DEEP Q LEARNING FOR COOPERATIVE AUTOMATED HIGHWAY DRIVING

DEEP Q LEARNING WITH A MULTI-LEVEL VEHICLE PERCEPTION FOR COOPERATIVE AUTOMATED HIGHWAY DRIVING

RICHARD HAMILTON, B.Sc, B.Eng

A thesis presented for the degree of

Masters of Applied Science M.A.Sc

Department of Electrical and Computer Engineering

McMaster University

Canada

McMaster University MASTER OF APPLIED SCIENCE (2021)

Hamilton, Ontario (Engineering)

TITLE:	Deep Q Learning with a Multi-Level Vehicle Perception for Cooperative Automated Highway Driving
AUTHOR:	Richard Hamilton, BSc, BEng (McMaster University)
SUPERVISOR:	Dr. Ali Emadi Canada Excellence Research Chair Faculty of Engineering Department of Electrical & Computer Engineering
NUMBER OF PAGES:	xv, 123

Abstract

Autonomous vehicles, commonly known as “self-driving cars”, are increasingly becoming of interest for researchers due to their potential to mitigate traffic accidents and congestion. Using reinforcement learning, previous research has demonstrated that a DQN agent can be trained to effectively navigate a simulated two-lane environment via cooperative driving, in which a model of V2X technology allows an AV to receive information from surrounding vehicles (termed Primary Perceived Vehicles) to make driving decisions. Results have demonstrated that the DQN agent can learn to navigate longitudinally and laterally, but with a prohibitively high collision rate of 1.5% - 4.8% and an average speed of 13.4 m/s. In this research, the impact of including information from traffic vehicles that are outside of those that immediately surround the AV (termed Secondary Perceived Vehicles) as inputs to a DQN agent is investigated. Results indicate that while including velocity and distance information from SPVs does not improve the collision rate and average speed of the driving algorithm, it does yield a lower standard deviation of speed during episodes, indicating lower acceleration. This effect, however, is lost when the agent is tested under constant traffic flow scenarios (as opposed to fluctuating driving conditions). Taken together, it is concluded that while the SPV inclusion does not have an impact on collision rate and average speed, its ability to achieve the same performance with lower acceleration can significantly improve fuel economy and drive quality. These findings give a better understanding of how additional vehicle information during cooperative driving affects automated driving.

Acknowledgments

I would like to express my deepest appreciation to a few individuals who have provided me with the immense support and guidance I needed as I pursued my Masters.

First and foremost, I would like to thank my supervisor, Dr. Ali Emadi, who has not only been a source of inspiration, but has provided me with a great deal of assistance, support and guidance in the development of my research project, methodology and approach. Your expertise was invaluable in the development of this project and I thank you for placing the amount of autonomy and trust in me to pursue this research. Your methodical thinking and approach has been an inspiration in my thesis (and outside of it) and I am grateful for this extremely valuable experience to be trained under your leadership. I would also like to thank my mentor, Dr. Brian Baetz, who encouraged me to pursue my Masters and has been an ongoing source of support and inspiration to me over the last four years. I would also like to thank Aashik Chandramohan for his guidance and willingness to share his insights from the previous research he conducted, which this current research is an extension of.

In addition, I would also like to thank all of my family and friends, who provided me with ongoing moral support as I worked full-time and pursued a Masters simultaneously. I could not have completed this project without their undying support, confidence, and love. Thank you for cheering me on, and helping me achieve my dreams.

I dedicate this achievement to my late grandfather, Leroy Grant, who taught me many invaluable lessons in life, including that my education is of utmost importance, and that it was okay to start over - his words are the reason that I have the privilege of submitting my Master's thesis.

Deep Q Learning with a Multi-level Vehicle Perception for Cooperative Automated Highway Driving

Richard Hamilton

Contents

List of Figures

List of Tables

List of Abbreviations

ACC	Adaptive Cruise Control
ADAS	Advance Driver Assistance System
AEB	Automated Emergency Braking
AI	Artificial Intelligence
AV	Autonomous Vehicles
C-V2X	Cellular Vehicle-to-Everything
CACC	Cooperative Adaptive Cruise Control
CAV	Connected and Autonomous Vehicle
CCC	Conventional Cruise Control
CI	Confidence Interval
CNN	Convolutional Neural Network
D2D	Device-to-Device
D2N	Device-to-Network
DDQN	Double Deep Q Network
DL	Deep Learning
DNN	Deep Neural Network
DPM	Deformable Part-based Model
DQL	Deep Q Learning
DQN	Deep Q Network
DRL	Deep Reinforcement Learning

DSRC	Dedicated Short Range Communication
ECS	Electronic Control System
FOV	Field of View
GPS	Global Positioning System
HD	High Definition
HOG	Histogram of Oriented Gradients
IMU	Inertial Measurement Unit
IS	Importance Sampling
LiDAR	Light Detection and Ranging
LRR	Long Range Radar
ML	Machine Learning
MMS	Mobile Mapping Systems
NEAT	NeuroEvolution of Augmenting Topologies
OEM	Original Equipment Manufacturer
RCNN	Regions of Convolutional Neural Network
RL	Reinforcement Learning
RNN	Recurrent Neural Network
PER	Priority Experience Replay
PPV	Primary Perceived Vehicle
SAE	Society of Automotive Engineers
SIL	Software in the Loop
SPV	Secondary Perceived Vehicle

SSD	Single Shot Multibox Detector
SUMO	Simulation of Urban Mobility
TOF	Time of Flight
TraCI	Traffic Control Interface
TTC	Time-to-Collision
UAV	Unmanned Aerial Vehicle
UER	Uniform Experience Replay
V2I	Vehicle-to-Infrastructure
V2V	Vehicle-to-Vehicle
V2X	Vehicle-to-Everything
VPGNet	Vanishing Point Guided Network
YOLO	You Only Look Once

1 Introduction

Autonomous vehicles (AVs), commonly known as self-driving cars, are increasingly becoming a reality [1]. AVs are vehicles that are able to sense their environment and navigate without human intervention. Given that AVs have the potential to radically change dynamics of transportation systems, they are of interest to automotive makers, electronics makers, the IT industry, among many other stakeholders.

In 2019, there were approximately 36,000 fatal traffic accidents in the US [2]. According to the US Department of Transportation, 96% of fatal crashes are caused by human error [3]. Another issue related to the current transportation system is congestion, where commuters especially in urban areas spend hours in traffic congestions each year. According to the INRIX traffic scorecards that analyze congestion in cities, a look at 1064 cities in 2016 revealed that drivers spend an average of 9% of their entire commute in congestion, with larger cities having greater congestion rates. Congestion has far reaching consequences; traffic congestion leads to environmental and sustainability issues, where 3.1 billion gallons of energy is wasted due to traffic congestion, globally [4].

AVs are of deep interest in the automotive field due to their potential to reduce traffic accidents and congestion [5]. Similar to a human driver, an AV requires the ability to sense its environment in order for it to perceive situations and make decisions. The most common method to sense the environment is through onboard sensors that are able to detect and localize traffic objects in relation to the AV. While useful, onboard sensors have their own set of limitations such as the susceptibility to external conditions and their ability to only perceive information in a limited field of view (FOV) [5]. One alternative is the use of vehicle-to-everything (V2X) technology, through which

AVs are able to send and receive relevant information with other vehicles and traffic infrastructures through wireless connections [6]. Vehicles and other traffic elements communicating with each other in an environment is known as cooperative driving. The shared information may consist of a vehicle's position and velocity which can be used by the AV's control algorithm.

An AV's control algorithm is very complex as it needs to be comprehensive enough to know how to handle almost all traffic scenarios, which is exhaustive [3]. This requirement for complete coverage of traffic scenarios makes rule-based approaches for developing control algorithms very limited. Rule-based approaches to vehicle interaction are rather inflexible; they require a great effort from engineers to develop and validate and also usually generalize poorly to new scenarios. An alternative to rule-based approaches is machine learning (ML), which is a very common research topic in the space of AV control development today.

ML is the approach for a system to automatically learn and improve from experience without being explicitly programmed [7]. There are three main types of ML: 1) Supervised Learning, 2) Unsupervised Learning, and 3) Reinforcement Learning (RL). In supervised learning, an algorithm learns what to do by using training data with external experiences and an answer key, whereas in unsupervised learning, training is done using external experiences in the absence of an answer key. Finally, in RL, an algorithm uses its own experiences to learn what to do. These three machine learning types are discussed in greater detail in further chapters of this thesis.

In order for supervised learning to work effectively, a large amount of labeled data is needed, which requires a considerable amount of human effort [3]. Applications of supervised learning demonstrate overall acceptable performance, however, it has not performed better than humans, as the data used to train is usually generated by humans and therefore suffers personal bias [3]. Hence, creating a control model using supervised learning is extremely difficult. Given that RL does not involve collecting large amounts of labeled data and therefore is unsusceptible to human bias, it has emerged as a prospective and interesting approach for developing control models for AVs.

Although RL driving algorithms have not yet been put in practice in the real world, research such as Chandramohan et al. (2019) has shown that RL is a prospective approach to autonomous driving. In this research, Chandramohan et al. (2019) trained a Deep Q Network (DQN) agent to navigate a multilane environment using cooperative driving. Building on these findings, the current thesis investigates whether including information from more vehicles in the environment improves the performance of a DQN agent similar to [8].

1.1 Background

Prior to research by Mnih et al. (2015), RL algorithms such as Q-learning had achieved some success, but it was limited to domains with low dimensional state space that could be fully observed [9]. Q-learning involves a table containing values ("Q-values") for each possible action for each state. An agent uses this table to determine which action to take by selecting the action with the maximum Q-value given its current state [48]. Most ML problems that involve interaction with the environment are complex and have high dimensional state space, and would be difficult to be captured fully by a table. Mnih et al. (2015) from DeepMind, Google's artificial intelligence research group, created a new RL that would address the the shortcomings of previous RL paradigms. In research conducted by this group a novel RL approach was created in which Q-learning was combined with a deep neural network (DNN), which is used to approximate Q-values instead of a table. A DNN used in this context is known as a deep Q-network (DQN) [10].

A DQN enables the agent to handle environments with high-dimensional state space which was previously not achieved [10]. Use of a DQN agent allowed researchers to make significant advances, in which several layers of nodes were able to efficiently represent high dimensional inputs. This development formed a new framework for RL called Deep Q-Learning (DQL), where the standard framework of making decisions by trial and error typical of RL now also incorporated deep learning, allowing an agent to use unstructured input data without manual engineering of the state space [10]. In research by Mnih et al. (2015), a DQN agent was developed and tested on challenging classic Atari 2600 games using pixel and game score as inputs. The DQN agent outperformed all previous algorithms and achieved a level comparable to that of a professional human games tester across a set of 49 games. Remarkably, the DQN agent proved its ability to learn and represent its environment to predict an optimal decision at each instance while solving a sequential problem [7]. This groundbreaking research paved the pathway for DQL to become one of the most prominent research domains in autonomous driving [1].

Using a DQN agent is a prospective approach to autonomous driving as it does not require data collection and is also able to handle complex environments. However, one major challenge with using DQL for AVs is the training process, which cannot be done on real-life systems as training is not only costly but would incur a high number of collisions and unpredictable situations [1]. Hence, simulating an AV in a virtual environment is an appropriate solution to this problem. Examples of autonomous driving tasks where RL can be applied include controller optimization, path planning, motion planning, decision-making, scenario-based policy for intersections, merges, and finally risk estimation [1]. This research focuses on the decision-making aspect of autonomous driving.

Decision-making algorithms manage the driving behaviours of AVs including acceleration, braking, lane-changing, lane-keep, and so on [11]. Decision-making is often generated by manual rules based on human driving experiences. In recent years, many attempts have been made to use DQL to deal with long sequential decision-making problems in autonomous driving. Most research that uses DQL in a decision-making algorithm focuses on agents with single objectives, such as longitudinal control (speeding up or slowing down) or latitudinal control (changing lane left or changing lane right). For example, in research by Ishikawa & Aria (2016), a driving algorithm was developed to apply brakes whenever required, by using the distance of the vehicle in front of the AV and the AV's own velocity as inputs. The RL agent was able to learn when to brake or not to brake to avoid collisions and reduce phantom traffic jams by 10%. In Li et al. (2015), a DQN agent was trained to change lanes and overtake slower moving vehicles, using the distance and velocity of the vehicle immediately around the AV and the AV's own velocity and lane as inputs. Using a reward scheme that discourages collisions and encourages overtaking, the researchers were able to develop a DQN agent that learned feasible overtaking policies in different traffic environments and the performance was comparable or at times even better than manually designed decision rules.

Recently, an emerging area of research has been the training of DQL agents to make multi-objective decisions, which includes both lateral and longitudinal control. For example, Duan et al. (2020) used hierarchical RL to learn the decision-making policy for autonomous driving in a highway environment. These researchers developed an end-to-end algorithm that is able to to change lanes, accelerate and decelerate to maneuver through traffic. This was achieved by decomposing the problem into three maneuvers namely driving in lane, right lane change and left lange change, and then learning low level motion control in both the lateral and longitudinal directions for each maneuver. In their model, a master policy learns to choose the maneuver policy that is to be executed at the current state. Compared to non-hierarchical methods, the driving time spent per simulation was reduced by approximately 25%.

Another example of research that uses DQL agents is by Chandramohan et al. (2019), in which the authors combined research from [12] and [13] to train a DQN agent to perform lane change, acceleration and deceleration to maneuver through traffic in a highway environment. Results demonstrated that a DQN agent can learn to navigate longitudinally and laterally, but had a prohibitively high collision rate (for real-life standards) of 1.5% - 4.8% and an average speed of 13.4 m/s. The goal of the current research is to improve the performance of the DQN agent used by Chandramohan et al. (2019) and expand upon its findings.

1.2 Motivation

Recent research has shown that using DQL to solve autonomous driving decision-making in the real world has potential, however, there is a lot of research still to be conducted and improvements to be made in this area. Currently, research is being done to improve the design of the different components of DQL to bring this application closer to real world implementation. Components of DQL that require research to drive its improvement include but are not limited to: DNN architecture, reward strategy and traffic simulation [1]. One aspect of DNN architecture is the selection of state representation that represents the environment. In order to train an effective DQN agent the correct input features have to be used to optimize the agent's decision-making. A part of selecting the correct input features is not only limited to attributes of traffic elements such as speed and distance but also determining which traffic elements should be used.

The motivation of this research is to investigate the impact of including information from traffic vehicles that are outside of those that immediately surround the AV as inputs to a DQL agent. For traffic vehicles to be included as input to DQL agents, they must be perceived by the system either through onboard sensors or wireless communication such as V2X technology. Figure 1 illustrates two levels of vehicles that an AV can perceive, specifically Primary Perceived Vehicle(s) (PPVs) and Secondary Perceived Vehicle(s) (SPVs). PPVs are traffic vehicles that immediately surround the host vehicle (the AV) and can be sensed by the AV since they are within the range and FOV of the AV's sensing technology. SPVs are vehicles that are two levels away from the AV – vehicles immediately surrounding the PPVs that can also be sensed by the AV because they are within the range and FOV of the AV's sensing capabilities.

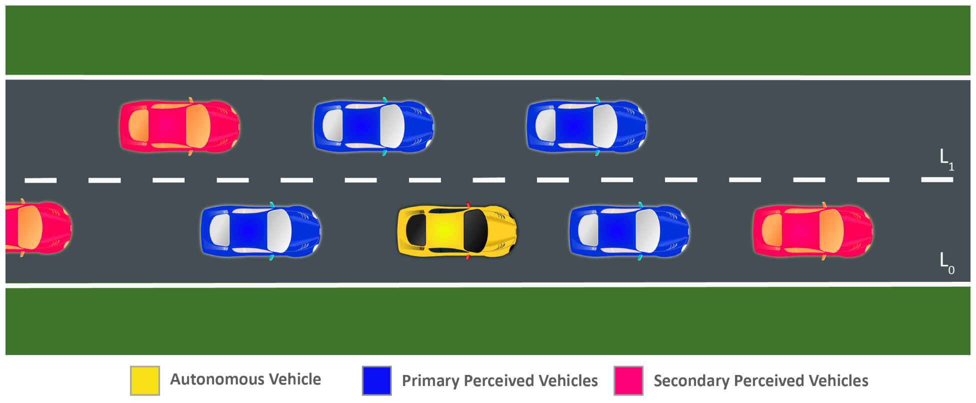


Figure 1: Primary perceived vehicles (blue) and secondary perceived vehicles (red) in relation to the AV (yellow).

Most research only includes PPVs as input when developing driving algorithms for two main reasons: 1) PPVs have the most impact on the performance of a driving algorithm because they have an immediate impact on collision rates and the average speed an AV travels, and 2) Given that most vehicles use sensors that can only perceive objects immediately in their line of sight and PPVs block SPVs from being detected by these sensors, SPVs can be more difficult to perceive by the AV. The ability of V2X technology to offer wireless communication provides an opportunity for SPVs to be perceived, opening up the opportunity for their information to be used. While there is no debate that PPVs have the largest impact on a model's performance, there has been little work to investigate the extent to which including SPV information impacts this performance. As such, the goal and motivation of this thesis is to bring light to the impact of including information from SPVs as inputs to a DQL agent on the performance of an AV.

1.3 Contributions

The current Master's thesis expands our understanding of the use of RL in the context of cooperative autonomous driving. In this research, we implement a DQL agent that reproduces results from Chandramohan et al. (2019), in which a learning algorithm trains a DQN driving algorithm using RL to navigate a multi-lane highway using cooperative driving. Replicating these results verifies and lends confidence to our understanding that RL is a plausible approach for autonomous cooperative driving, paving way for further novel research in this area. This research makes modifications and successful improvements to the design of the DQN training algorithm such that it is less sensitive to random seeds and hyper-parameter choices, in comparison to the model created by Chandramohan et al. (2019). As such, this research successfully produces an improved model that reduces overoptimistic biases, which is a significant improvement for future experiments. Additionally, an alternative sampling technique is designed and implemented to increase the probability of sampling important experiences during training of the DQL agent, yielding a model that has increased efficiency during training. Together, with the replication of the existing model and the subsequent improvements implemented, this research presents a new and improved DQN learning algorithm for cooperative autonomous driving.

Furthermore, this thesis offers an innovative and methodical approach to the integration of

SPV information as input to the DQN agent. Analysis performed in this research will help provide support for whether the use of SPV information is beneficial when designing DQN driving algorithms for cooperative highway driving.

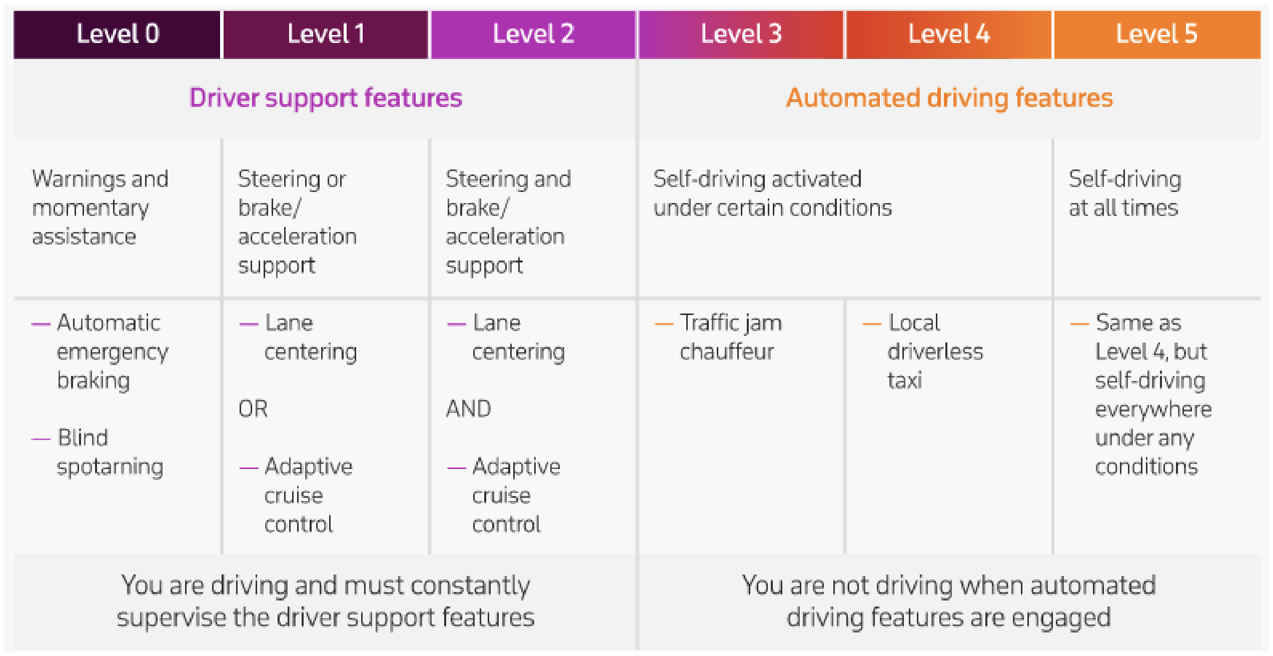
1.4 Thesis Outline

While PPVs theoretically provide enough information for a DQN agent to successfully navigate a highway environment with cooperative driving, including information from SPVs will improve the performance, namely collision rate and average speed, of the DQN agent, as the SPVs will provide longer range information that can give the AV critical seconds of reaction time to avoid collisions or extreme decelerations.

