kaiserslautern, 2018.

[Hec+17] Martin Heckmann et al. “Development of a personalised intersection assistant”. In: *ATZ worldwide* 119.5 (2017), pp. 36–41.

[HM+07] Verena Heidrich-Meisner et al. “Reinforcement learning in a nutshell.” In: *ESANN*. 2007, pp. 277–288.

[HNB17] Martin Hofmann, Florian Neukart, and Thomas Bäck. “Artificial Intelligence and Data Science in the Automotive Industry”. In: *arXiv preprint arXiv:1709.01989* (2017).

[Ise+17] David Isele et al. “Navigating Occluded Intersections with Autonomous Vehicles using Deep Reinforcement Learning”. In: *arXiv preprint arXiv:17 05.01196* (2017).

[Jan+17] Joel Janai et al. “Computer vision for autonomous vehicles: Problems, datasets and state-of-the-art”. In: *arXiv preprint arXiv:1704.05519* (2017).

[Jar+18] Maximilian Jaritz et al. “End-to-end race driving with deep reinforcement learning”. In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2018, pp. 2070–2075.

[Kie18] Frank Kienle. *Introduction to Data Science: Machine Learning Part 2*. University Lecture. 2018.

[Kna+09] Andreas Knapp et al. “Code of Practice for the Design and Evaluation of ADAS”. In: *Preventive and Active Safety Applications, eSafety for road and air transport, European Commission Project, Brüssel* (2009).

[KPM17] SA KPMG. *KPMG’s Global Automotive Executive Survey 2017*. KPMG, 2017.

[Kra+12] Daniel Krajzewicz et al. “Recent Development and Applications of SUMO - Simulation of Urban MObility”. In: *International Journal On Advances in Systems and Measurements* 5.3&4 (2012), pp. 128–138.

[Lil+15] Timothy P Lillicrap et al. "Continuous control with deep reinforcement learning". In: *arXiv preprint arXiv:1509.02971* (2015).

[LJY17] Fei-Fei Li, Justin Johnson, and Serena Yeung. "Cs231n: Training Neural Networks, Part I". In: *University Lecture 6* (2017).

[LLW16] Li Li, Yisheng Lv, and Fei-Yue Wang. "Traffic signal timing via deep reinforcement learning". In: *IEEE/CAA Journal of Automatica Sinica* 3.3 (2016), pp. 247–254.

[Luc+16] Andre Luckow et al. "Deep learning in the automotive industry: Applications and tools". In: *Big data (Big Data), 2016 IEEE international conference on*. IEEE. 2016, pp. 3759–3768.

[Maf16] Stefano Mafrica. "Bio-Inspired Visual Sensors for Robotic and Automotive Applications". PhD thesis. Aix-Marseille Universite, 2016.

[Min09] Onivola Henintsoa Minoarivelo. "Application of Markov Decision Processes to the Control of a Traffic Intersection". In: *postgraduate diploma, University of Barcelona* (2009).

[Mni+13] Volodymyr Mnih et al. "Playing atari with deep reinforcement learning". In: *arXiv preprint arXiv:1312.5602* (2013).

[Mni+15] Volodymyr Mnih et al. "Human-level control through deep reinforcement learning". In: *Nature* 518.7540 (2015), p. 529.

[Moh+13] Detlev Mohr et al. "The road to 2020 and beyond: What's driving the global automotive industry". In: *McKinsey&-Company (Pub.), Automotive & Assembly–Latest thinking, available online at http://www. mckinsey. com/˜/media/McKinsey/dotcom/client_service/Automotive% 20and% 0 Assembly/PDFs/McK_The_road_to_2020_and_beyond. ashx, accessed* 28.3 (2013), p. 2014.

[MW10] Paula Marchesini and Wilhelmina Adriana Maria Weijermars. *The relationship between road safety and congestion on motorways*. SWOV Institute for Road Safety Research Leidschendam, 2010.

[Rlu] *Reinforcement Leanring with Q - tables*. 2018. URL: `http://www.apastyle.org/learn/faqs/web-page-no-author.aspx`.

[SB13] Olivier Sigaud and Olivier Buffet. *Markov decision processes in artificial intelligence*. John Wiley & Sons, 2013.

[SB17] Richard S Sutton and Andrew G Barto. *Introduction to reinforcement learning*. MIT press Cambridge, 2017.

[Sil+14] David Silver et al. "Deterministic policy gradient algorithms". In: *ICML*. 2014.

[Sil15] D Silver. *Reinforcement learning course by David Silver*. University Lecture. 2015.

[Sta14] SAE Standard. "J3016."" In: *Taxonomy and Definitions for Terms Related to On-Road Motor Vehicle Automated Driving Systems* 4 (2014), pp. 593–598.

[SWM12] Falko Saust, Jorn Marten Wille, and Markus Maurer. "Energy-optimized driving with an autonomous vehicle in urban environments". In: *Vehicular Technology Conference (VTC Spring), 2012 IEEE 75th*. IEEE. 2012, pp. 1–5.