This thesis is organized into 7 distinct Chapters, each covering a fundamental area that will collectively help address the thesis statement outlined above. Chapters 1 to 4 provide a detailed literature review covering various important topics. Chapter 1 provides the introduction and motivation for this research, following which Chapters 2 and 3 provide a detailed background on the fundamentals of AVs, advanced driver assistance system features and how AVs are modeled and simulated. Chapter 4 discusses the field of ML with a focus on a specific type of ML called RL in great detail. Following this, Chapter 5 provides the detailed experimental set up used in the experiments performed in this research. In Chapter 6, results from experiments are presented and discussed in detail. Finally, Chapter 7 provides conclusions, limitations and key takeaways of this research.

2 Fundamentals of Autonomous Vehicles

An AV is a vehicle that is able to intelligently navigate itself along a trajectory without human intervention, i.e. a vehicle that is capable of performing a steering or speed adjustment task by itself [14]. The Society of Automotive Engineers (SAE) has defined six levels of driving automation known as the SAE J3016, which classifies AVs based on their capabilities and the level of human intervention that is required [15]. The SAE J3016 has become the de facto global standard for stakeholders in the AV technology space. Figure 2 outlines the six levels of driving automation from level 0 (no automation) to level 5 (full automation). As shown in Figure 2, one of the specifications for Levels 0 to 2 is that the driver is required to drive or to be engaged at all times even if the driver's feet are off the pedals and he or she is not steering. Level 1 and Level 2 AVs have features that are able to provide lateral (steering) and longitudinal (brake/acceleration) support to the driver. The two levels differentiate based on the capability to perform lateral and longitudinal control simultaneously; AVs that can provide lateral and longitudinal control simultaneously are considered Level 2, while AVs that can only perform one of these functions at a time are Level 1. Level 2 is the maximum level of automation for commercial AVs in the market today [5]. For a Level 3 AV, the driver is not required to drive or be engaged while the vehicle is in control, however should be ready to take control of the vehicle when requested in difficult situations. Level 4 and Level 5 AVs do not require the driver to take over driving in any situation, rendering them fully automated vehicles. AVs are classified as Level 4 when they are fully autonomous (no driver required) but only in controlled areas, whereas Level 5 is reserved only for vehicles that are fully autonomous under all situations everywhere.



Level 0	Level 1	Level 2	Level 3	Level 4	Level 5
Driver support features			Automated driving features		
Warnings and momentary assistance	Steering or brake/ acceleration support	Steering and brake/ acceleration support	Self-driving activated under certain conditions		Self-driving at all times
— Automatic emergency braking — Blind spotarning	— Lane centering OR — Adaptive cruise control	— Lane centering AND — Adaptive cruise control	— Traffic jam chauffeur	— Local driverless taxi	— Same as Level 4, but self-driving everywhere under any conditions
You are driving and must constantly supervise the driver support features			You are not driving when automated driving features are engaged		

Figure 2: Six levels of automation by SAE J3016 [16].

AV systems are considered to possess artificial intelligence (AI) capabilities, which is the ability for a machine to simulate intelligent human tasks [17]. Intelligent human tasks include: 1) Perception: translating external information into an internal representation, 2) Cognition: manipulating and processing internal information to derive new signals, and 3) Execution: converting signals into action. As illustrated in the general architecture of AV systems in Figure 3, AV systems generally perform all three tasks [18].

AV systems should be able to determine where the vehicle is located and what objects are around it, both of which are achieved with the use of powerful sensors during perception [18]. Sensors are used to perceive key information such as – but not limited to – distance, speed and classification of traffic objects from the vehicle's surroundings. Once sensors receive information from the surroundings, dedicated computing modules process the information to derive control signals during cognition (control). Before computing a control signal, some AV systems may also be required to make high level control decisions (decision making) and/or determine a reference trajectory (path planning). High level control decisions may include lateral lane change in both directions, speeding up, speeding down, idling, and others. Finally AV systems use computing modules and mechanical systems to execute control signals during execution [18].

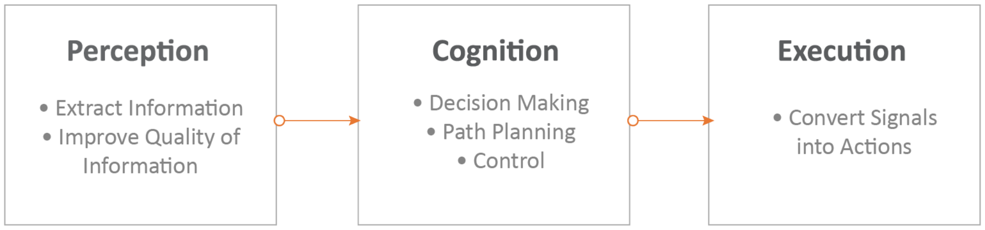


Figure 3: AV subsystem components based on intelligent tasks.

The remainder of this chapter discusses AV subsystem components.

2.1 Autonomous Vehicle Intelligent Subsystems

2.1.1 Perception

The goal of perception in AV systems is to extract required information about the vehicle and its environment in order to navigate successfully [18]. In general, perception is achieved by using sensors to measure or capture information from the environment and subsequently post processing sensor data to a) extract relevant information and b) improve the quality of the information. This ability to perform perception successfully is what affords AVs the capability to make intelligent decisions in real time. Moreover, advanced technologies such as V2X allow high level perception in which vehicles have the capability to communicate with traffic systems in their surroundings, including other vehicles. Given the unique perception capabilities of V2X technologies and the relevance to this research, more information about V2X is provided separately in Section 2.2.

2.1.1.1 Sensing

AVs require the ability to perceive the environment surrounding the vehicle and also measure values within the system; both of these tasks are achieved by its sensors [18]. Sensors must be able to create a perspective and localizational view of the environment quickly and accurately to produce a high quality understanding of what situation the vehicle is in. There are two types of sensors used in AVs: 1) exteroceptive sensors, which are used to perceive and extract information about the environment surrounding the vehicle and 2) proprioceptive sensors, which measure values from within the system such as motor speed, vehicle position, etc. [19].

Exteroceptive Sensors

Four exteroceptive sensors that are commonly used in AVs, namely LiDAR, radar, camera and ultrasonic sensors are discussed in detail below.

1) *LiDARs* (Light Detection and Ranging) are active electromagnetic sensors used for measuring distance and creating 3D representations of an AV's surrounding environment [19]. LiDARs emit laser or light outwards and measure the time it takes for signals to be reflected back to the sensor, a principle known as "time of flight" (TOF).TOF and the speed of the signal – which in the case of LiDARs is the speed of light – are used to calculate the distance of objects detected. LiDARs today can make distance measurements at rates higher than 150 kilohertz (150,000 pulses per second). LiDARS have a range of over 250 m and are therefore classified as long-range sensors. LiDARs

provide major advantages in AV systems due to their precision, accuracy, and high resolution. However, drawbacks of LiDARs include costliness and low availability. Given the advantages that LiDARs provide, a plethora of current research aims to address their drawbacks, including the development of solid-state LiDARs and infrared LiDARs for widespread usage.

2) *Radars* are electromagnetic sensors that emit radio waves to measure distance, angle and velocity of objects [19]. Radars can be used in multiple frequency bands (24 GHZ, 77 GHz, etc.); the higher the frequency, the higher the ability for the sensor to distinguish between multiple objects. Classes of radars include short, medium and long range and individual classes cover a subset of ranges between 50m to 150m and over. One of the greatest advantages of using radars is their robustness towards all kinds of environmental conditions such as rain, fog, night conditions, etc. Additionally, in contrast to LiDARs, radars are significantly less expensive and in general more readily available.

3) *Cameras* are a type of passive exteroceptive sensor which means that unlike the aforementioned sensors, cameras do not emit their own signals. Instead, cameras use ambient light to produce a digital 2D representation of a 3D environment [20]. One significant advantage of using cameras is their ability to perceive colours and textures, which is extremely beneficial when identifying objects in the environment such as road signs, traffic lights, lane markings, etc. [19]. While images from cameras can be used to compute distance to objects, this requires complex algorithms and most times also a stereo vision set up, in which cameras with two or more separate lenses each produce separate images. One disadvantage to cameras as sensors in AV systems is that their performance is adversely affected by low intensity light and adverse weather conditions, as a result making them only reliable in appropriate conditions.

4) *Ultrasonic sensors* are sonar based sensors that emit sound waves to measure the distance of

objects [19]. Similar to other active sensors, ultrasonics use the TOF principle to compute distances to detected objects. Major advantages to using ultrasonic sensors include their cost-effectiveness as well as their robustness towards adverse and changing weather conditions. While these are excellent benefits, ultrasonic sensors come with their own set of drawbacks. While measuring distances to objects in close proximity yields best results using ultrasonics, their accuracy depletes with increasing distances given that disturbances in sound waves increase with distance. Ultrasonic sensors are typically used in ranges from 15 cm to 11 m [21].

Majority of today's automotive original equipment manufacturers (OEMs) typically use LiDARs, radars and cameras for exteroceptive sensors in AVs [16]. Figure 4 illustrates the common locations for such sensors on an AV and also the location of computing modules.

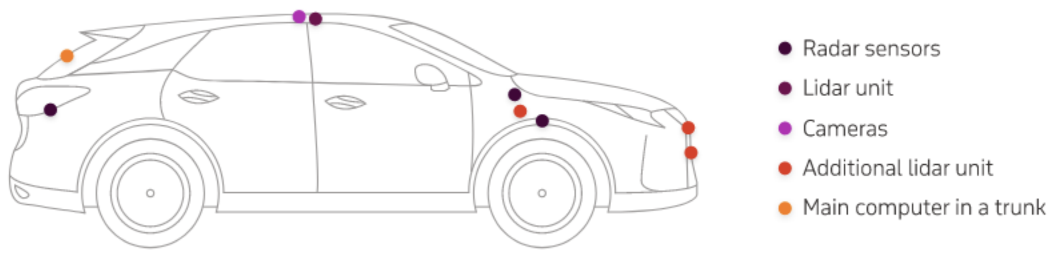


Figure 4: Typical location of exteroceptive sensors on AVs [16].

Proprioceptive Sensors

Three proprioceptive sensors that are commonly used in AVs, namely global positioning system, inertial measurement units, and encoders are discussed in detail below.

1) *Global Positioning System (GPS)* is a navigation system that uses satellite data, receiver input and algorithms to synchronize the location, and velocity and time data for air, sea, and

land travel [19]. GPS technology is commonly found in vehicles of all types today. Just as in non-Avs, AVs use GPS for navigation and localization purposes. While GPS has become a very commonplace navigation tool, one significant disadvantage of using GPS for localization is that there are many factors that can diminish its accuracy. For example, GPS requires an unobstructed line of sight between satellites and receivers, but signals can easily be obstructed due to objects in the environment including high buildings, trees, tunnels, etc.

2) *Inertial Measurement Unit (IMU)* is an electronic device that is composed of accelerometers, gyroscopes and magnetometers to measure an object's acceleration, angular rate and magnetic field, respectively [19]. The aforementioned measurements are calculated for each of the orthogonal X, Y, and Z axes. IMU provides valuable information about the AV's orientation in space. One setback of IMU sensors is that they are only capable of providing information about orientation and dynamics of the vehicle and are unable to provide actual positional information. Another disadvantage of IMUs is that the gyroscope measurement of the current angle is an estimation by integrating the measured angular rotation; this estimation induces errors which can be compounded, known as drift error. A common solution to overcome this drawback is to use GPS to correct for drift errors.

3) *Encoders* are electronic devices used to convert linear or angular position to an analog or digital signal [19]. Encoders are typically used to provide odometer data on the vehicle. Another interesting application of encoders is called dead reckoning, a process in which the encoder is attached to one of the wheels to approximate the relative position travelled, based on measured distance from a known starting point. Dead reckoning is a common solution to estimate a vehicle's position when GPS signal is lost.

2.1.1.2 Sensing Post-Processing

In general, raw data from sensors is not immediately useful and consequently requires post processing to filter and improve accuracy. Below four common post processing algorithms – object detection, lane detection, localization and mapping, and sensor fusion – are discussed in detail.

1) *Object Detection*: Object detection is the computer vision process of detecting and classifying instances of objects of a certain class with an image, such as traffic signs, lane markings, vehicles, etc. [22]. Object detection is an important task in AV systems application given that different classes of objects warrant different actions. Generally speaking, over the course of time and development in this field, object detection has evolved from conventional techniques to deep learning techniques . Conventional techniques, in which algorithms are developed manually, include paradigms such as Histogram of Oriented Gradients (HOG) feature descriptor, and Deformable Part-based Model (DPM) [22]. Although many of today's most advanced methods far outperform object detection via conventional techniques, many current algorithms are still influenced by their contributions including hybrid models and bounding box regression. Deep learning based object detection techniques are algorithms that automatically learn models of object detection using neural networks. Examples of the state-of-the-art methods that use deep learning techniques include the Regions of Convolutional Neural Network (RCNN) features series, Single Shot MultiBox Detector (SSD) series, and You Only Look Once (YOLO) series [22].

2) *Lane Detection*: Lane detection is a special type of object detection that involves detecting lanes on a road using input from a camera [22]. Performing lane detection is a pivotal function of AV systems, because it enables AVs to drive within their lanes, correctly avoid collisions, and also perform subsequent decision making and path planning tasks. Traditional lane detection techniques

rely on cues such as colour-based features combined with Hough transform and Kalman filters to detect lane markings [22]. The performance of these traditional techniques, however, varies with road scene variations consequently making it difficult to achieve reasonable accuracy. Recently, deep learning-based approaches have provided more accurate and faster performance in the lane detection field. For instance, a Vanishing Point Guided Network (VPGNet), an end-to-end deep Convolutional Neural Network (CNN) that can simultaneously detect road markings and lanes lines, introduces vanishing point prediction task into the network which has shown to improve the accuracy of detection even in some bad situations such as rainy days or at night [23].

3) *Localization and Mapping*: Localization is the process of finding the ego vehicle's position relative to a map, and its accuracy affects the feasibility and safety of navigation and path planning [24]. Currently, GPS-IMU based localization is widely used for automated highway driving, however, the accuracy required for urban automated driving cannot be fulfilled by GPS-IMU systems. One practical solution is the use of pre-build high definition (HD) maps, which are maps containing geographic details that are not normally present on traditional maps, such as road and lane information [25]. Data for HD maps are collected from Mobile Mapping Systems (MMS), which are typically vehicles equipped with mapping sensors, navigation sensors, control units and vehicle mounted cameras. The raw data then gets processed to organize input and output data in a sensible way, as well as generate search-able indexes for all data – allowing the system to handle large amounts of data. AVs can access information from HD Maps stored onboard the vehicle to achieve accurate lane level localization.

4) *Sensor Fusion*: The goal of perception is to accurately sense the surrounding environment [22]. Fusing the data from multiple sensor types creates a more reliable system as it significantly increases precision and accuracy. In sensor fusion, a suite of sensors provide specific information about the

same environment scene; this information is then combined from all sources, ultimately yielding higher quality output. As described previously, sensors of different types come with their own set of strengths and weaknesses; leveraging and fusing the strengths of each individual sensor ultimately results in the best performance. For example, cameras provide excellent color and texture information, which is significant for object classification, however cameras are less adept at performing accurate distance estimation with images due to the loss of information from reduced dimension. On the other hand, radars and LiDARs provide little to no information regarding color or texture but are highly proficient at actively providing spatial measurements that yield accurate distance estimation. In order to obtain a more accurate and complete picture of the vehicle's environment, AVs combine information from different sensors in a way that emphasizes and leverages their strengths. Table 2 summarizes common sensor fusions used in AVs.

Table 2: Typical sensor fusions used for AV applications.

Sensor Types	**Objectivive**	**Sensor Information Extraction**
Vision - LiDAR/Radar	Localization, classification and distance and velocity measurement	Camera $\rightarrow$ classification Distance measure $\rightarrow$ LiDAR/Radar Velocity measurement $\rightarrow$ Radar
GPS-IMU	Absolute localization	GPS $\rightarrow$ localization (unobstructive) IMU $\rightarrow$ dead reckoning
GPS-Encoder	Absolute localization	GPS $\rightarrow$ localization (unobstructive) Encoder $\rightarrow$ dead reckoning

2.1.2 Cognition

The goal of cognition in AV systems is to receive information from the surroundings using any of the above mentioned perception techniques, and process this information to derive control signals [18].

2.1.2.1 Decision Making

Advanced AV systems are required to make decisions similar to how human drivers do [26]. Generally, decisions include behaviours such as left lane change, right lane change, speeding up, speeding down or idling. While these decision making capabilities are present in AVs, deriving accurate, efficient and safe decisions in complex environments can still be a significant challenge. Decision making is based on various factors including traffic rules, surrounding environmental conditions, and the dynamic state of the vehicle. Currently, two main paradigms exist for decision making: 1) rule based method and 2) machine learning based method [26]. The rule based method establishes a behaviour rule library according to information such as driving rules, knowledge, experience and traffic rules, and subsequently converts the vehicle states into logic rules corresponding to driving behaviours. The downside of using rule based methods is that at any given point, a vehicle can be in a number of vehicle states, making it difficult to implement a rule library that has coverage for all states. Machine learning (ML) methods provide an alternative to rule based methods through their ability to autonomously generate a driving algorithm by learning what actions to take given different traffic situations. Common ML techniques that perform decision making include deep learning methods, RL and recurrent neural networks (RNN).

2.1.2.2 Path Planning

Path planning is the process of computing a reference trajectory approaching the vehicle’s destination that the vehicle can safely navigate [27]. Path planning is a complex process because not only does the trajectory have to be within driveable regions, it also has to obey the laws of the lane and rules between other vehicles for a collision-free path. Similar to decision making, not all AV systems have path planning capabilities. More specifically, only AV systems that have lateral control – in which the driver is not required to steer when the AV system is in control – tend to have path planning algorithms [26]. Path planning consists of two types of planners, a high level planner and a low level planner, both of which have their own particular functionality. The high level planner selects the optimal path from a digital map like a car navigation system while the low level planner generates a trajectory at a lane and sublane level that the AV finally follows [18].

2.1.2.3 Control

The control component of cognition computes the command request for the mechanical systems of the vehicle to achieve the desired behaviour determined inherently by the AV feature or by the decision-making and/or path planning process [18]. The command request is arbitrated from the delta of the vehicle's current state and its desired state. The control component interfaces the execution components.

2.1.3 Execution

The goal of execution in AV systems is to convert signals derived during cognition into actions [18]. In execution, controllers use lateral and longitudinal control requests from the cognition component or driver to actuate the steering system and propulsion system (throttle and braking systems) of the AV, respectively. The steering system changes the orientation of the front wheel, which results in the change of the heading of the vehicle, ultimately changing its lateral position. On the other hand, commands related to longitudinal control are executed by the propulsion system, which includes the braking system.

2.2 Vehicle-To-Everything

As stated previously, V2X is a technology that allows communication and exchange of messages among vehicles and infrastructure elements [6]. Two important components of V2X technology are called vehicle-to-vehicle communication (V2V), in which a vehicle can communicate with other vehicles in its surrounding, and vehicle-to-infrastructure communication (V2I), in which a vehicle can communicate with external systems such as street lights, buildings, etc. V2V and V2I are highly advanced technologies that present opportunities to make connected and autonomous vehicles (CAVs) a reality, and have therefore gained a lot of interest and popularity in the AV space today. V2V and V2I also have the potential to improve perception processes in the general AV architecture by complementing or replacing traditional sensor systems.

There are 2 main V2X standards, 1) IEEE 802.11 p, and 2) Cellular V2X [6]. The IEEE 802.11 p or Dedicated Short-Range Communications (DSRC) is the original V2X standard and operates similar to Wi-Fi [6]. The standard enables senders that are in close range to directly exchange information with each other illustrated in Figure 5; this mode of communication is referred to as device-to-device (D2D) mode. DSRC works on the 5.9 GHZ band frequency [6]. This standard provides ease of implementation since a communication infrastructure is not required, which is especially beneficial in remote locations. DSRC also has a low latency and high reliability for direct communication, making it suitable for mission critical applications including driving control which requires these attributes [6]. The standard, however, is not without shortcomings; the major drawback for DSRC is its range. The DSRC range exceeds that of most exteroceptive sensors such as radars and cameras, but relative to other V2X standards its communication range of under 1 km is considered short. The DSRC range can be extended via an ad hoc network illustrated in Figure 6.

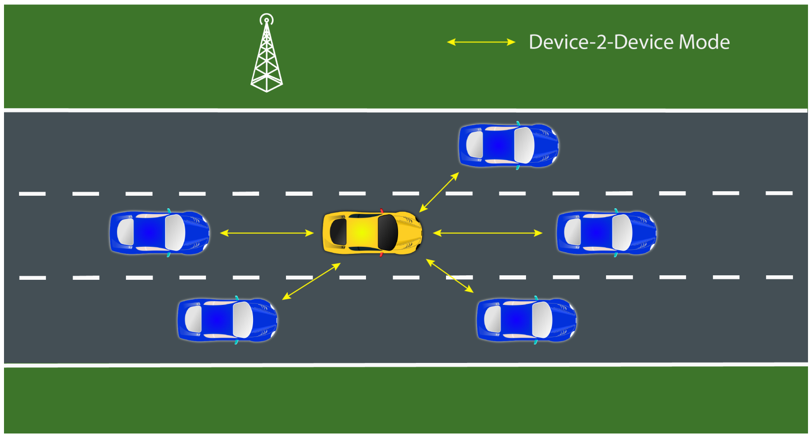


Figure 5: Device-to-device mode in IEEE 802.11 p.

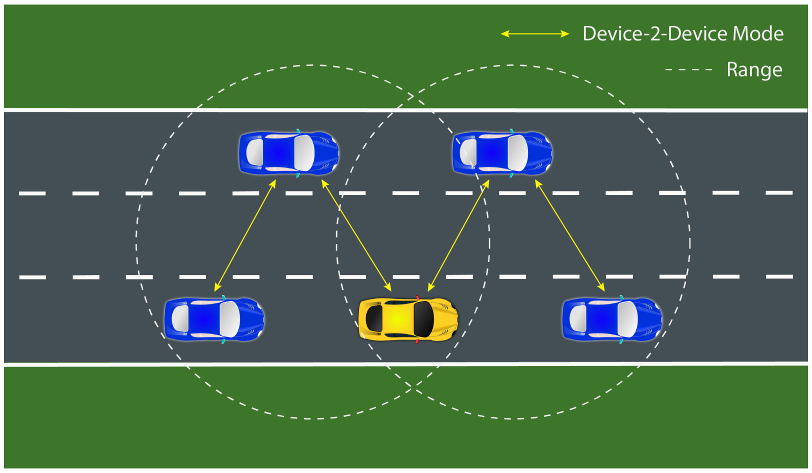


Figure 6: Range extension of IEEE 802.11 p via ad hoc network.

The second standard is the Cellular V2X (C-V2X), which is an emerging alternative to DSRC. C-V2X has two operational modes which cover most applications of V2X [28]. The first mode is D2D which is similar to the DSRC standard where it uses the 5.9GHz band to enable direct communication between entities [29] using a PC5 interface. This mode is designed for mission critical applications. With twice the range of DSRC, C-V2X direct communications can provide critical seconds of reaction time in active safety applications. The second mode is called device-to-network (D2N), which allows communication over cellular networks such as 4G and 5G technology, using a Uu interface illustrated in Figure 7. Two key features of this mode include its long range that extends for miles, and its ability to handle V2N use cases, such as latency-tolerant applications. C-V2X D2N has relatively higher latency compared to its D2D counterparts; however more recently, 5G technology has enabled even more capabilities for C-V2X. 5G technology has allowed improvements such as extreme throughput, low latency and reliable transmission, creating opportunities for fully autonomous and enhanced driving experiences such as augmented reality.

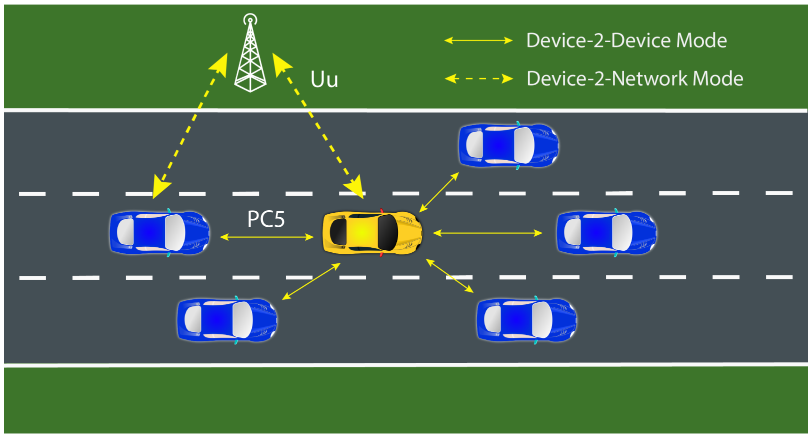


Figure 7: Device-to-Device and Device-to-Network mode in Cellular V2X.

2.3 Advance Driver Assistance Systems

The precursors to fully autonomous driving are Advanced Driver Assistance Systems (ADAS), which involve having the human driver in the loop. ADAS features are used to prevent collisions (which can be passive or active) and/or perform active navigation. Active control involves taking full or partial control of the vehicle's functionality, whereas passive control involves alerting the driver of unsafe situations via audio or visual information, so that the driver can mitigate a collision [30]. Due to the consequences posed by vehicle collisions, fuel economy and congestion, ADAS has become a central focus of research in this field. Advantages of ADAS include enhanced driving comfort, improved safety, improved traffic capacity and reduced fuel consumption. Before the era of high penetration of AVs in the market, driver assistance systems such as ADAS were the precursors that delivered safety benefits to users [31]. The remainder of this section will discuss active ADAS features including automatic emergency braking, adaptive cruise control, cooperative adaptive cruise control, lane keep assist and lane centering control in detail.

2.3.1 Automatic Emergency Braking

Automatic emergency braking (AEB) is an active safety system that activates a vehicle's braking system when a potential collision is detected. Vehicles that are equipped with AEB have two components competing for control over the vehicle: 1) the human driver and 2) the AEB control. Both controllers' objective is to avoid a collision with objects in its path. According to experimental analysis, under normal situations, the time interval between the perception of an object to the initiation of a controlled action to avoid a collision for the average human driver is in the order of seconds [32], which can in some situations be too much latency to avoid an imminent collision. AEB systems can reduce the number of forward traffic collisions.

In non-autonomous passenger vehicles, the human driver has three mechanical control inputs, namely the gas pedal, brake pedal and steering wheel, which control acceleration, deceleration and steering, respectively [32]. The strength of the three driver inputs are measured by an Electronic

Control System (ECS) that forms 3D control variables to determine the current intention of the driver. The ECS also evaluates the practicality of the driver's intention based on physical laws and safety of the vehicle, and makes modifications in the event of violations [32].

In AEB control, the laser scanner and camera, which form a vital part of an AEB architecture, retrieve information and direct it to the sensor fusion unit, to perceive accurate information about the vehicle's environment. The AEB control unit uses accident aversion logic to assess the criticality of the scenario and determine the brake point – vehicle's deceleration – to avoid collision. AEB uses this information in addition to information from the ECS about the driver's intention to determine whether to activate AEB and if so, determine the brake request to meet the computed brake point [32].

2.3.2 Adaptive Cruise Control

Adaptive cruise control (ACC) is an ADAS feature that automatically maintains a steady desired speed while keeping a set time gap to the leading vehicle [33]. While ACC is considered an extension of Conventional Cruise Control (CCC), it actually incorporates two modes: 1) the CCC mode, which allows a vehicle to travel at a desired speed set by the driver, and 2) the headway control mode, which allows the vehicle to maintain a safe distance from a preceding vehicle by adjusting its speed [33]. If a preceding vehicle is not present, ACC will operate in the CCC mode only. If a preceding vehicle is present close to the ego vehicle or travelling slowly ahead, ACC will shift and operate in the headway control mode. Distance and relative speed information of preceding vehicles used by ACC systems is provided by onboard range sensors (such as radar, LiDAR or camera). In the first generation of ACC systems, long range radar (LRR) and LiDAR sensors were commonly used. To improve the safety and usage rate of ACC systems, mid- and short-range radars were included in subsequent generations.

A key part to the design of any ACC system is the type of spacing policy implemented. Three common spacing policies are constant distance, constant time headway, and constant safety factor spacing, which are summarized in Table 3 [34]. Today, constant time headway is the most common spacing policy used by automakers due to their feasibility, traffic flow stability, safety, capability and reliability.

Table 3: Three common spacing policies used in ACC systems.

Spacing Policy	**Description**
Constant distance	Fixed gap between host and lead vehicle, independent from the velocity.
Constant time headway	Desired following distance is proportional to the speed of the vehicle.
Constant safety factor spacing	Following distance is determined by safe stopping distance and proportional to a safety factor.

2.3.3 Cooperative Adaptive Cruise Control

One of the most promising technologies of CAVs, vehicles with V2X technologies that are able to connect to external traffic systems, is cooperative adaptive cruise control (CACC). CACC is an extension of ACC with the added element of cooperative speed control [35]. CACC systems use shared information from other CAVs in a network via V2V technology. With shared information such as acceleration, speed and position, CACC systems have the capability to orchestrate platooning, a cooperative driving application where autonomous and semi-AVs move in a train-like manner by keeping constant distance between vehicles. Additionally, CACC systems can allow vehicles to be driven at harmonized speeds with shorter time headways between CAVs. These features result in vehicles being tightly coupled together, which results in the increase of roadway capacity leading to decreased congestion. The most widely adopted design for CACC application is called the distributed consensus strategy. The consensus strategy contains a network of agents (CAVs) that collectively reach an agreement on a certain state, in this case velocity, – of all agents. "Distributed" in distributed consensus strategy refers to how a network of agents come to a consensus, in that vehicles in a platoon only communicate with its following vehicle to reach a consensus for the whole platoon instead of a centralized scheme in which all vehicles communicate with each other in a network. The distributed consensus strategy has multiple advantages including 1) string stability parameter errors are attenuated along upstream direction and 2) platoon size is not limited by communication range. While proposed CACC control strategies have been proven mathematically and analyzed in theory, only few have been implemented and tested in real world situations [35].

2.3.4 Lane Keeping Assistance

LKA is another ADAS feature that automatically detects lane markings on the road to actively steer the vehicle back into its lane with a slight steering torque in the event of lane drifting [31]. Between 2015-2017, departures from lanes were the cause of 52% of fatal traffic accidents in the US [36]. Therefore, ADAS features that prevent vehicles from drifting outside their lane can significantly help reduce fatal accidents. In LKA systems, perception is achieved by 1) extracting lane lines and road boundary related information such as lane width, lane number, and roadway curvature, and 2) assessing the current state of the ego vehicle's dynamics. Sensor technologies currently used to extract lane and road information includes cameras, radars, sonars and LiDARs, or high definition maps. Lane detection data may be augmented by information from connected data sources such as GPS data, HD Maps and V2X for localization and safety purposes. LKA systems process information received via perception to localize the vehicle relative to its lane and determine torque command required to keep the vehicle from drifting [37]. Next, the steering system arbitrates the torque request with steering input from the driver to orient the vehicle's front wheels to adjust the vehicle's heading.

2.3.5 Automated Lane Centering

LCC is an automated lane centering system that provides automatic lateral control [36]. It is important to distinguish LCC from LKA; while LKA aims to nudge a vehicle back into its lane upon drifting, LCC aims to maintain a central position of the vehicle within a lane. The perception design of LCC is similar to that of LKA, where lane and road information is extracted from an onboard camera and distance measuring sensors. However, unlike LKA, LCC requires a path planning function that computes a reference trajectory in the lane travelled by the vehicle [36]. The reference trajectory is an imaginary desired path which may be in the center of the lane or offset from the center of the lane. For example, the reference trajectory may be closer to the inner lane markings when traveling along a curve. LCC control determines the corrective adjustment the vehicle should make by computing the vehicle's headway error, defined as the offset of the vehicle's heading relative to the reference trajectory. LCC control algorithm uses the headway error to determine the torque request required to make the adjustment to minimize headway error.

Similar to LKA, the torque request gets sent to the steering system to orient the vehicle’s front wheels to adjust the vehicle’s heading accordingly.

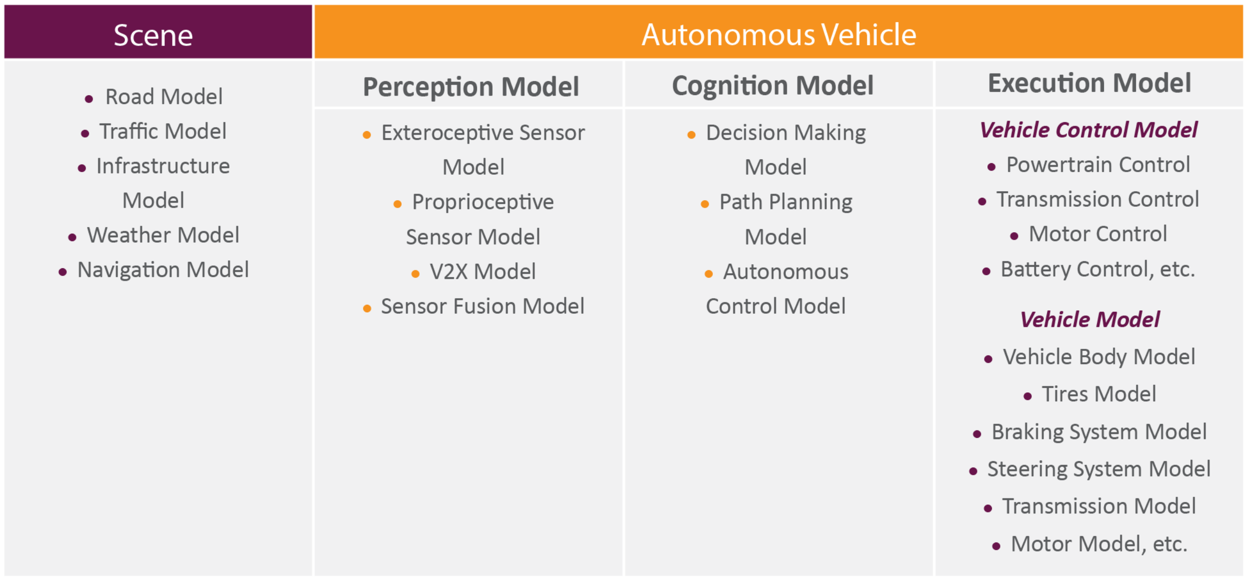
3 Modeling and Simulation of Autonomous Vehicles

The high expenditure, infrastructure and risk associated with the development of real scale models of AVs place limitations on the ability to test AVs on the road [38]. A low cost alternative, as a preceding verification step, would be to use simulated vehicle models to study dynamic responses of such AVs' components. Besides decreased expenditure, simulational verification can have several advantages including the optimization of the product development flow, lower risk of injuries, failure prediction, design and concept gap assessment, and so on. However, simulations have limitations including the fact that they are inherently approximations of the real world scenarios, which might lead to uncertainties in some cases [39]. Nevertheless, if used appropriately and the limitations of the models used in the simulation are known, they can be a very valuable medium. Furthermore, this can be especially useful with newer technological innovations such as in the field of AV technology.

This section provides an overview of the major building blocks associated with the development of an autonomous capable model in a virtual simulation environment, as shown in Figure 8, as well as a review of the relevant literature. The subsystems that are vital in simulations of AVs include the driving environment model, sensor model, and ADAS Control Model, discussed in detail next.

3.1 Driving Scene Model

The driving scene model of AV virtual simulation includes environmental objects and elements such as roads, traffic, infrastructure, weather, navigation constraints, etc. Most simulation software allows users to create and customize road networks' lengths, gradients and intersections [32].



Scene	Autonomous Vehicle		
	Perception Model	Cognition Model	Execution Model
• Road Model • Traffic Model • Infrastructure Model • Weather Model • Navigation Model	• Exteroceptive Sensor Model • Proprioceptive Sensor Model • V2X Model • Sensor Fusion Model	• Decision Making Model • Path Planning Model • Autonomous Control Model	***Vehicle Control Model*** • Powertrain Control • Transmission Control • Motor Control • Battery Control, etc. ***Vehicle Model*** • Vehicle Body Model • Tires Model • Braking System Model • Steering System Model • Transmission Model • Motor Model, etc.

Figure 8: Components of an AV's virtual simulation model [40].

Some software also allows the user to model real road networks by using third-party software such as Google Maps [41]. Full topographical road network models can also be generated using surveying methods, which involve capturing a road network using cameras and sensors. Wellington et al. (2006) describe a method for simulating off-road terrain which may contain variations in local terrain height due to vegetation and obscured obstacles. Another important driving scene element involved in simulations is traffic, a vital component in AV simulations given that AVs will interact with their surrounding traffic. Majority of modeling software allow automatic traffic generation by specifying features such as traffic density and direction. Additionally, they enable the control of individual vehicles in order to create customized traffic scenarios [41]. Traffic can be modeled from microscopic, macroscopic, or micro-macroscopic levels. Microscopic modeling focuses on individual vehicle decisions and their effects on traffic; microscopic modeling is useful for modeling real life drivers [42]. Macroscopic modeling focuses on aggregate behavior, and is useful for simulating traffic flow, such as freeway flow. Weather conditions are also an important environmental factor that should be captured in ADAS simulations. Weather conditions not only have an impact on the AV's sensors, but also the dynamic behaviour of the vehicle such as drag from wind [32]. Praveen et al. (2017) describe a simulation model for fog, rain, and windshield effects.

3.2 Autonomous Vehicle Model

3.2.1 Perception Modeling

AVs must possess the capability to perceive the ego vehicle's environment and its own dynamic state; this holds true both in reality and in simulations [19]. Recall from Section 2.1.1 that exteroceptive sensors are used to perceive information from the environment and proprioceptive sensors are used to measure values within the system. Modeling exteroceptive sensors allows AVs to extract information such as distance (range), speed (range rate), acceleration, classification, etc. of objects around the AV in a simulation. Proprioceptives sensor models allow the AVs to extract information about its own speed, acceleration, position, etc. in a simulation. When creating a sensor model, there are two options available: 1) ground-truth (ideal) sensor modeling, and 2) raw-signal sensor modeling [43]. Ground-truth sensors output measurements with 100% accuracy (ideal). Specifications of FOV, and mounting information (position, orientation, range, etc.) of sensors are typically sufficient for ground truth modeling in most software. Unlike ground-truth sensors, raw-signal sensors output raw imperfect measurements which are distorted by factors such as noise, fog, glow and blur from the environment. Modeling raw-signal sensors requires the modeling of propagation, reflection, and reception of electromagnetic or sound signals. To choose the appropriate modeling sensor, one must consider the objective of the simulation. For control algorithms and model testing (discussed in the next section) ground-truth sensor modeling is generally sufficient. For testing and validating sensor design and signal processing algorithms, a raw-signal sensor model is required [43].

V2X modeling can also be included in simulations of AVs, for example, Ryu et al. (2015) use V2V and V2I systems to autonomously link a swarm of unmanned aerial vehicles (UAVs) for wireless communication. Modeling V2X communication is simple in most modeling software because information such as speed, acceleration and position of objects in the simulation are readily available; typically users are only required to specify the range and frequency of communication. Users can also model communication loss by specifying the probability of packet loss during transmission.

Sensor fusion, discussed in Section 2.1.1.2, is an important requirement for AVs. Multiple sensors such as LiDARs and cameras need to work together to determine objects in the environment for decision making. In Stover et al. (1996), a fuzzy logic architecture is developed for sensor fusion to control sensing resources, data fusion for tracking, object recognition, and automated situation assessment which can be applied to AV sensor fusion. Li et al. (2014) describe a multi-level sensor fusion algorithm, using feature-level fusion for determining a safe, drivable region for an AV and a second conditional lane detection algorithm to determine the driving region of the vehicle without prior knowledge of the environment. Martinez et al. (2017) use neural networks for sensor fusion, determining a sensor's uncertainty of information due to noise. The uncertainty of a sensor is used for decision making algorithms. Collectively, the discussed strategies fulfill the requirements of perception modeling in simulating AVs.

3.2.2 Cognition Modeling

A cognition model or driving algorithm determines the AVs driving behavior in a simulation using information from the vehicle's sensing system (perception) [44]. Most modeling software have built-in algorithms of common ADAS feature controls such as ACC, AEB, LKA, and others discussed previously in Section 2.3 3. In general, AV simulation models are developed to validate proprietary control algorithms or train driving algorithms (machine learning) which are often designed in third-party software such as MATLAB and Python. Therefore, most modeling software allow interfacing with third party software, providing software in the loop (SIL) functionality, used by Lin et al. (2018). SIL allows interactive testing and modification of control algorithms, by directly connecting software to an AV plant model substituting for costlier systems, prototypes or test benches.

3.2.3 Execution Modeling

3.2.3.1 Vehicle Model

The vehicle model is a representation of the physical properties and characteristics of the AV as well as its dynamic behaviors. The extent to which the representation matches real life systems depends on the objective of the experiments or tests that the model is being used for. Modeling software range from simple point models to complex math models that represent the system. Modeling software allow users to model various components of the vehicle including its body, tires, braking system, steering system, transmission and motor, amongst many others. Modeling software range in the extent to which a user can customize the model, based on the requirements of the simulation. For example, MATLAB Simulink allows users to implement their own math models of the listed components, providing users with a vast amount of flexibility, whereas other modeling software provide users with an interface to parametrize ready-made models. For example, in Vehsim a suspension system is modeled using a two-mass spring damper system, and the user can parameterize the model by specifying the spring and damper constants to mimic the vehicle's suspension. Similarly, the other components of the vehicle mentioned previously can be modeled in the same way, depending on the software being used. On the other hand, software such as Simulation of Urban Mobility (SUMO), which is primarily a traffic modeling software, provides limited capability for customizing vehicle modeling, allowing the user to model basic parameters such as maximum speed and vehicle length, but not the dynamic behaviors of mechanical systems.

3.2.3.2 Vehicle Control Model

Vehicle control models map the control signals sent by the cognition model to actuate corresponding system models such as steering, motor, brakes, etc. Often, the vehicle controls do not need to be modeled by the user, as they are typically abstracted away by the vehicle modeling software. If required, users have the flexibility to develop their own vehicle control model by using SIL.

4 Machine Learning with Focus on Reinforcement Learning

4.1 Machine Learning Overview

ML can be broadly defined as the use of computer algorithms that automatically improve through the use of data and experience [7]. ML can be classified into three types, namely: 1) supervised learning, 2) unsupervised learning and 3) RL [7].

Supervised learning uses a training data set that has been labeled with the correct outcome to create an algorithm that can predict the outcome of new data [7]. Hence, the objective of supervised learning is to extrapolate or generalize its responses so that it acts correctly in situations that are distinct but representative of situations in the training set. One example of the application of supervised learning is training a computer to detect vehicles in an image, by using a training dataset of images with all vehicle instances labelled.