[TA96] Thomas L Thorpe and Charles W Anderson. *Tra c light control using sarsa with three state representations*. Tech. rep. Citeseer, 1996.

[Tru13] Volvo Trucks. "EUROPEAN ACCIDENT RESEARCH AND SAFETY REPORT 2MI3". In: (2013).

[VHGS16] Hado Van Hasselt, Arthur Guez, and David Silver. "Deep Reinforcement Learning with Double Q-Learning." In: *AAAI*. Vol. 2. Phoenix, AZ. 2016, p. 5.

[Wag12] Kiri Wagstaff. "Machine learning that matters". In: *arXiv preprint arXiv:1206.4656* (2012).

[Wan+15] Ziyu Wang et al. "Dueling network architectures for deep reinforcement learning". In: *arXiv preprint arXiv:1511.06581* (2015).

[WCLF18] Pin Wang, Ching-Yao Chan, and Arnaud de La Fortelle. "A Reinforcement Learning Based Approach for Automated Lane Change Maneuvers". In: *arXiv preprint arXiv:1804.07871* (2018).

[WML16] Zia Wadud, Don MacKenzie, and Paul Leiby. "Help or hindrance? The travel, energy and carbon impacts of highly automated vehicles". In: *Transportation Research Part A: Policy and Practice* 86 (2016), pp. 1–18.

[Wu+17] Cathy Wu et al. "Flow: Architecture and benchmarking for reinforcement learning in traffic control". In: *arXiv preprint arXiv:1710.05465* (2017).

[Bosnd] Bosch. *Community-based parking: helping one another find the nearest available parking space more quickly*. n.d. URL: https://www.bosch-mobility-solutions.com/en/products-and-services/mobility-services/connected-parking/community-based-parking/.

[Connd] Continental. *Turn Assist - oncoming traffic*. n.d. URL: https://www.continental-automotive.com/de-DE/Passenger-Cars/Chassis-Safety/Software-Functions/Active-Safety/Turn-Assist/Turn-Assist-Oncoming-Traffic.

[DLRnda] DLR Institute for Transport Systems. *Simulation/Basic Definition*. n.d. URL: http://sumo.dlr.de/wiki/Simulation/Basic_Definition#Defining_the_Time_Step_Length.

[DLRndb] DLR Institute for Transport Systems. *SUMO - Simulation of Urban Mobility*. n.d. URL: https://www.dlr.de/ts/en/desktopdefault.aspx/tabid-9883/16931_read-41000/.

[Daind] Daimler. *Sideguard Assist. Greater safety for pedestrians and cyclists*. n.d. URL: https://www.daimler.com/products/trucks/mercedes-benz/emergency-brake-assist-system.html.

[Eur18] European Commission. *Intelligent transport systems. Vehicle safety systems*. 2018. URL: https://ec.europa.eu/transport/themes/its/road/application_areas/vehicle_safety_systems_en.

[Eurnd] European Commission. *Horizon 2020*. n.d. URL: https://ec.europa.eu/programmes/horizon2020/.

[Fis16] Fiszels, R. *Reinforcement Learning and DQN, learning to play from pixels*. 2016. URL: https://rubenfiszel.github.io/posts/rl4j/2016-08-24-Reinforcement-Learning-and-DQN.html.

[Mednd] Medium. *Genetic Algorithm for reinforcement learning*. n.d. URL: https://becominghuman.ai/genetic-algorithm-for-reinforcement-learning-a38a5612c4dc.

[Mernd] Mercedes Benz. *Benz-Patent is Part of the World Documentary Heritage*. n.d. URL: https://www.mercedes-benz.com/en/mercedes-benz/classic/history/benz-patent-motor-car.

[NHTnd] NHTSA. *Fatality analysis reporting system (FARS)*. n.d. URL: https://www.nhtsa.gov/research-data/fatality-analysis-reporting-system-fars.

[Opend] OpenAI. *CartPole –v0*. n.d. URL: https://gym.openai.com/envs/CartPole-v0/.

[Wik18a] Wikipedia contributors. *Intelligence — Wikipedia, The Free Encyclopedia*. [Online; accessed 5-September-2018]. 2018. URL: https://en.wikipedia.org/w/index.php?title=Intelligence&oldid=858136315.

[Wik18b] Wikipedia contributors. *SAE International — Wikipedia, The Free Encyclopedia*. [Online; accessed 11-July-2018]. 2018. URL: https://en.wikipedia.org/w/index.php?title=SAE_International&oldid=846220874.

[Wik18c] Wikipedia contributors. *Wireless ad hoc network — Wikipedia, The Free Encyclopedia*. [Online; accessed 15-September-2018]. 2018. URL: https://en.wikipedia.org/w/index.php?title=Wireless_ad_hoc_network&oldid=858589362.

[dSPnd] dSPACE. *Developing Advanced Driver Assistance Systems (ADAS) and Functions for Autonomous Driving*. n.d. URL: https://www.dspace.com/en/pub/home/medien/brochures/adas.cfm.