In unsupervised learning, an unlabeled training dataset is used to train an algorithm to automatically discover patterns and groupings based on similarities and differences in the data [7]. Unsupervised learning is less predictable than supervised learning especially in complex tasks because of the lack of prior knowledge of a dataset. In unsupervised learning, the algorithm will look for similarities and differences to discover and present the interesting structure in the data in order to make predictions. Due to the unpredictability of unsupervised learning, this technique is not appropriate for use in autonomous driving.

RL is an approach of automating goal-oriented learning and decision-making while solving a sequential problem [7]. In RL, the learner is not provided with instructions, but instead uses trial-and-error to discover which actions produce the maximum reward while interacting in an environment. The goal of the algorithm is to learn to select and perform behaviours associated with the maximum notion of cumulative reward.

The role that ML plays in AV systems will be discussed next, followed by a detail overview of RL.

4.2 Machine Learning in Autonomous Vehicles

ML is increasingly being used in the research and development of AVs. Given the complexity and volume of different driving tasks and scenarios, using ML is an efficient alternative for both performing perception for AVs and designing AV driving algorithms that are comprehensive and capable of handling such driving scenarios.

4.2.1 Machine Learning in Perception

As discussed previously, perception is the task of extracting required information about the environment and the vehicle in order to navigate [18]. ML algorithms have played a crucial role in perception, specifically object detection and classification, which is the process of identifying the category an object belongs to [45]. Supervised ML is the most common approach for detecting and classifying traffic objects from camera images [45]. ML in object detection is analogous to the visual cortex of the human brain, which is designed to sequentially extract edges, followed by patches, surfaces and finally objects and other features [45]. A subset of ML called Deep Learning (DL) attempts to reproduce and duplicate a similar architecture in a computer. In recent years, CNN, a type of DL model, has outperformed different paradigms of ML in object detection and classification.

ML has also played a role in sensor fusion of AV systems, where data from a suite of sensors is combined in a way to emphasize their strengths to get more accurate information. Again, supervised ML algorithms have been the dominant technique for pre- or post-data fusion to reduce errors and improve accuracy and perception.

4.2.2 Machine Learning in Cognition

As discussed previously, cognition in AV systems refers to receiving and processing perceived information to derive control signals. In general, the cognition of most AV systems uses a rule-based approach to ensure absolute security [3]. Rule-based approaches were adopted early on in the AV space, but resulted in limited performance particularly as AV systems became more advanced. Consequently, there has been mounting interest, research and focus on using ML for cognition tasks in AV systems, especially decision-making – this is also the focus of the research in this thesis. Research has been done to use supervised ML to improve the performance of rule-based approaches. For example, NeuroEvolution of Augmenting Topologies (NEAT), which tunes weighting parameters and neural network structures, was used to design optimal neural network topologies to create a race driving algorithm that could overtake and avoid collisions [46]. Kim et al. (2009) enhanced the generalization capabilities of the NEAT evolved model for multi-objectives including steering and speed control [47]. Ye et al. (2018) demonstrated that while these supervised ML approaches are sufficient in certain applications, they are not an efficient technique for creating driving algorithms for the following reasons:

1. Difficult to capture labeled data for new scenarios
2. Large quantity of data required
3. Hard to label decision-making behaviour (driver's psychological factors cannot be quantified)

An agent that is able to learn from its own experiences is capable of overcoming the challenges stated above – this can be achieved through RL. In 2015, Google's DeepMind team used deep RL (DQL) to make decisions in successfully playing classic Atari 2600 games [10], opening up a new channel and paradigm for research in the AV space. Preliminary research has demonstrated that DQL is a formidable approach for decision-making in AVs, however, results were still not satisfactory enough for real life AV applications. The next sections will discuss RL in more details.

4.3 Reinforcement Learning Overview

As mentioned previously, RL is a ML approach that maps situations to actions, so as to maximize a numerical signal [7]. RL is used to solve problems that originate from dynamical systems, which are systems that present output based on past input – sequential problems. In RL, the learner is not instructed on what to do but instead uses trial-and-error to discover which actions produce the maximum reward. RL is distinct from other ML paradigms because it interacts directly with an environment and learns from its own experiences, without requiring supervision [7]. While it may be tempting to classify RL as a type of unsupervised learning since supervised and unsupervised learning seem exhaustive when classifying machine learning paradigms, this is not the case. The difference between RL and unsupervised learning is that while RL aims to maximize a reward signal, unsupervised learning aims to find hidden patterns in a dataset. RL is therefore considered a third and distinct ML paradigm, alongside supervised and unsupervised learning.

As shown in Figure 9, RL can be modeled as a closed loop system. In this model, the agent is the learner and the decision maker, which determines what actions need to be taken [7]. The environment is either a model or the real conditions in which the agent operates. The interaction between the Agent and the Environment occurs at discrete time steps (eg. t = 0, 1, 2, 3,...n). At the current time step (t) the agent selects an action (A_t) from a set of possible actions (A), $A_t \epsilon A$. This action influences the environment and changes its state to a new state (S_{t+1}) and leads to a reward (R_{t+1}), both of which become the new input to the agent in the next time step (S_t and R_t). The goal of the agent is to maximize the reward it gains over time.

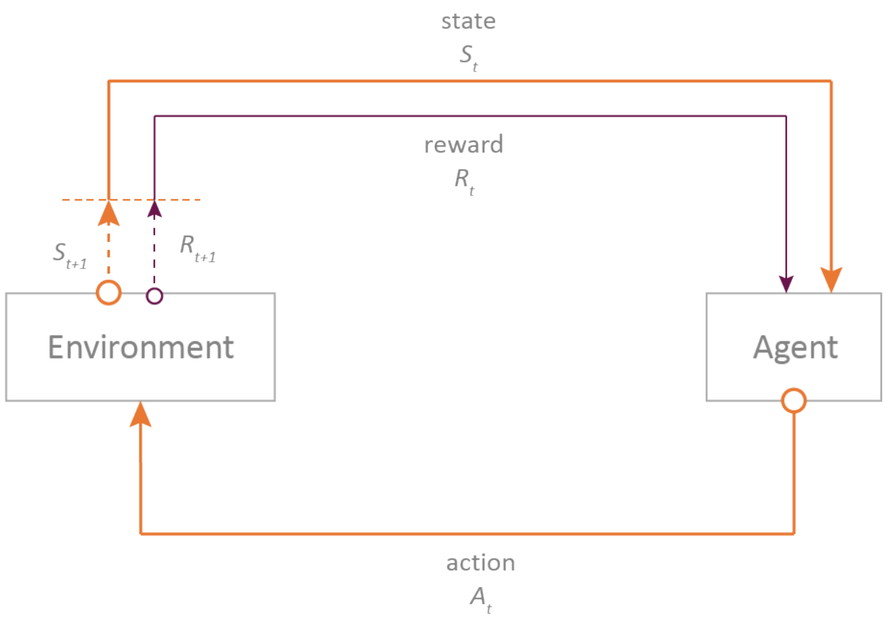


Figure 9: Closed loop system in RL including the environment and agent [40].

4.3.1 Elements of Reinforcement Learning

Along with the learning agent and environment, a RL model also includes a) a policy, b) a reward function, c) reward signal and d) a value function [7]. A policy maps the actions an agent will take given the perceived state; it is dynamic and gets updated throughout training. The policy is the core of a reinforcement agent in that it is sufficient on its own to determine the agent's behaviour. This is analogous to the stimulus-response rules in learning procedures such as classical conditioning in psychology [7]. A reward function evaluates the reward or penalty of every action based on the given situation. A reward function maps relevant situations to a reward signal. It is important to note that the learning objective is implied in the reward function [3]. The reward signal is a number that defines how good or bad a situation is for the agent. Reward signals are the primary forces of altering the policy; for example, if a policy selected an action that resulted in low reward then over time the policy might change to select another action in that situation in the future. A fundamental characteristic of RL is that it does not focus on maximizing immediate reward, but rather the overall reward in the long run. For instance, a policy might choose to take an action that does not provide immediate maximum reward, but the action leads to a situation that will maximize the total reward received in the future. The value function computes an indicator (value signal) that represents what is advantageous in the long term. The value signal of an action-situation pair is the total of the immediate reward (reward signal) and estimated (predicted) reward to accumulate in the future from the action, as illustrated in Figure 10. Reward signal is the quantification of immediate gratification whereas value signal is the quantification of delayed gratification. The way the value function estimates the value signal is an important algorithm design.

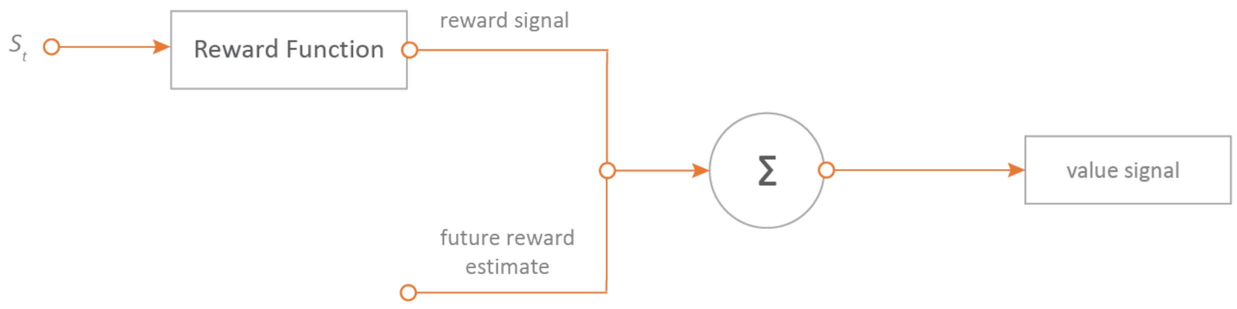


Figure 10: RL value function

4.3.1.1 Exploration vs Exploitation

One unique challenge that is inherent to RL is the trade-off between exploration and exploitation. In order to maximize reward, a learning agent must prefer actions that it has taken in the past and have been effective in producing rewards; this is referred to as exploitation [7]. However, in order to discover such actions, the agent has to also new actions, referred to as exploration. The learning agent must exploit what it has already learned to obtain reward and also explore by trying new actions that may lead to more rewards in the future. This trade-off is a dilemma that has been extensively studied by mathematicians for decades, and still remains unresolved [7]. Currently in research we manage the balancing of exploration and exploitation by using strategies such as greedy policy (see Section 5.1.4).

4.4 Types of Reinforcement Learning

4.4.1 Q-Learning

Q-learning is a model-free form of RL in which the agent does not need to know or have a model of the environment that it will be in; the same algorithm can be used across various environments [48]. Q-Learning works by way of each possible action having a "Q" value per state, which creates a lookup table. The agent chooses an action by using the lookup table to evaluate which action has the highest "Q" value for the state that it is in. The idea of learning is that the Q-values in the table are updated overtime based on the value signal received for each experienced state during training, with the aim that for each possible state the action with the highest Q-value will ultimately maximize the reward overtime [48]. The value function used in Q-learning is the Bellman equation [49], shown in Equation 1.

$$Q^{new}(s_t, a_t) \quad \leftarrow \quad (1 \; - \; \alpha) \; \cdot \; \underbrace{Q(s_t, a_t)}_{old\ value} + \alpha \quad \cdot \; + \overbrace{\Big(r_{t+1} + \underbrace{\gamma \cdot max_a Q(s_{t+1}, a)}_{estimate\ of\ optimal\ future\ value}\Big)}^{learned\ value} \tag{1}$$

4.4.2 Deep Q-Learning

The decision making of DQL agents operate with the same principles as Q-learning agents, where the agent chooses the action with the maximum Q-value based on the given state and the Q-values get updated overtime through the training process. However, unlike Q-learning, DQL uses a DNN – referred to as DQN – instead of a lookup table as a Q-value reference. Q-values are predicted by the DQN illustrated in Figure 11, where inputs (states) get passed into the input layer and are translated by neurons within hidden layers to predict Q-values of actions within the action space at the output layer [50]. Layers are made up of neurons that can be modeled by Equation 2:

$$\gamma = \phi(z) = \phi(w^t x + b) \tag{2}$$

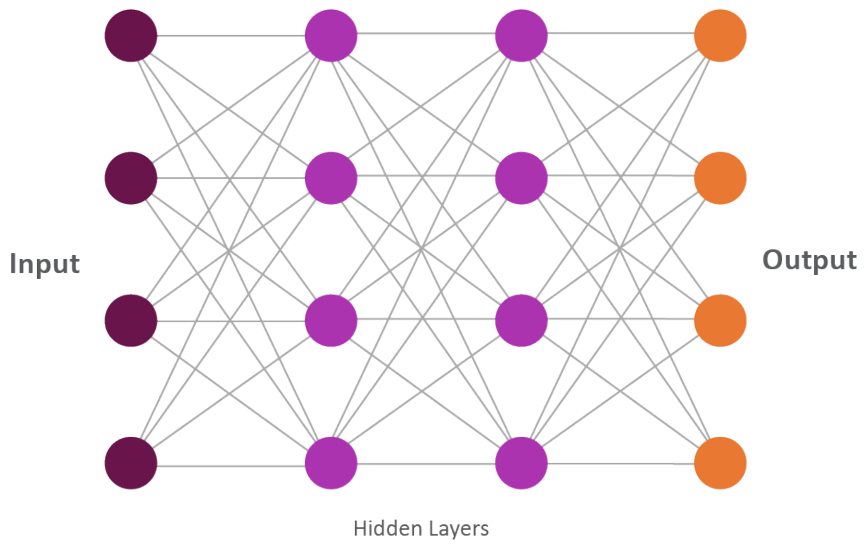
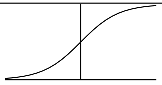
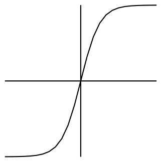
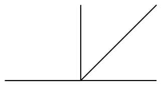
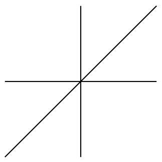


Figure 11: General DQN architecture used in DQL.

The input (x), which is derived from input features or the output of neurons from previous layers, is weighted by (w) and offset by the bias (b) resulting in a scalar value (z) [50]. The output of the neuron is determined by the activation function (ϕ) that uses (z) as the input. A list of activation functions is summarized in Table 4 [50].

Table 4: Summary of activation functions.

Name	Function	Derivative	Figure
Sigmoid	$\sigma(x) = \frac{1}{1+e^{-x}}$	$f'(x) = f(x)(1 - f(x))^2$	
tanh	$\sigma(x) = \frac{e^x - e^{-x}}{e^z + e^{-z}}$	$f'(x) = 1 - f(x)^2$	
ReLU	$f(x) = \begin{cases} 0 & \text{if } x < 0 \\ x & \text{if } x \geq 0. \end{cases}$	$f'(x) = \begin{cases} 0 & \text{if } x < 0 \\ x & \text{if } 1 \geq 0. \end{cases}$	
Linear	$f(x) = x$	$f'(x) = 1$	
Softmax	$f(x) = \frac{e^x}{\sum_i e^x}$	$f'(x) = \frac{e^x}{\sum_i e^x} - \frac{(e^x)^2}{(\sum_i e^x)^2}$	

The output of neurons within the input and hidden layers are feedforward to neurons in subsequent layers [50]. Unlike Q-learning where the Q-values are updated directly in the table, in DQL the DQN updates the weights and biases, since they ultimately predict the Q-values [50]. Weights and biases are updated (during the learning process) in a neural network by a process of backpropagation, which is an algorithm where the error (delta between the estimated Q-value and the predicted Q-value from the output layer) is used to compute the gradient of the loss function with respect to each weight iterating backward from the last layer [50].

Ultimately, with many training scenarios over time the neural network will be able to accurately predict the Q-values of all states, and subsequently the action with the maximum predicted Q-value will be selected by the agent.

5 Experimental Setup of a DQL Agent for Cooperative Highway Driving to Investigate SPVs

This research project aimed to test the effect of including information from SPVs on the performance (collision rates and average speed) of a DQN agent. In the experiments conducted, an AV navigated a two-lane highway environment in which vehicles were able to travel in either the first or second lane, called the driving lane (L0) and overtaking lane (L1), respectively. Given this set up, it is possible for a vehicle to be in front of and/or behind the AV at any given point in both L0 and L1, as shown in Figure 12. Experiments conducted as part of this research aimed to develop a driving algorithm that automatically makes decisions to speed up, speed down, change lane left, change lane right or idle with the goal of the AV moving forward at the fastest speed possible without colliding with other vehicles. Fastest speed possible refers to travelling at a high speed that is within the speed limit and does not violate the rules or safety of the road.

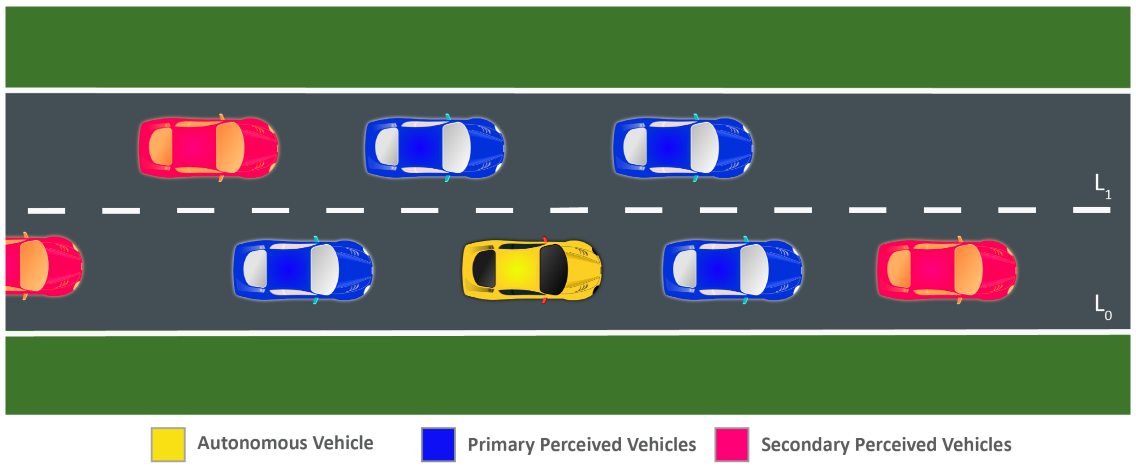


Figure 12: Highway cooperative driving of an AV, PPVs and SPVs.

To investigate the effect of including information from SPVs, three experiments were conducted:

1. Experiment 1: DQL for cooperative automated driving rerun.

2. Experiment 2: DQL for cooperative automated driving revise.

3. Experiment 3: DQL for cooperative automated driving with SPV information.

The goal of Experiment 1 was to replicate the results by Chandramohan et al. (2019), in which the authors used a DQN driving algorithm to maneuver an AV in a multi-lane highway. Replicating these results would verify whether RL is a plausible approach for autonomous cooperative driving and more importantly, helps establish a baseline onto which novel research can be performed. Experiment 2 aimed to improve the setup and design of [8], and create a model that would serve as a control for the subsequent and final experiment. In Experiment 3, SPV information is included as input features to the driving algorithm, and results are compared to that of Experiment 2, in which SPVs are not included. The remainder of this chapter discusses the design and setup for each of these experiments.

5.1 Deep Q-Learning Cooperative Automated Driving Rerun

The goal of Experiment 1 is to establish a baseline setup for the DQL closed loop system for cooperative autonomous driving, which was done by replicating research conducted by [8]. Chandramohan et al. (2019) developed a DQN driving algorithm that manoeuvres a two-lane highway environment using only information from vehicles that are immediately surrounding the AV, namely, the PPVs. The DQL closed loop system used in Experiment 1 is illustrated in Figure 13. Recall from Section 4.3 that RL has two main components: the environment and the agent. In this particular experiment an additional component called the interpreter is included, which serves as an interface between the two main components.

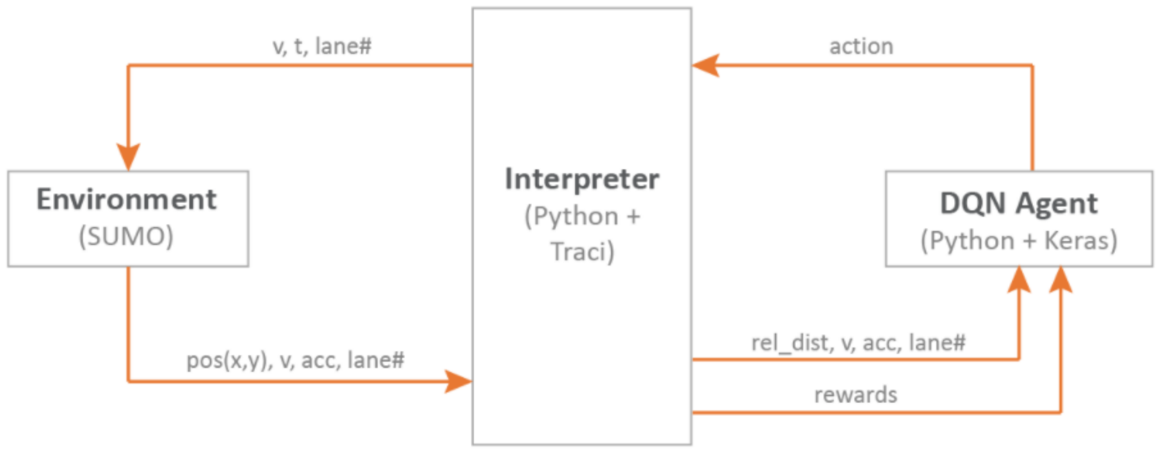


Figure 13: Closed loop system of the DQN agent used for cooperative autonomous driving.

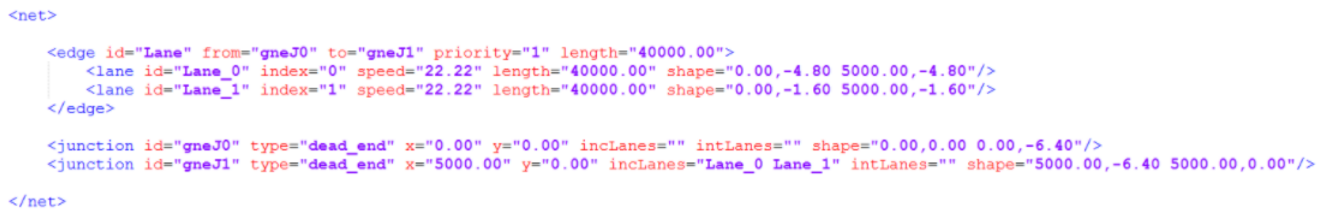
The remainder of this section discusses the implementation and design of each component used in Experiment 1 to establish the baseline.

5.1.1 Environment

The environment is a simulation of an AV navigating in a two-lane highway with traffic. In this research, the environment was developed in SUMO, an open source traffic simulation software that allows design of road networks and traffic scenarios [51].

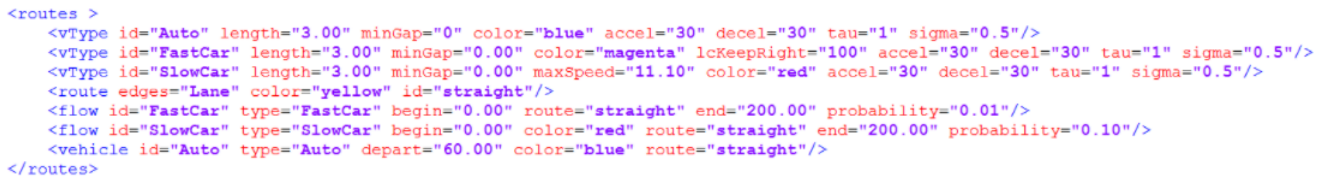
SUMO consists of a suite of tools that allow users to model the different subsystems of an AV simulation. In SUMO, each simulation requires a sumo network file (.net.xml), route file (.rou.xml) and a sumo configuration file (.sumocfg.xml). The network file describes the roads and intersections that simulate vehicles driving along or across. The route file describes the vehicle types, possible routes, and the flow of traffic within the simulation. The sumo configuration file describes how the simulation will be executed by specifying the network and route files to use, the duration of the simulation, how to handle collisions, etc. The three required files were generated with NetEdit, a SUMO tool that allows users to specify layouts and rules for a given road network, create routes, add vehicles, among other specifications. A screenshot of all three files used in this experiment are provided in Figures 14, 15, and 16.

The environment in this experiment is an AV simulation. Recall from Chapter 3 that an AV simulation contains 2 major sub-models, namely the scene and the AV. The process of designing and configuring the two sub-models of an AV simulation will be discussed in the remainder of this section.



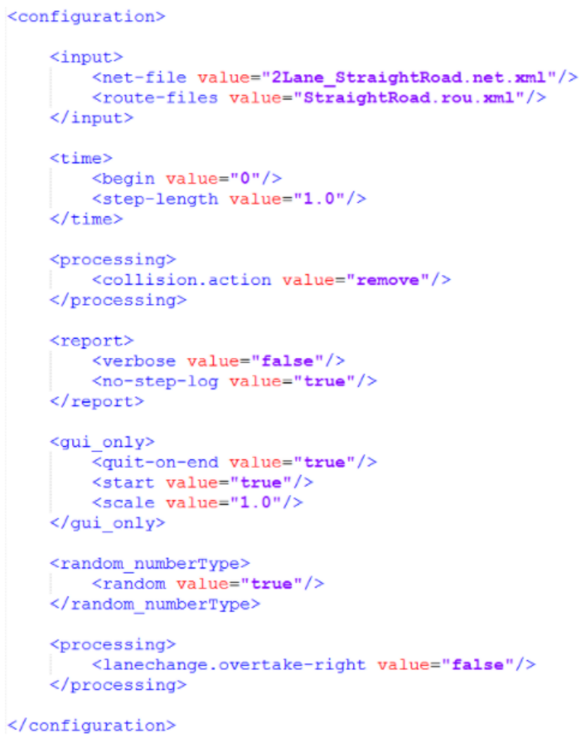
```
<net>

    <edge id="Lane" from="gneJ0" to="gneJ1" priority="1" length="40000.00">
        <lane id="Lane_0" index="0" speed="22.22" length="40000.00" shape="0.00,-4.80 5000.00,-4.80"/>
        <lane id="Lane_1" index="1" speed="22.22" length="40000.00" shape="0.00,-1.60 5000.00,-1.60"/>
    </edge>

    <junction id="gneJ0" type="dead_end" x="0.00" y="0.00" incLanes="" intLanes="" shape="0.00,0.00 0.00,-6.40"/>
    <junction id="gneJ1" type="dead_end" x="5000.00" y="0.00" incLanes="Lane_0 Lane_1" intLanes="" shape="5000.00,-6.40 5000.00,0.00"/>

</net>
```

Figure 14: Example of a SUMO network file.



```
<routes >
    <vType id="Auto" length="3.00" minGap="0" color="blue" accel="30" decel="30" tau="1" sigma="0.5"/>
    <vType id="FastCar" length="3.00" minGap="0.00" color="magenta" lcKeepRight="100" accel="30" decel="30" tau="1" sigma="0.5"/>
    <vType id="SlowCar" length="3.00" minGap="0.00" maxSpeed="11.10" color="red" accel="30" decel="30" tau="1" sigma="0.5"/>
    <route edges="Lane" color="yellow" id="straight"/>
    <flow id="FastCar" type="FastCar" begin="0.00" route="straight" end="200.00" probability="0.01"/>
    <flow id="SlowCar" type="SlowCar" begin="0.00" color="red" route="straight" end="200.00" probability="0.10"/>
    <vehicle id="Auto" type="Auto" depart="60.00" color="blue" route="straight"/>
</routes>
```

Figure 15: Example of a SUMO route file.



```
<configuration>

    <input>
        <net-file value="2Lane_StraightRoad.net.xml"/>
        <route-files value="StraightRoad.rou.xml"/>
    </input>

    <time>
        <begin value="0"/>
        <step-length value="1.0"/>
    </time>

    <processing>
        <collision.action value="remove"/>
    </processing>

    <report>
        <verbose value="false"/>
        <no-step-log value="true"/>
    </report>

    <gui_only>
        <quit-on-end value="true"/>
        <start value="true"/>
        <scale value="1.0"/>
    </gui_only>

    <random_numberType>
        <random value="true"/>
    </random_numberType>

    <processing>
        <lanechange.overtake-right value="false"/>
    </processing>

</configuration>
```

Figure 16: Example of a SUMO configuration file.

5.1.1.1 Scene Modeling

Recall from Chapter 3 that the scene model of an AV simulation includes elements such as roads, traffic, infrastructure and weather. As mentioned previously, the SUMO network file and route file contain the information for the design and modeling of the scene, as displayed in Figures 14 and 15.

Road Network

The road network used in this experiment is a straight two-lane highway. In SUMO each road segment is defined by two junctions (referred to as endpoints) and an edge, which is a street connecting two junctions. Figure 14 illustrates the positioning of junctions gneJ0 and gneJ1 at (0,0) and (40000,0) on a cartesian plane, respectively. The width of the road is specified to be 6.40 m using the shape attribute for the junctions. Only one road segment is used in this experiment and therefore only a single edge is defined, named "highway". The edge is parametrized to connect junctions gneJ0 and gneJ1. Figure 14 also illustrates two lanes, namely "Lane_0" and "Lane_1", which are specified for the edge, making it a two-lane road segment. The speed limit (m/s) and length (m) for both lanes are defined as 22.2 m/s and 40,000 m respectively. Based on the configuration of the shape attribute of "Lane_0 " and "Lane_1", both lanes will have a width of 3.2 m. All additional parameters of defining the road network are summarized in Table 5 below.

Traffic

Traffic modeling includes modeling of vehicles in the simulation, traffic density and traffic flow, amongst other parameters. Traffic models are defined in the SUMO route file.

In this experiment, the traffic model simulates "high density" and "low flow" traffic, which refers to the number of vehicles per unit of road and the number of vehicles per unit of time,

Table 5: SUMO network file attribute description.

Level 1 Design	Attributes	Description
junction	id	Junction identifier
	type	Type of junction
	x	x-coordinate of the junction
	y	y -coordinate of the junction
	incLanes	The ids of the lanes that end at the intersection; sorted by direction, clockwise, with direction up = 0
	intLanes	The IDs of the lanes within the intersection
	shape	A polygon describing the road boundaries of the intersection
edge	id	Edge identifier
	from	Junction's id where edge starts from
	to	Junction's id where edge ends
	length	Distance of the edge in meters
lane	id	Lane identifier
	index	A running number, starting with 0 at the right-most lane
	speed	Maximum speed allowed in the lane (m/s)
	length	Length of the lanes
	shape	Position vector that specifies the center line geometry of the lane

respectively. This traffic model will facilitate opportunities and challenges for the AV to overtake other vehicles and navigate autonomously. One important element of traffic modeling is defining the types of vehicles that will be involved in the model. Two types of traffic vehicles are used in the experiments in this research, specified by "vType": 1) slow moving vehicles and 2) fast moving vehicles, identified with 'id' parameters "SlowCar" and "FastCar", respectively. Slow moving vehicles were included in the study to easily create overtaking scenarios, whereas fast moving vehicles were included to create complicated scenarios for the AV to navigate. "SlowCar" and "FastCar" can travel at a maximum speed of 11 m/s and 22.22 m/s, respectively. To simulate high density traffic, both vehicle types are configured to maintain a one second headway gap ("tau") when following other vehicles. To simulate variation in traffic flow, the "sigma" attribute, which models the driver's imperfection in driving (where 0 denotes perfect driving), was set to 0.5 for both vehicle types. Note that vType "Auto" is the modeling of the AV, which will be discussed in Section 5.1.1.2. When simulating traffic it is important to define the routes that vehicles will travel; in this experiment, only one route is used along the one way "highway" edge defined as "straight" in the network file. Traffic density and flow is affected by the frequency of which vehicles enter the simulation; a macroscopic modeling of emission is done using "flow". The "begin" and "end" attributes specify the time when the emission of multiple instances of the vehicle type will begin and stop, respectively. The start and stop time of emission for both vehicle types were specified to be the first 200 seconds of the simulation. The "probability" attribute specifies the probability for a vehicle to be emitted randomly at each time step. The values set for slow moving vehicles ("SlowCar") and fast moving vehicles ("FastCar") are 0.1 and 0.01, respectively; at each time step, there is an 11% chance that a vehicle will be emitted for the duration of emission. Slow-moving vehicles had 10 times the probability to be emitted to fast-moving vehicles, simulating low traffic flow. The "flow" attribute is used for defining entry and route of repeated vehicles while the "vehicle" attribute is used to define entry and route for a single vehicle and is used for configuring the AV in this experiment. During experiments in this research the AV is emitted (departs) at the 60 second mark such that by the time the AV enters, other vehicles are already present on the route. The AV is emitted with an initial speed (departSpeed) of 11.1 m/s.

While other subsystems such as weather can be modeled as part of traffic modeling, they were omitted in this experiment as they are outside the scope of this research. A summary of the route file attributes used in this experiment can be found in Table 6 below.

Table 6: SUMO route file attribute description.

Level 1 Design	**Attributes**	**Description**
vType	id	Name of the vehicle type
	length	Length of the vehicle in meters
	minGap	Empty space after leader
	color	Vehicle type's color
	lcKeepRight	The eagerness for following the obligation to keep right
	accel	The acceleration ability of vehicles of this type (in m/s^2)
	decel	The deceleration ability of vehicles of this type (in m/s^2)
	tau	The driver's desired (minimum) time headway
	sigma	The driver imperfection (0 denotes perfect driving)
route	id	Name of the route
	edges	The edges the vehicle shall drive along
	color	This route's color
flow	id	Name of emissions
	type	Type of vehicles that will be emitted
	begin	First vehicle departure time
	route	Name of route vehicles will be emitted
	end	End of departure interval
	probability	Probability for emitting a vehicle each second
vehicle	id	Name of emission
	type	Type of vehicle that will be emitted
	depart	Determines the time at which the vehicle enters the network
	route	Name of route vehicle will be emitted
	departSpeed	Initial speed vehicle that gets emitted

5.1.1.2 Autonomous Vehicle Modeling

As previously discussed, the second main component of an AV simulation is the AV model. Recall from Chapter 3 that the AV model can be further divided up into four main components, namely the vehicle model, vehicle control model, cognition model and perception model. The AV's vehicle model was modeled in the RL environment, discussed in this section; the perception model and vehicle control model was implemented separately in the interpreter (discussed in Section 5.1.3) and the cognition model implemented in the DQN agent (discussed in Section 5.1.1.2.2).

5.1.1.2.1 Vehicle Model - Vehicle Body

The AV vehicle model is specified in the SUMO route file. In this experiment the AV vehicle is defined with the "Auto" vType identifier. "Auto" has similar attributes to the other vehicle types ("SlowCar" and "FastCar") detailed in Section 5.1.1, except for its maximum speed which is set to 55.55 m/s, allowing the AV to be capable of exceeding the speed limit. This is advantageous in the model as it provides opportunities for the agent to learn to perform or avoid performing certain behaviours.

Modeling of mechanical systems such as tire, braking and transmission models (amongst others) was not required since vehicle dynamics is not relevant to the scope of this experiment; instantaneous vehicle behaviour is sufficient for the intended driving algorithm.

5.1.1.2.2 Cognition Model

The cognition model used in the experiments in this thesis is one that is external to the driving algorithm and therefore needs to be implemented. The forward collision avoidance system (FCAS) cognition model is external to the DQN driving algorithm and was used to detect forward vehicles that are very close to the AV and automatically brake to avoid rear-end collisions. FCAS was implemented in SUMO and is enabled by setting "tau" in the route file (Appendix A) to a value greater than or equal to the time step duration. The FCAS will automatically brake if the AV comes very close to the vehicle in front of it (within 3 meters – set by "minGap"). An external FCAS is enabled for the experiments in this research because in previous research [8] it was found that the collision rate was very high when only the trained driving algorithm was responsible for avoiding forward collisions and FCAS was not enabled. Enabling FCAS means there is an external entity that prevents forward collisions, and the driving algorithm can only focus on maneuvering through the traffic. This design was preserved throughout the experiments in this thesis; this was deemed reasonable given that most vehicles have and use FCAS. It is important to note that even though FCAS is implemented, collisions are still possible during lane change situations.

5.1.2 DQN Agent

As shown in Figure 13, the DQN agent is one of three important components of the DQL closed loop system. The DQN agent is composed of two algorithms: a driving algorithm and a learning algorithm. The driving algorithm is responsible for making decisions for cooperative driving on a highway with the goal of maximizing rewards over time, and is used both during training and testing of the model. On the other hand, the learning algorithm is only used during the training period, in which the driving algorithm's DQN is updated to optimize the prediction of the Q-values (which determines the decisions that are taken). The learning algorithm is responsible for using data from the environment and value signals generated from the value function (Bellman equation) to update weights and biases (learnable parameters) of the DQN with the aim of accurately predicting Q-values for the action space given a state.

5.1.2.1 DQN Agent Driving Algorithm

In this experiment, the driving algorithm is a part of the DQN agent which can be considered the cognition model of the AV in a SIL implementation. Recall from Section 4.4.2 that a DQN agent uses a DNN, implemented in this experiment with Tensorflow and Keras. Tensorflow is an open source software library based in Python, used for numerical computation using dataflow graphs. It contains a wide range of functionalities for ML mainly related to DNN. Keras is a high level neural networks API capable of running on TensorFlow. It provides simple APIs for most ML tasks such as learning, model creation and modification of neural networks. The list of APIs used from Keras for this research is given in Appendix B: External APIs Used. The DQN is designed to both make driving decisions and learn how to make those decisions, both of which are part of the learning algorithm.

Figure 33 illustrates the architecture of the DQN used in this experiment, which contains an input layer, three hidden layers and an output layer. The input layer takes in the input features that represent the state of the AV within the highway environment. Distance and velocity information of PPVs is deemed important to help the driving algorithm make decisions since they immediately dictate the actions that are safe. Knowledge of the current distance of vehicles will allow driving algorithms to make better decisions. For example, if the distance between the AV and the vehicle in front of it is decreasing, the AV should change to the overtaking lane (left lane change), provided the lane is free. Knowledge of the velocity of the PPVs becomes important here; for example, if the vehicle in front of the AV is close but travelling at a high velocity, there is no need for the AV to move to the overtaking lane. Each PPV is indexed according to its location relative to the AV. More specifically, a vehicle that is in the same lane as the AV and is travelling immediately in front of the AV is indexed 1, and one that is travelling immediately behind the AV is indexed 2. Vehicles that are travelling in the left of the AV and are positioned in front of and behind the AV are indexed 3 and 4, respectively. Finally, when the AV is in the left lane, vehicles that are travelling to the right of the AV and are positioned in front of and behind are indexed 5 and 6, respectively. Figure 18 and Figure 19 illustrate the notation used for input information from target vehicles when the AV is in the right lane (driving lane) or left lane (overtaking lane), respectively. Additionally, as indicated by these figures, d denotes the distance between the front of AV and the front of each of the surrounding vehicles, using the same indexing convention described above.

One way that the driving algorithm controls the vehicle is by accelerating or decelerating, based on certain conditions. Hence, it is vital for the driving algorithm to have information about the current acceleration rate and speed of the AV, denoted as "vacc" respectively. Given that the AV will behave differently in the two lanes based on certain conditions, vacc is required as an input to the driving algorithm.Table 7 summarizes the input features selected for this experiment.

The output layer of the DQN generates the Q-values for the action space of the driving algorithm, which are basic actions a vehicle can perform such as changing lanes or changing velocity. Hence the action space includes the following:

1. Idle

2. Left lane change

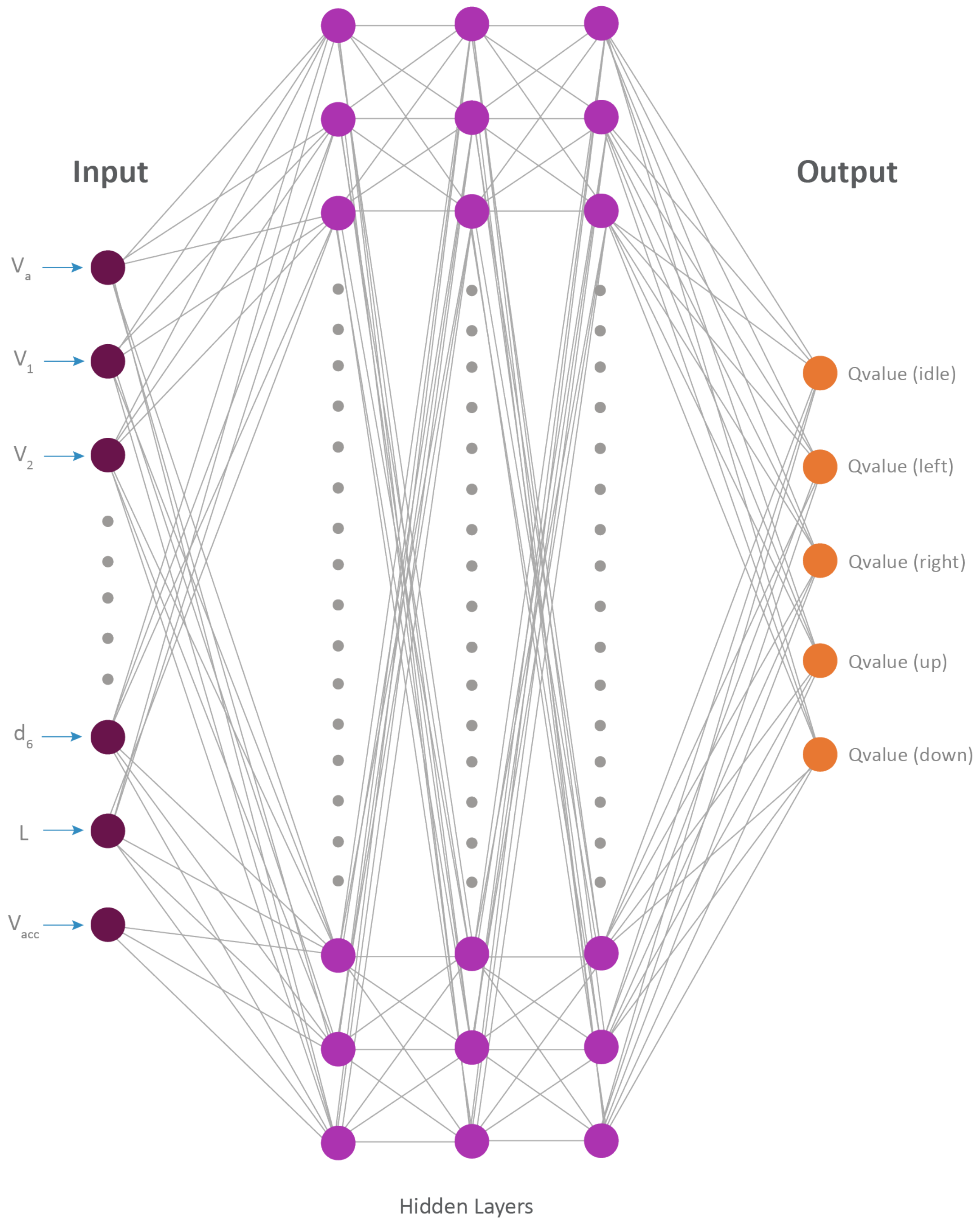


Figure 17: DQN architecture with input layer, three hidden layers and output layer used in experiments.

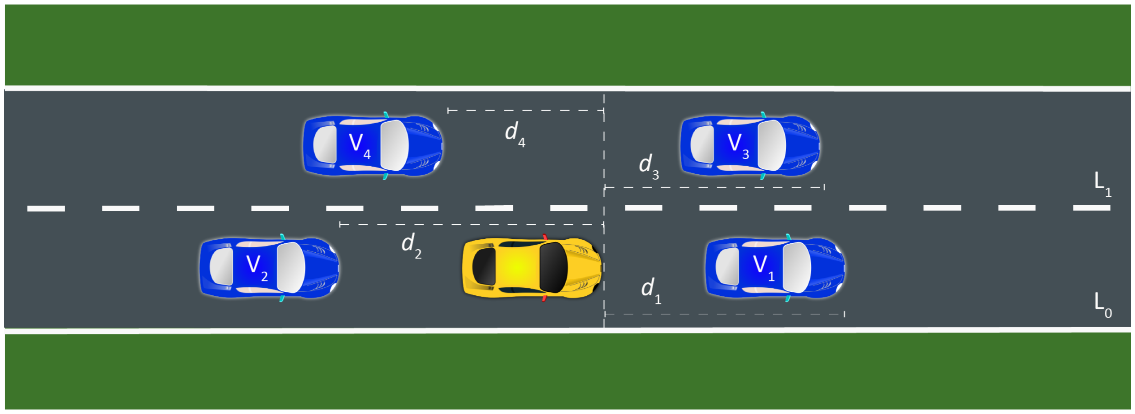


Figure 18: AV driving in the driving lane (L0), with input feature notation for PPVs.

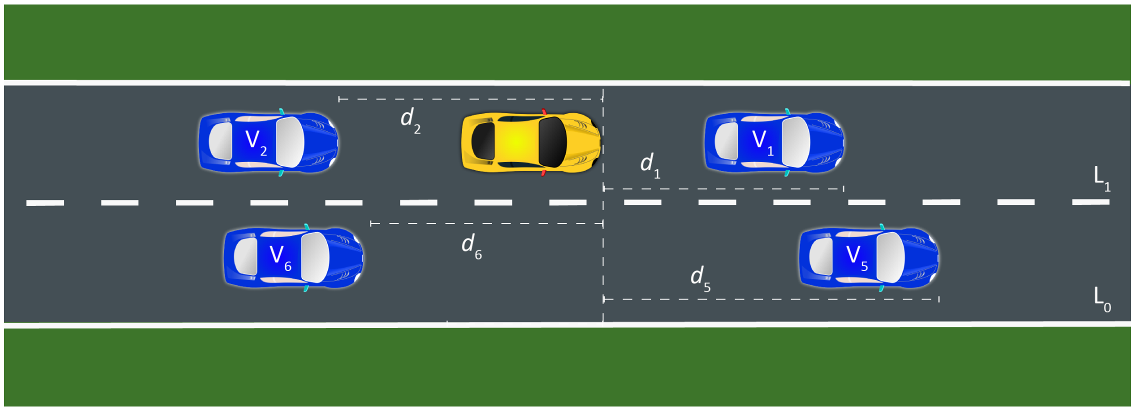


Figure 19: AV driving in the overtaking lane (L1), with input feature notation for PPVs.

Table 7: Input features

Features	**Description**
v_a	Current velocity of the autonomous vehicle
v_1	Current velocity of the vehicle immediately in front of the AV in the same lane
v_2	Current velocity of the vehicle immediately behind the AV in the same lane
v_3	Current velocity of the vehicle immediately in front of the AV in the left lane
v_4	Current velocity of the vehicle immediately behind the AV in the left lane
v_5	Current velocity of the vehicle immediately in front of the AV in the rightlane
v_6	Current velocity of the vehicle immediately behind the AV in the right lane
d_1	Current distance of the vehicle immediately in front of the AV in the same lane
d_2	Current distance of the vehicle immediately behind the AV in the same lane
d_3	Current distance of the vehicle immediately in front of the AV in the left lane
d_4	Current distance of the vehicle immediately behind the AV in the left lane
d_5	Current distance of the vehicle immediately in front of the AV in the rightlane
d_6	Current distance of the vehicle immediately behind the AV in the right lane
L	Current lane the AV is occupying (L=0 to maximum number of lanes)
v_{acc}	Current acceleration rate of AV

3. Right lane change

4. Increase speed

5. Decrease speed

Given the possible actions that can be performed, the output layer of the DQN contains 5 nodes (neurons), each generating a Q-value to a mapped action in the action space.

As shown in Figure 33, the DNN used in this experiment contains three hidden layers, each containing 1500 neurons. A sequential neural network was used for the DNN where the layers are stacked in front of each other so that data can propagate through them. The ReLu function (see Section 4.4.2) was used as the activation function for the input and hidden layers, whereas a linear activation function was used for the output layer.

Recall from Section 4.4.2 that the driving algorithm decides which action the agent should make by choosing the action that has the highest predicted Q-value from the output layer, illustrated in the Equation 3:

$$a_t = maxQ(s_t, a) \tag{3}$$

Where Q is the DQN, and a is the Q-value of the action space at the output layer given the current input (s_t).

5.1.2.2 DQN Agent Learning Algorithm

The second component of the DQN agent is the learning algorithm. During training, the learning algorithm uses experience replay and the Bellman equation (value function) to help update the DQN to predict Q-values for the action space of each state. In a standard learning algorithm, the knowledge gained by a certain action is discarded as soon as the action is performed. This is different in experience replay, where at each time step a "transition" is stored in a memory buffer of size 2000. Each transition stored consists of the following values: $s_t, a_t, r_t, s_{t+1}, done$. When the buffer becomes full with 2000 transitions, the data inside gets overwritten in a first-in first-out format. Using experience replay in this manner allows the learning algorithm to learn more effectively and efficiently during the training period as it does not just learn from actions taken during the simulation but also from the results of previous simulations. Once the buffer is full, a batch of 32 transitions are uniformly selected in a random manner from the memory buffer (from a total size of 2000) to be used as input for training the DQN. In this experiment, this random transition selection process is performed at each subsequent timestep. Given that the transitions from the memory buffer in this experiment are selected randomly and with uniform probability, this sampling method is referred to as uniform experience replay (UER). Algorithm 1 provides a pseudo code of how experience replay is implemented in this experiment.

Algorithm 1: Uniform Experience Replay Algorithm

```
input : memory size $N$, minibatch size $m$, learning rate $\alpha$, discount factor $\gamma$, Total steps
        $T$, initial weights $\theta$, update policy $\pi_\theta$
Initialize replay memory buffer with capacity $N$
Observe initial state $s_0$
for t=1 to T do
    take action $a_t$ based on $\pi_\theta(s_t)$
    observe $r_t$ and $s_{t+1}$
    store transition $(s + t, a_t, r_t, s_{t+1})$ in the memory buffer
    for j=1 to m do
        sample a transiton $(s + t, a_t, r_t, s_{t+1})$ randomly from the memory buffer
        retain the neural network and update its weights
    end
end
```

Once a batch is selected at a timestep (t) using UER, the learning algorithm begins the learning process; this is the point where weights and biases are updated. Initially, the weights and biases are assigned random values. Recall from Section 4.4.2, that a DNN learns through the process of backpropagation, where the error of the prediction at the output layer is used to update the weights and bias – iterating backwards – to reduce the error. The error is the difference between the expected Q-value (calculated using the Bellman equation) and the predicted Q-value for the selected action. The equation used in this experiment to calculate the expected Q-value for the state (s_t) and action (a_t) for the current time step is as follows:

$$Q_{expected}(s_t, a_t) = r_{t+1} + \gamma * Q(s_{t+1}, a_{max}) \tag{4}$$

$$a_{max} = argmaxQ(s_{t+1}, a) \tag{5}$$

The transitions stored in the memory buffer contain all of the values that are required to compute the above equations. Based on the equations (4 and 5), the expected Q-value (value signal) for the action taken (a_t) when the state of the environment was (s_t) is calculated by adding the reward received for being in the new state (r_{t+1}), to the estimated future value discounted by γ, which in this experiment was set to 0.9 – this ensures that the learning algorithm maximizes the long term rewards. The estimated future value is computed to be the maximum Q-value of the action space for the new state (s_{t+1}) using the DQN (Q). The maximum Q-value is extracted by using the action that has the maximum predicted Q-value (a_{max}), which is determined by equation 5. The DQN will then use the error calculated from the expected Q-value and predicted Q-value to update the weight and biases using backpropagation. The rate at which the DQN learns is determined by the learning rate, which was chosen to be 0.0001 for this experiment. The learning rate is a very low value, and was selected so that the Q function will have a higher chance of converging to its optima without exceeding it. This learning process ensues at each time step after learning begins, once the memory queue becomes full.

5.1.3 Interpreter

Although represented separately in the closed-loop system in Figure 13, the interpreter can be considered an extension of the environment and the DQN agent. The interpreter serves the following three functions in all experiments in this thesis:

1. Initializing and controlling the simulated environment
2. Interfacing between the environment and the agent (perception model and vehicle control model)

3. Determining rewards

The interpreter was implemented using Python.

The interpreter connects the RL's environment developed in SUMO and the DQN agent developed in Python with the Traffic Control Interface (TraCI) API library. TraCI is a Python library that enables access to functionalities inside SUMO from Python. For example, in this research TraCI is used for controlling the AV via the driving agent's decisions, controlling the simulation (start, stop, etc.) and retrieving vehicle information from SUMO (perception model). The TraCI API calls are provided in Appendix B.

5.1.3.1 Environment Simulation Control

Initializing and controlling the simulated environment is done by the interpreter. This functionality is invoked with a Python class inherited from the Gym Python library [52]. Gym is a toolkit used for developing environments used in RL. It provides the structure for interfacing with an environment that can be used by the learning algorithm. The main structures from Gym that were used in this research include:

1. Initialization state: initializes the environment, input states and actions
2. Reset state: sets up the environment in SUMO and begins the simulation. This state is maintained until the AV has become active in the simulation. The learning algorithm only begins once this state is complete.
3. Step State: Executes the action in the environment each time an action is taken by the driving algorithm.

5.1.3.2 Perception Model

The second function of the interpreter is to serve as an interface between the environment and the agent. As can be seen from Figure 13, the interpreter interface is bidirectional, moving information from the environment to the agent as well as from the agent to the environment. The interface from the environment to the agent is the AV's perception model (discussed in this section), whereas the interface from the agent to the environment is the AV's vehicle control model (discussed in the next section).

Recall from Section 5.1.2 that the driving algorithm requires distance and velocity from PPVs in order to make decisions. This information is captured by a V2V perception model. The model extracts velocity and position information from vehicles within an 800 m range of the AV. Only information from PPVs is included in this experiment. Velocity and position information of the PPVs is filtered and passed to the DQN agent as v1 - v6 and d1 - d6 input features. Distance is calculated by subtracting the positions of the PPVs from the position of the AV (see Section 5.1.2). V2V communication was modeled to be ideal such that there was no loss of communication or corrupted information transmitted.

The driving algorithm also requires information about the AV's vehicle dynamics. To model proprioceptive sensors such as encoders, IMUs and GPUS, the AV's velocity, acceleration and position (x, y) are passed to the driving algorithm. The lane number is also passed to the driving algorithm by the interpreter modeling a lane detection system.

5.1.3.3 Vehicle Control Model

The interpreter also arbitrates the action (idle, lane change right, lane change left, speed up or speed down) selected by the driving algorithm, similar to how a vehicle control module arbitrates control signals from the cognition modules. In this research, the vehicle control model includes throttle, braking and steering control. The model takes the action (idle, lane change right, lane change left, speed up or speed down) from the agent as input, and outputs the target velocity (v_{target}), time (t) to achieve target velocity, and the lane number of the AV (L_a). The throttle and brake control is responsible for arbitrating the speed up and speed down actions, respectively. To change the speed of a vehicle in SUMO (in this case the AV) using TraCI, an API call (traci.vehicle.slowDown – see Appendix B) from the library is used, which requires a vehicle identifier, the target speed and time as inputs. This will accelerate or decelerate the specified vehicle in the simulation to the target speed in the time specified. The throttle or brake control model is responsible for translating the action taken by the DQN agent to speed and time signals required by the TraCI API call for controlling speed. The throttle and brake control models used in this research are similar, with the only difference being that one is responsible for accelerating while the other is responsible for decelerating and they do so at different rates. The first step in each model is to calculate the acceleration (acc) or deceleration (dec) rate. Constant acceleration increments of 1.26 m/s^2 and deceleration increments of -0.63 m/s^2 are used by the throttle and braking control models, respectively. Using the throttle control model as an example, the first time speed-up is selected as an action in a sequence, the acceleration rate of the vehicle will be 1.26 m/s^2. If the next action is also speed-up, then the new acceleration rate will be 1.26 * 2 = 2.52 m/s^2. Similarly, if the subsequent action is also speed-up, then the new acceleration rate will be 1.26 * 3 = 3.78

m/s^2. This pattern will continue with the rate being 5.04 m/s^2 if speed-up is selected as the action for the fourth time. Following this, if the next action is not a speed-up, then the throttle control model's acceleration rate gets reset to 1.26m/s^2. The same concept is applied for the brake control model, in which repeated consecutive actions of speed-down result in increased deceleration (dec) by increments of -0.63 m/s^2. The acceleration (acc) and (dec) is then used to compute the target speed (v_{target}) and time ($t_{acceleration}$ or $t_{deceleration}$) signals.

Speed (v_{target}) and time(t) signals are computed by the equations below:

$$v_{target} = v_t + acc \tag{6}$$

$$v_{target} = v_t + dec \tag{7}$$

$$t_{acceleration} = (v_{max} - v_t)/acc \tag{8}$$

$$t_{deceleration} = (0 - v_t)/dec \tag{9}$$

Equation 6 adds the acceleration rate acc to the current speed of the AV to calculate the target speed v_{target}. The time the AV should accelerate ($t_{acceleration}$) is derived by calculating the time it would take for the AV to reach the maximum speed (v_{max}) from the current speed (v_t) with the computed acceleration rate (acc) as seen in Equation 8. The AV will then try to reach its target speed in the calculated time, provided that it does not receive an action to decelerate within that time period. The time for deceleration ($t_{deceleration}$) is computed by calculating the time it takes for the AV to come to rest (0 m/s) with the calculated deceleration rate (dec) as seen in Equation 9. The AV will then reduce its speed gradually to the target speed within the given time provided that an action for acceleration does not occur during that time.

The steering control model is responsible for arbitrating the lane change actions selected by the DQN driving algorithm. The model is quite simple, as it uses the TraCI's API call traci.vehicle.changeLane, which requires a vehicle identifier, the lane number the vehicle should change to, and the time for the change to occur (see Appendix B).

5.1.3.4 Rewards

The final function of the interpreter is to determine the reward (r_{t+1}) for the new state (s_{t+1}) that results from each action taken. Table 8 contains the rewards that are given to undesirable and desirable traffic situations. As expected, in the event of a collision, a very high negative reward ($R_{Collision}$) of -101 is implemented, as this is the most undesirable outcome. $R_{Collision}$ should be high enough to discourage collisions but not too high that the DQN agent only learns to be a car-following model. It is important to note that it is assumed that the target vehicles in the simulation are following traffic rules and therefore only the AV can be responsible for collisions. A large negative reward ($R_{Standstill}$) of -50 is given if the the speed of the AV (v_a) comes to rest on the highway as this causes a traffic jam. The proximity distance d represents the minimum gap that is considered safe and is set to a value of 160 m. When the AV comes within the close proximity distance to the vehicle in front ($d_1 < d$), a reward ($R_{Proximity}$) of -5 is given. For each time-step the AV spends within this range, the negative reward adds up. The rationale for this reward is that once the driving agent senses that it is receiving this negative reward, it will learn to change lanes or slow down to increase d_1.

One desirable situation that is awarded a positive reward ($r_{LaneChange}$) is when the AV changes lanes to overtake a vehicle in the driving lane and the AV is accelerating ($v_{acc} > 0$). This reward is dynamic in that it depends on the value of d_5, the distance of the vehicle in front of the AV in the right lane. The reward is set as 50 minus d_5; as d_5 decreases while the AV tries to overtake, the reward will increase during the overtaking period. This rewarding system helps to ensure that the AV overtakes quickly and does not slow down unnecessarily during the overtaking period. If the AV slows down ($v_{acc} < 0$) while overtaking ($L_a = 1$) because there is also a vehicle in the overtaking lane that is in close proximity ($a_1 < d$), a positive reward ($R_{Lane1AprroachingVehicle}$) of 0.5 is given. If the AV does not decrease its speed in this situation a reward of -0.5 is given. To prevent the AV from staying in the overtaking lane unnecessarily, a reward (r_{Lane1}) of $-1.5 * d_5$ is given when the AV is in the overtaking lane ($L_a = 1$) and there is no leading vehicle in close proximity in the driving lane ($d_5 > d$).

As discussed in the Road Network section in Section 5.1.1, the speed limit for the highway is 22.2 m/s; a negative reward ($R_{Overspeeding}$) of -1 is given when the AV exceeds this speed limit ($v_a > SpeedLimit$). A positive reward ($R_{MaintainSpeedLimit}$) of 2 is given if the AV has a speed that is equal to the speed limit ($v_a = SpeedLimit$). A smaller positive reward ($R_{Speeding}$) of 1 is given when the vehicle accelerates towards the speed limit ($v_{acc} > 0$).

Table 8: Reward function

S.No	**Reward**	**Reward Condition**
1	-101	Collision
2	-50	$Else\&v_a = 0$
3	-5	$Else\ \&\ L_a \neq L_{max}\ \&\ d_1 < d_{proximity}$
4	$50 - d_5$	$Else\ \&\ L_a \neq L_0\ \&\ d_5 < d_{proximity}\ \&\ a_a > 0$
5	$-1.5 * d_5$	$Else\ \&\ L_a \neq L_0\ \&\ d_5 > d_{proximity}$
6	0.5	$Else\ \&\ L_a = L_{max}\ \&\ d_1 < d_{proximity}\ \&\ a_a < 0$
7	-0.5	$Else\ \&\ L_a = L_{max}\ \&\ d_1 < d_{proximity}\ \&\ a_a > 0$
8	-1	$Else\&v_a > SpeedLimit$
9	1	$Else\&a_a > 0t$
10	2	$Else\&v_a = SpeedLimit$
11	0	$Else$

5.1.4 Training Process Design

Recall that the weights of the neural network at the beginning are assigned random values, and therefore need to be trained. The training period in this experiment consists of 7000 simulation episodes. An episode is an entire simulation spanning from the moment the AV is inserted into the simulation to when the simulation comes to an end either due to the end of the simulation time or due to a collison. The experiments in this research had a set simulation time of 100 steps, with each step being 1 second long specified by the "step-length" in the SUMO configuration file (see Figure 16). The DQN agent is trained during each 1 second time step.

As discussed in Section 4.3.1.1, there is a trade-off between exploration and exploitation that must be managed. The experiments in this research use the greedy policy to manage this trade-off, in which exploration and exploitation are chosen randomly. The greedy policy chooses to explore or exploit based on a probability referred to as epsilon or exploration rate ε. A random number is chosen between 0 and 1 and if the number is less than ε then the policy chooses to explore, where a random action is selected. On the other hand, if the random number is greater or equal to ε then the policy chooses to exploit, in which case the driving algorithm chooses the action. Initially ε is set to 0.9 to ensure that the driving agent can learn the consequences from random actions. Subsequently, ε decreases at each episode exponentially by a factor of 0.9992 until it reaches the minimum epsilon of 0.1 as seen in Equation 10.

$$\varepsilon = 0.9 * 0.9992^{episode} \tag{10}$$

Once ε reaches this minimum value, actions are taken purely based on the output of the driving algorithm.

5.1.5 Testing Process Design

Once the training process is complete and the agent has been successfully trained, it is cross-validated by evaluating the agent with new scenarios. The testing period consists of 500 simulation episodes; similar to the episodes from the training period, each episode is scheduled to last 100 steps and each time step is 1 second long. An episode is terminated once the 100 time steps have been reached or if a collision occurs. Each of the 500 episodes during the testing process are unique. To ensure consistency and comparable results, the same set of 500 episodes are used to perform testing on all agents.

5.2 Deep Q-Learning Cooperative Automated Driving Revise

The goal of this experiment was to improve the baseline design used by Chandramohan et al. (2019). Three areas of improvement were identified that could improve the training and performance of the driving algorithm, namely stability, training efficiency and implementing a time headway instead of a distance headway for the proximity parameter.

5.2.1 Double DQN

As explained previously, the DQN learning algorithm uses the same network to both select and also evaluate an action when estimating the future value, as quantified by Equation 4 and 5. The algorithm always uses the maximum value to evaluate an action, causing overestimated values to be used. This leads to overoptimistic value estimates when selecting an action. Research has shown that overestimations lead to suboptimal policies, which result in instability, characterized as frequent policy changes in the event of slight modifications to the training data. This is problematic as it creates variations in policy performance [53]. One way to prevent overoptimistic value estimates is to decouple the selection of the action value pair from the evaluation of the value for the action. This decoupling strategy can be achieved via a Double DQN (DDQN). In the DDQN algorithm, two networks are used referred to as the main network (Q) and target network (Q'), respectively. The updated equations for determining the expected value ($Q_{expected}$) for a DDQN agent are shown in Equation 11 and 12.

$$Q_{expected}(s_t, a_t) = r_{t+1} + \gamma * Q'(s_{t+1}, a_{max}) \tag{11}$$

$$a_{max} = argmaxQ(s_{t+1}, a) \tag{12}$$

As can be seen in Equation 11 the main network Q is used to select the action with the maximum Q-values from the action space a of the new state s_{t+1}. Following this, the target network Q' is used to evaluate the Q-value of the action selected, thereby decoupling the selection and evaluation process. The weights and biases θ of the main network are copied to the target network every few steps so that $\theta = \theta'$, which was set to 5. Given the advantages to decoupling the selection and evaluation by using a DDQN, the learning algorithm from the DQN agent is updated to use DDQN for Experiments 2 and 3. To evaluate whether this helped to improve stability, the DDQN agent was tested and compared to the DQN agent control.

5.2.2 Priority Experience Replay

Recall that in Experiment 1 UER was used in the training of the RL agent. As discussed previously in Section 5.1.2, UER consists of storing experiences in a memory buffer that is sampled during the training process. The samples in the training batch are sampled randomly and uniformly, meaning that all samples have an equal probability of being selected across the experience history. Given that none of the experiences are prioritized, the DQN agent has an equal likelihood of being trained with experiences that are not as important (little discrepancy between expected and predicted Q-values) as experiences that are important (large discrepancy between expected and predicted Q-values). Therefore, in order to help improve the design of the original experiment, another modification that was implemented is the use of priority experience replay (PER) PER over UER in the learning algorithm of the DDQN. In PER, priority is assigned to each experience, thereby prioritizing the experiences and giving them varied probabilities. The measure that determines the priority of each experience is the difference between the predicted Q-value and what the Q-value was calculated to be, known as the TD error (δ_i). The TD error of an experience is a measure of how much the network can learn from the experience. The common way to set the priorities for experiences (p_i) is by adding a small constant ϵ to the TD error:

$$p_i =\mid \delta_i \mid +\epsilon \tag{13}$$

This ensures that even experiences with very low TD errors have an opportunity to be sampled. The ϵ used in this experiment was set to 0.1; therefore, $p_i =\mid \delta_i \mid +\epsilon$. The determined priority is then used to calculate the discrete probability $P(i)$ of sampling experience (i) shown in Equation 14:

$$P(i) = \frac{p_i^\alpha + \epsilon}{\sum_k p_k^\alpha} \tag{14}$$

Priority scale (α) is a parameter that scales the priority of experiences based on the TD error. If α is 0 then all priority terms go to 1 and every experience has the same chance of being selected, regardless of TD error. Alternatively, if is 1 then every sample is randomly selected proportional to the TD error (plus the constant ϵ), considered full prioritization. In this experiment α is set to a value of 0.7, so that prioritization takes place but not fully, leaving room for some exploration in addition to the PER process.

Another important aspect to PER is importance sampling (IS). When training, ideally the DQN agent will learn (via backpropagation) every state action pair the environment gives. With UER, the environment is approximated with the memory buffer, which gets sampled randomly and uniformly to accelerate the training. In UER, uniform random sampling ensures that the original distribution of experiences that the environment produces is preserved. However, in PER, prioritized sampling changes the distribution in favor of the higher prioritized experiences, making the DQN anticipate these experiences to a greater extent, consequently overfitting them. One solution is using IS, in which weights are applied to the TD error to try to correct for the aforementioned bias. Below is an expression of how the weights that are applied to TD error of each experience (i) are computed:

$$w_i = \left(\frac{1}{N} * \frac{1}{P(i)} \right)^\beta \tag{15}$$

This weight value is multiplied by the TD error, which has the same effect as reducing the gradient step during backpropagation. These weights can slow down the learning of certain experience samples with respect to others. N represents the number of experiences in the memory buffer. Based on the equation, it can be observed that the higher the probability of sampling, the lower the weight value will be. Because experiences with high priority and probability will be sampled more frequently under PER, this weight value ensures that the learning is slowed for such samples. Additionally, it is important to note that given that learning is chaotic at the beginning of the training process, prioritization bias at the start of training is inconsequential. The bias needs to be corrected near the end of the training and therefore, β is used to control how much IS is applied. When the value of β is 0, the weight is 1, resulting in IS not being applied at all. On the other hand, when β is 1, IS is applied fully. In this experiment the β value was set to $1 - \varepsilon$. Recall from Section 5.1.4 that the exploration rate ε is the probability of exploration, which is initially set to 0.9 and is reduced exponentially at each episode by a factor of 0.9992 until it reaches a minimum of 0.1. This ensures that the value of β will be low initially and progressively increases with each training, reflecting that IS is more important later in the training.

To compare results, the agent trained with PER was tested and compared to the UER based agent control to verify if efficiency was improved.

5.2.3 Time Headway

Recall from Section 5.1.3 that the proximity distance d in the reward function, which represents the minimum gap the AV should maintain between other vehicles to ensure safety, was set to 160 m. A constant distance headway of this nature is problematic given that using distance as a value for the minimum gap ignores velocity as a factor. More specifically, no matter the speed the AV is travelling, it must maintain a gap of 160 m from the other vehicles in the simulation. However, it is important to factor that a vehicle that is travelling at a high speed and is 160 m away has a higher chance of collision than a vehicle that is travelling at a lower speed and is 160 m away from another vehicle. Hence, another modification that was implemented to improve Experiment 1 was a constant time headway (instead of distance) to represent proximity. Using time is more accurate as it factors both the distance and the relative velocity of the associated vehicle(s). The time proximity ($t_{proximity}$) is a threshold that determines which time-to-collision (TTC) is deemed unsafe, where TTC is the time it takes for two vehicles to collide. TTC is calculated by using the following expression:

$$TTC_i = \frac{X_i - X_{av}}{v_{av} - v_i} \tag{16}$$

The time proximity used in this experiment was set to 3 seconds; the updated reward policy is shown in Table 9.

Table 9: Reward function with time headway proximity.

S.No	Reward	Reward Condition
1	-101	Collision
2	-50	$Else\&v_a = 0$
3	-5	$Else\ \&\ L_a \neq L_{max}\ \&\ TTC_1 < t_{proximity}$
4	$50 - d_5$	$Else\ \&\ L_a \neq L_0\ \&\ TTC_5 < t_{proximity}\ \&\ a_a > 0$
5	$-1.5 * d_5$	$Else\ \&\ L_a \neq L_0\ \&\ TTC_5 > t_{proximity}$
6	0.5	$Else\ \&\ L_a = L_{max}\ \&\ TTC_1 < t_{proximity}\ \&\ a_a < 0$
7	-0.5	$Else\ \&\ L_a = L_{max}\ \&\ TTC_1 < t_{proximity}\ \&\ a_a > 0$
8	-1	$Else\&v_a > SpeedLimit$
9	1	$Else\&a_a > 0t$
10	2	$Else\&v_a = SpeedLimit$
11	0	$Else$

To compare results, the agent trained with the time headway was tested and compared to the agent trained with constant distance headway as a control to verify whether the proximity parameter was impacted.

5.3 Deep Q-Learning Cooperative Automated Driving with Secondary Perceived Vehicles Input Features

The goal of the third and final experiment was to determine whether including information from SPVs during training of the driving algorithm has an impact on the performance of the driving agent, quantified by the collision rate and average speed. To include information from the SPV, the perception model (from the interpreter) and the DQN of the agent's driving algorithm were updated such that they would accept the SPV information. It is important to note that no additional updates were made; this includes the reward function, which still did not include SPV information.

5.3.1 Including Secondary Perceived Vehicles

In terms of the information that was included from the SPVs, it is important to note that it was the same information that was included from the PPVs, velocity and distance. Additionally, only forward SPVs were included in this experiment given that rear SPVs are not expected to impact the decision-making of the driving algorithm. To distinguish between PPV and SPV velocity and distance, a new subscript was added to the features' notations. Given v_{xy} or d_{xy} , x represents the perception level and y specifies the relative position of the target vehicle. x is set to 1 to classify PPVs and 2 to classify SPVs. Vehicles that are in the same lane as the AV are given a y index of 1 for vehicles in front of the AV and 2 for vehicles behind it. Vehicles that are in front and behind the AV in the lane left of the AV are given a y index of 3 and 4, respectively. Finally, vehicles in front of and behind the AV in the lane right of the AV are given a y index of 5 and 6, respectively. Figure 20 and Figure 21 illustrate the notations used for input information from target vehicles when the AV is in the right (driving) or left (overtaking) lane, respectively. Table 10 summarizes the complete list of features used in this experiment.

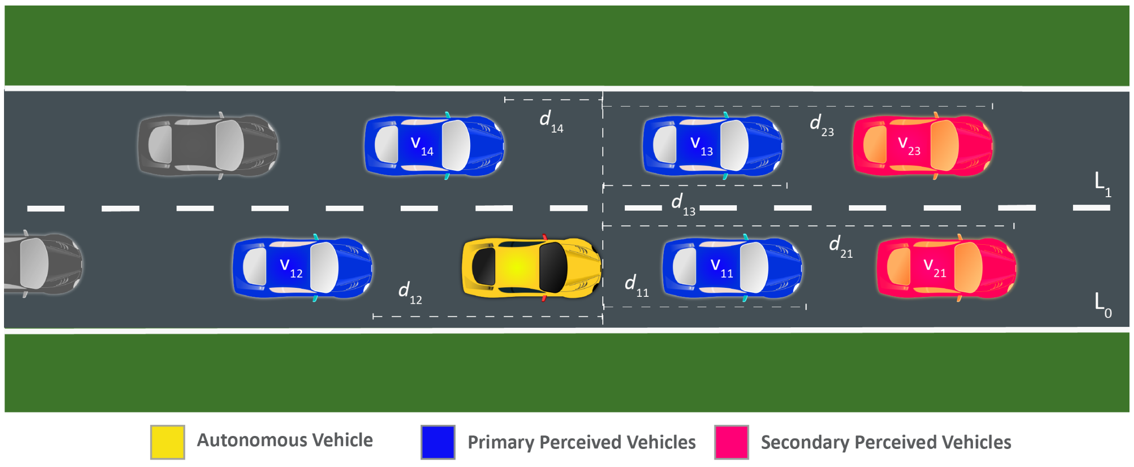


Figure 20: AV driving in the driving lane, with input feature notation for PPVs and SPVs.

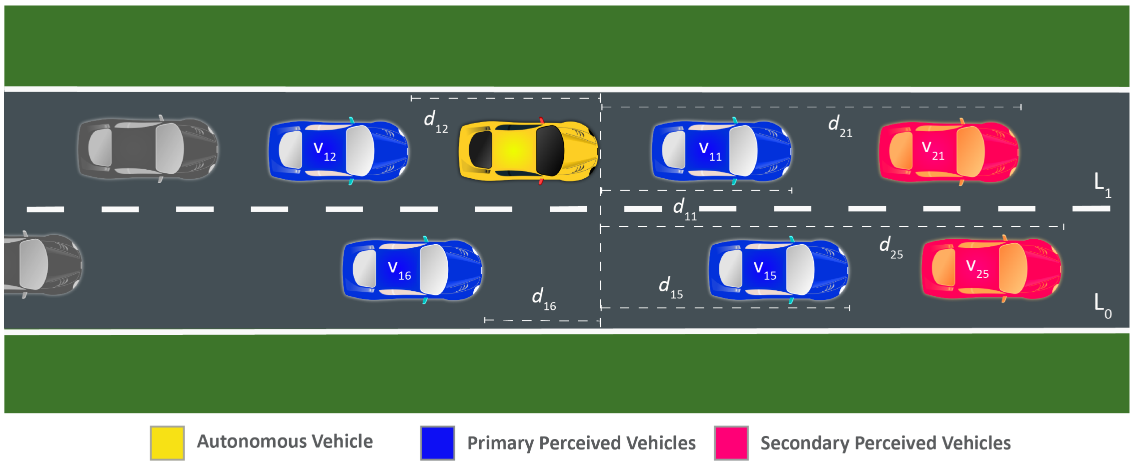


Figure 21: AV driving in the overtaking lane, with input feature notation for PPVs and SPVs.

The size of the input layer of the DQN in the agent was updated to receive the new additional

Table 10: Input features with SPVs included

Features	**Description**
v_a	Current velocity of the autonomous vehicle
v_{11}	Current velocity of the PPV in front of the AV in the same lane
v_{12}	Current velocity of the PPV behind the AV in the same lane
v_{13}	Current velocity of the PPV in front of the AV in the left lane
v_{14}	Current velocity of the PPV behind the AV in the left lane
v_{15}	Current velocity of the PPV in front of the AV in the rightlane
v_{16}	Current velocity of the PPV behind the AV in the right lane
v_{21}	Current velocity of the SPV in front of the AV in the same lane
v_{23}	Current velocity of the SPV in front of the AV in the left lane
v_{25}	Current velocity of the SPV in front of the AV in the rightlane
d_{11}	Current distance of the PPV in front of the AV in the same lane
d_{12}	Current distance of the PPV behind the AV in the same lane
d_{13}	Current distance of the PPV in front of the AV in the left lane
d_{14}	Current distance of the PPV behind the AV in the left lane
d_{15}	Current distance of the PPV in front of the AV in the rightlane
d_{16}	Current distance of the SPV behind the AV in the right lane
d_{21}	Current distance of the SPV in front of the AV in the same lane
d_{23}	Current distance of the SPV in front of the AV in the left lane
d_{25}	Current distance of the SPV in front of the AV in the rightlane
L	Current lane the AV is occupying (L=0 to maximum number of lanes)
v_{acc}	Current acceleration rate of AV

information. The perception model in the interpreter was updated to extract the new SPV information, similar to how information from PPVs was extracted, detailed in Section 5.1.3. No information from vehicles outside the 800 m range of the AV was included, as they would fall outside the simulating V2V range.

6 Results and Discussion on DQL Design and the Impact of SPVs

6.1 Deep Q-Learning for Cooperative Automated Driving Rerun

The purpose of this experiment was to replicate the results from the previous research in order to establish a baseline for novel research. To measure whether the model from previous research was successfully replicated, key metrics captured and compared included: 1) training collision rate profile, 2) collision rate, and 3) average speed. Notably, these metrics were selected as they were the only available results from the previous research. The training collision rate profile is a plot that illustrates the cumulative collision rate over the 7000 episodes during training. Figure 22 demonstrates the training collision rate profile of previous research (right) and current research (left), respectively. The green vertical dashed lines represent episodes where the epsilon value is at 0.9, 0.5, and 0.1. Recall from Section 4.4.2 that the epsilon, also called the exploration rate, represents the probability of whether the driving agent chooses to explore or exploit. As illustrated in Figure 22, the plot on the left demonstrates initial episodes having a high collision rate, as expected. This can be explained by the fact that epsilon is considerably high at the starting point resulting in the driving agent making a great amount of random decisions, ultimately yielding a high collision rate. Additionally, given that the DQN agent has little to no training experience initially, it will have random weights and biases, also leading to random decisions. The cumulative collision rate peaks roughly at about the 500th episode, and begins to decrease at an approximately linear rate. The decrease in collision rate during these periods can be attributed to the increase in the learning experience of the DQN agent and the decrease in epsilon value. The decrease in epsilon leads to an increase in exploitation and decrease in exploration, increasingly making decisions with the trained DQN agent. The probability for more exploitation than exploration occurs when epsilon falls below 0.5, which is after the 745th episode. At about the 2900th episode the cumulative collision percentage begins to plateau, coinciding with the point when epsilon reaches its lowest value of 0.1, after which epsilon remains static. The plateauing of the collision percentage can be attributed to two causes: the increased learning of the DQN agent, as well as the fact that new learning opportunities to improve performance are rare, especially with epsilon being low. The cumulative collision rate reached a value of roughly 30% by the end of the training at the 7000th episode.

As illustrated in Figure 22, the current research demonstrated similar behaviour, as can be seen on the plot on the right. It is important to note that the rise, fall and plateau of the collision rate in the current research occurred generally at the same points (episodes) as the previous research. This contributes to the assertion that the current experiment has a similar setup to the previous experiment. The current research also demonstrates that the cumulative collision rate reached at the end of training is similar to previous research at 30%. In comparing previous and current research, it is evident that the only difference that is observed in the training collision rate profiles is the peak collision. In the previous research, the collision rate peaked at roughly 70%, whereas in the current experiment, the collision rate peaked at roughly 65%. The difference in peak collision rate can be attributed to differences in initial weight and biases and different scenarios experienced by the agents.

Additional vital metrics to capture and compare performance of the previous and current research are the collision rate and average speed during testing. These two measurements encapsulate

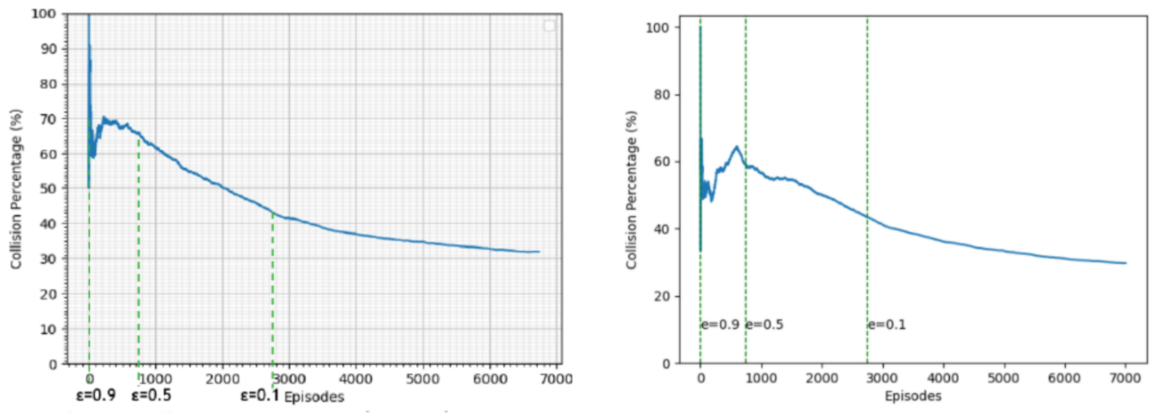


Figure 22: Cumulative collision rate during training of previous research (left) [8] and of current research (right).

the objective of the driving algorithm, which is to minimize collision rate and maximize the average speed within the given speed limit. Table 11 summarizes the collision rate and average speed during testing for the two trained DQN agents being compared. To compare results, the collision rate is represented as a range, specifically the 95% confidence interval (CI), which was calculated using the bootstrapping method from [54]. As can be seen from Table 11, the collision rate of the driving algorithm from the current research has a slightly wider range of 1.6%-5.6% compared to the driving algorithm of the previous research with a range of 1.5% to 4.8%. Additionally, 77.5% of the CI from the current research overlaps with the CI of the previous research. In terms of average speed, the two driving algorithms were very similar with the current experiment and previous experiment having an overall average speed of 13.4 m/s and 13.6 m/s, respectively.

Table 11: Comparing collision rate and average speed between Chandramohan et al. (2019) and current research.

	95% Conf. Interval of collision %	**Average speed**
Previous research scenario	[1.5% - 4.8%]	13.4
Current research scenario	[1.6% - 5.6%]	13.6

6.2 Deep Q-Learning for Cooperative Automated Driving Revise

As mentioned in Section 5.2, three areas that could improve the training and performance of the driving algorithm from the baseline design were: 1) stability 2) training efficiency and 3) incorporate speed into proximity parameter. To improve stability of the agent, a DDQN learning algorithm was implemented. To address training efficiency, PER was implemented. Last, to address speed not being considered in the proximity parameter, time-based proximity was implemented in the design. In this section, the performance of each integration is reported.

6.2.1 Addressing Stability via DDQN

As mentioned in Section 5.2, the neural network used to train the driving algorithm in Experiment 1 was a DQN, which revealed an instability in the agent. To address this issue, a DDQN was implemented and comparisons were made with the original DQN. Figure 23 illustrates the shaded plot of DQN (purple) and DDQN (orange) based on the results obtained from running 6 driving algorithms of each implementation (DQN and DDQN), trained with distinct random seeds. In Figure 23, the purple and orange lines in the plots represent the median cumulative collision rate over the testing sessions of 500 episodes for DQN and DDQN, respectively. Additionally, the shaded purple and orange areas represent the range of extreme values (10% and 90% quantiles with linear interpolation) over the testing sessions of 500 episodes for DQN and DDQN, respectively. As illustrated by Figure 23, DDQN agent produced a significantly reduced range of values by about 50% on average, when compared to the DQN agent over the testing period. The reduced range of the DDQN indicates that the collision percentage over the 6 trained algorithms was more precise compared to the 6 trained DQN algorithms, demonstrating that DDQN yields improved stability. Additionally, it is important to note that the DDQN median was lower than the DQN median, indicating that DDQN performed with a lower collision rate in general.

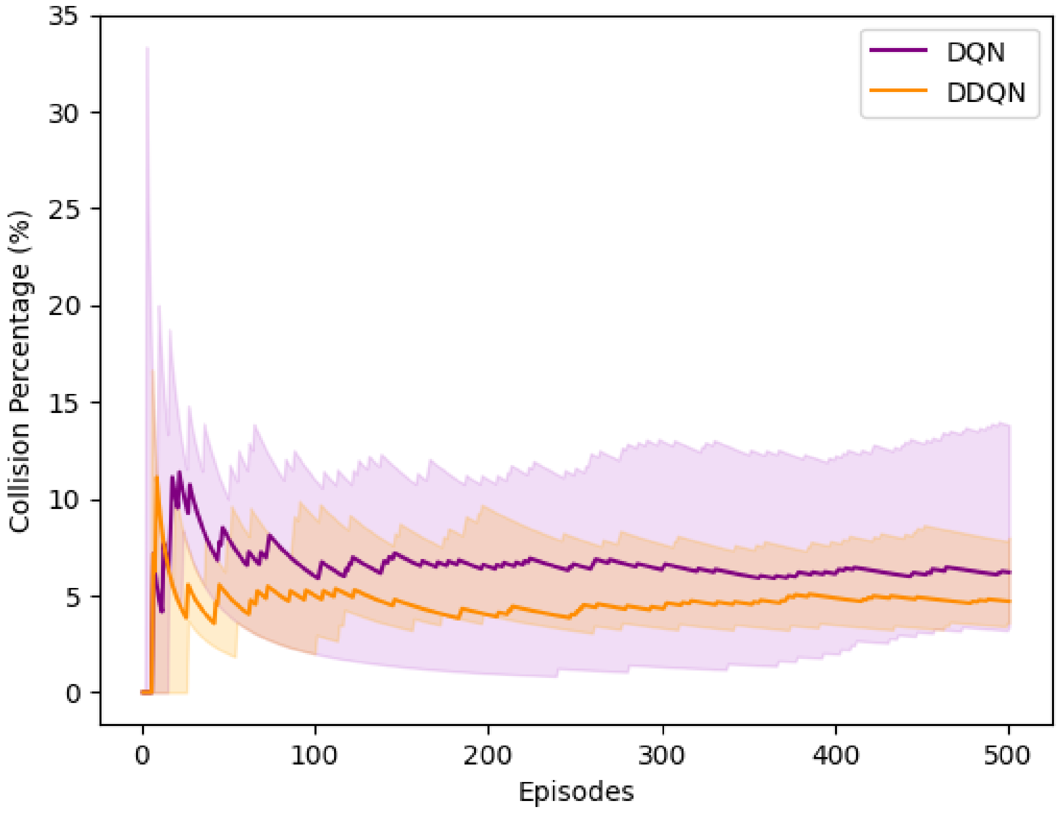


Figure 23: Shaded plot of cumulative collision rate of DQN (purple) and DDQN (orange) agents during training. The darker lines plot the median over 6 different agents and the shaded area represents the two extreme values (i.e., 10% and 90% quanttiles with linear interpolation).

While Figure 23 illustrates the stability of the driving algorithms based on collision rate, Figure 24 illustrates the stability in terms of average speed. The top graph in Figure 24 demonstrates that the average speeds across the 6 DQN driving agents were relatively stable, with only a small difference between the lower and upper values during testing. Comparatively, the bottom graph in Figure 24 demonstrates that the DDQN agents also showed stable average speeds. The median average speeds between DQN and DDQN agents were also similar, averaging at about 14 m/s. While the average speeds between the two are similar, one key difference is that the DQN agents had a greater amount of variation in average speed between episodes, compared to DDQN agents. The difference in these variations will be interpreted and elaborated upon in the Discussion section of this thesis.

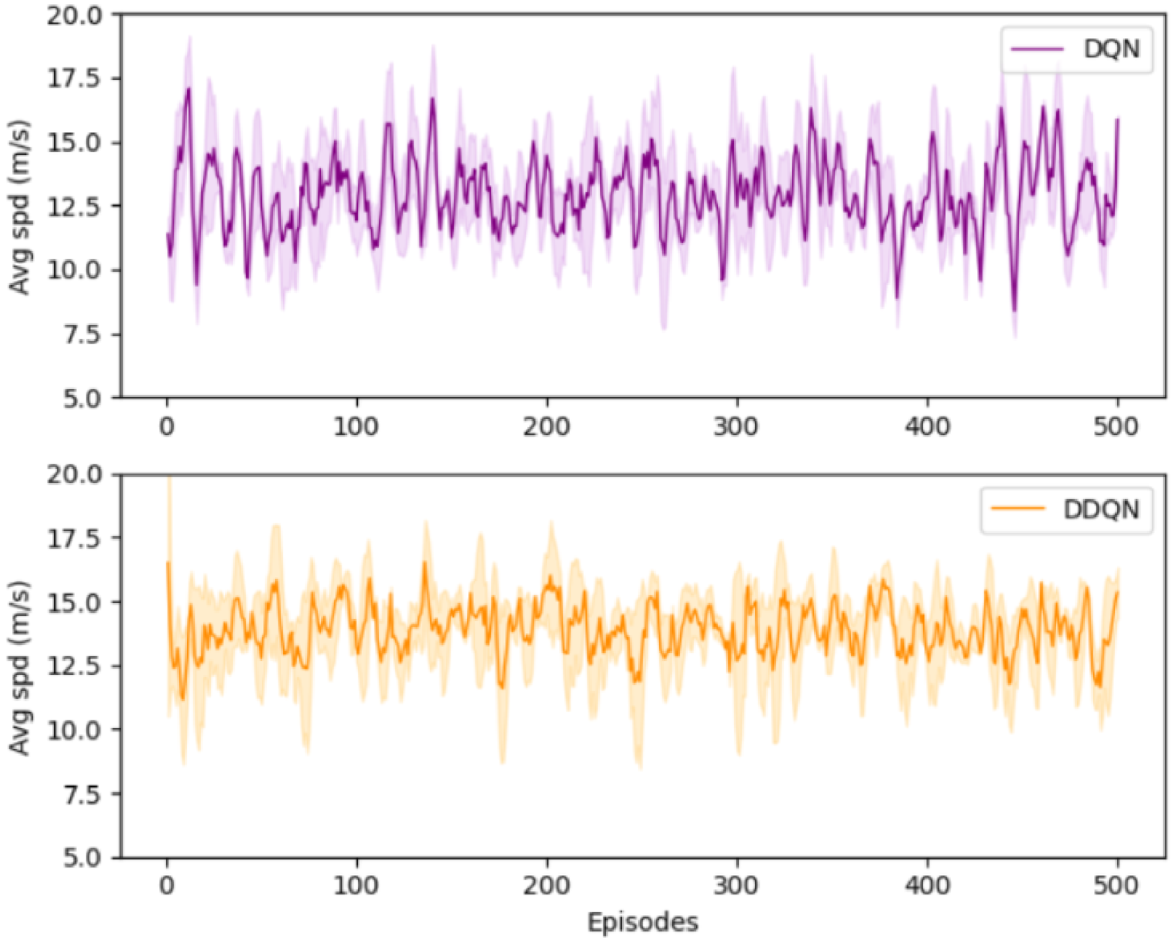


Figure 24: Shaded plot of average speed of DQN (purple) and DDQN (orange) agents during training. The darker lines shows the median over 6 different agents and the shaded area represents the two extreme values (i.e., 10% and 90% quantiles with linear interpolation).

In summary, the results for this experiment demonstrate that the DDQN agents produce better policies in terms of collision rate, as well as more precise driving algorithms, when compared to the DQN agents. In terms of average speed DDQN agents did not show any difference in precision, however, they produced lower variation in average speed, when compared to DQN agents.

6.2.2 Addressing Efficiency via PER

The goal of this experiment is to evaluate whether PER improves training efficiency. Recall from Section 5.2 that PER is a technique for choosing transitions (scenarios) from a memory queue when training the agent, by prioritizing the most important transitions. Transition importance is determined by TD errors, which is the delta between the predicted Q-value and the value signal that is computed (expected Q-value) for a given transition. In this experiment PER replaces UER, which is a technique of choosing transitions from a memory queue randomly with each transition having an equal probability of being selected. PER is expected to increase the efficiency of the training process, specifically making learning faster by increasing the frequency of replaying important transitions. Key metrics to measure and evaluate the impact of PER were the cumulative collision rate and the average speed throughout the training process (7000 episodes), in comparison to the UER. Figure 25 and Figure 26 illustrate the cumulative collision rate, and average speed throughout the training period for both UER and PER. Figure 25 demonstrates that the cumulative collision rate begins to decrease sooner for PER based agents compared to the UER agent, illustrating faster learning. It is also important to note that the PER based agent finished with a lower cumulative collision rate at approximately 20% than the UER at approximately 30%. The results presented here will be analyzed further in the Discussion section of this thesis.

Figure 26 illustrates the average speed during training for PER and UER based agents. Evidently,

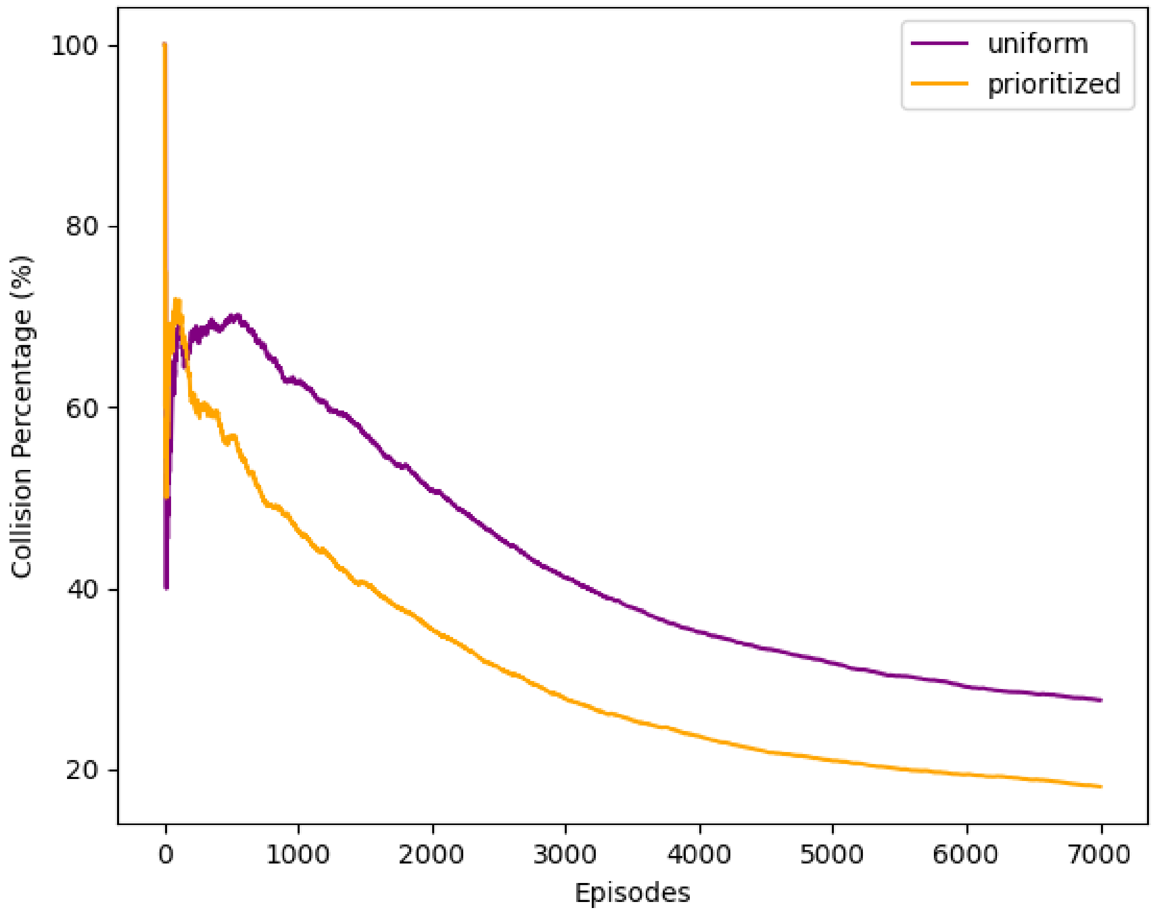


Figure 25: Cumulative collision rate per episode during training of agent using UER (purple) and agent using PER (orange).

the plot demonstrates that the learning rate for speed control between the two was not significantly different. Both agents seem to gradually increase average speed at the same rate and eventually plateau at roughly 13 m/s. While the average speed profile looks similar, it is important to note that the PER based agents demonstrate significantly less fluctuations when compared with the UER based agents; this pattern is interpreted and discussed in the Discussion section of this thesis.

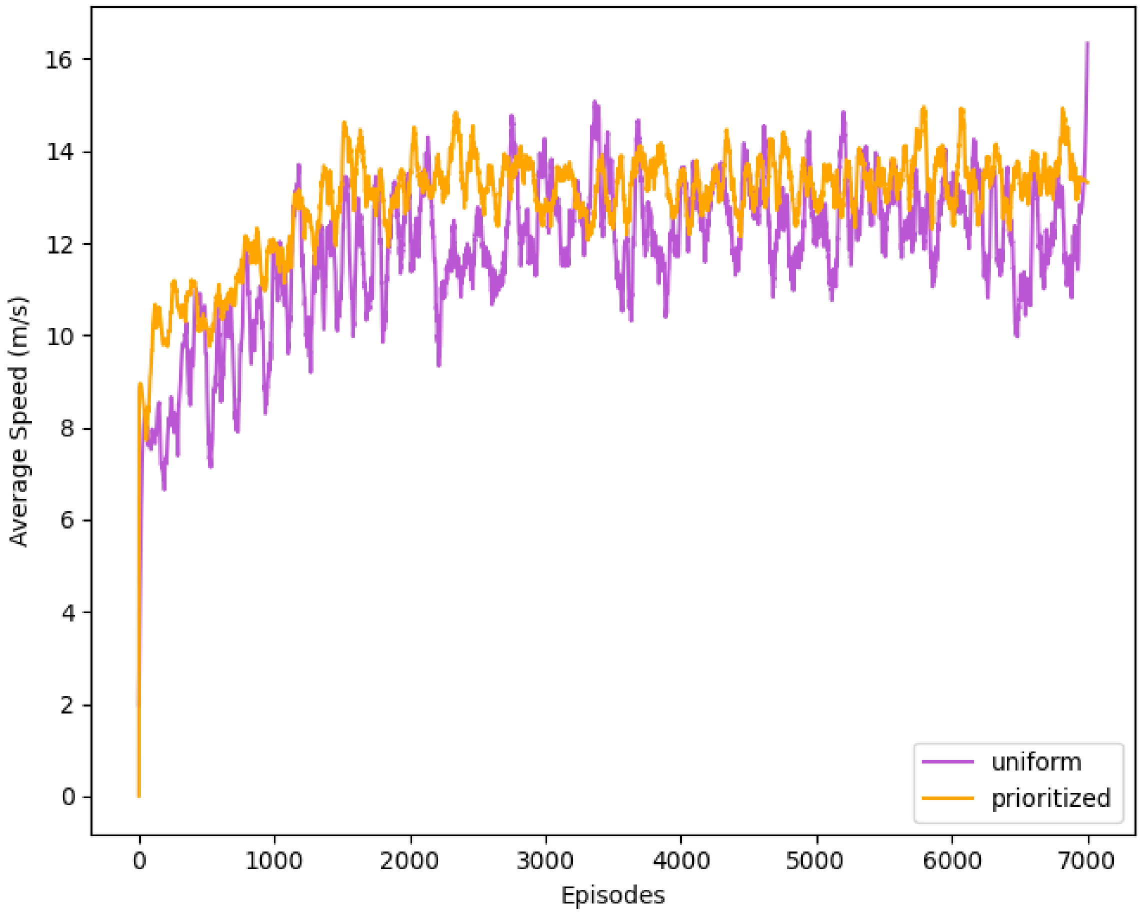


Figure 26: Average speed per episode during training of agent using UER (purple) and agent using PER (orange).

In summary, the PER technique resulted in faster learning for collision avoidance behaviour and produced less variation in average speeds, when compared to the UER based agents.

6.2.3 Addressing Proximity Distance via Time Headway

The goal of this experiment was to test whether replacing proximity distance with time-based proximity for the reward function helps to improve the set up and design of the baseline. Based on the proximity of the AV from a target vehicle, specific rewards are applied. In the previous research, the proximity is fixed as a distance of 160 m, meaning that any vehicle within 160 m proximity to the AV is considered unsafe and close proximity. The issue with using a distance value for the proximity in the reward function is that it does not account for the speed of the AV and target vehicle. This experiment uses a time headway that determines the proximity and accounts for the speed of the vehicles involved. Table 12 summarizes the collision rate and average speed of the testing over 500 episodes of the training algorithm that uses constant distance headway and the training algorithm that uses constant time headway. As indicated by similar results in the 95% confidence interval for both algorithms, it is evident that the collision rate was not affected by the use of time-based proximity. However, it is important to note that the overall average speed during testing was higher for the time-based proximity agent compared to the agent using distance-based proximity.

Table 12: Test Results for Constant Distance Headway and Time Headway

	95% Conf. Interval of collision %	**Average speed**
Distant based proximity	[1.3% - 5.4%]	13.6
Time based proximity	[1.1% - 5.5%]	14.8

6.3 Deep Q-Learning for Cooperative Automated Driving with SPVs

A central investigation of this research is the effect of including the SPV information as an input to train driving algorithms. This effect was quantified by measuring collision rate and average speed. For this experiment, the agent from Section 5.2 was used as the control, and the experimental agent included the SPVs as an input - the inclusion of the SPV was the sole modification between the control and experimental agents.

This thesis hypothesizes that including information from SPVs will improve the performance of the driving agent, indicated by a reduced collision rate and an increased average speed.

In this experiment, the control and experimental agents were run through an identical testing scenario of 500 episodes, and collision rate and average speed were measured.

Table 13: Test Results for Control DQN Agent and Experimental DQN Agent

	95% Conf. Interval of collision %	**Average speed**
Control	[1.1% - 5.5%]	14.8
Experimental	[1.2% - 5.3%]	14.6

Including SPV information to the agent did not improve the collision rate and average speed of the driving algorithm as can be seen from Table 13. However, this inclusion is expected to at least modify the speed control behaviour of the agent.

To investigate further, we evaluated whether the addition of the SPV impacts the speed control behaviour of the agent, illustrated in Figure 27. Figure 27 plots the average speed (top) and standard deviation (bottom) over the course of 500 episodes in the test scenario for the control agent (purple) and the experimental agent (orange). Comparatively, it can be noted that while the two agents had similar overall average speeds during testing, the experimental agent demonstrates a lower standard deviation indicating less variation in speed.

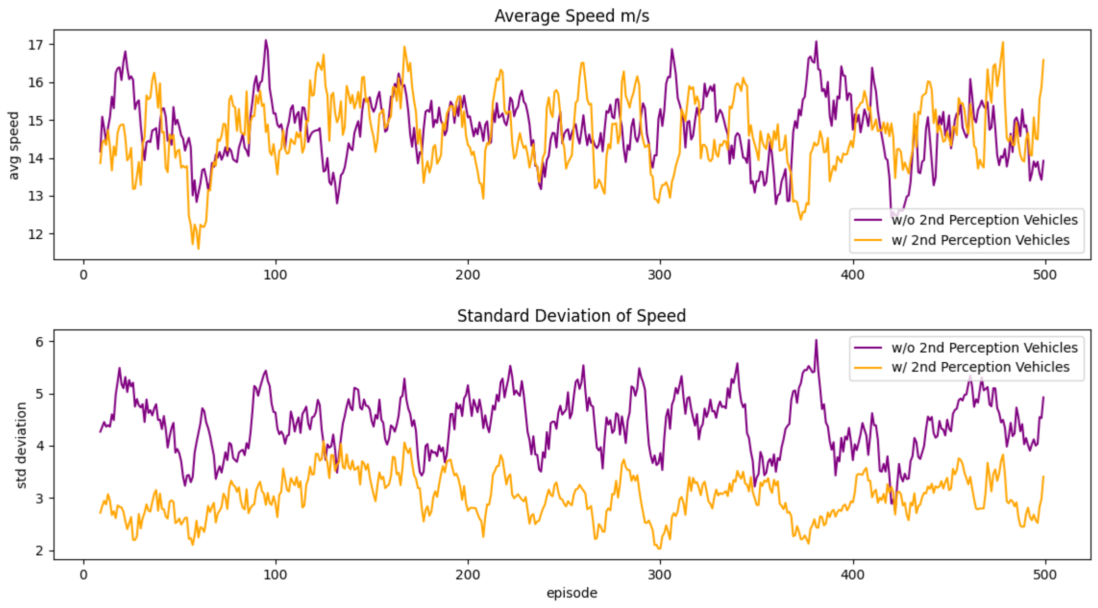


Figure 27: Average speed per episode (top) and speed standard deviation per episode during testing (bottom). Plots for agent that does not include SPV information (purple) and plot for agent that includes SPV information (orange).

The experimental agent's relatively lower speed variation in comparison to the control agent's is likely a result of lower overall acceleration. To confirm this assertion an analysis was done on the acceleration of the two agents in which the acceleration of the first 1000 time steps (10 episodes) during testing were recorded and illustrated in Figure 28. Figure 28 is a histogram that shows the frequency of acceleration values within 1 m/s^2 bins, for the control agents (without SPV) and the experimental agent (with SPV). As illustrated in Figure 28, the peak for both agents occurs within the 0-1 m/s^2 bin and the majority of acceleration values for both agents are considered low (between 1m/s^2 and -1m/s^2). Notably, the experimental agent has a greater number of instances with low acceleration of roughly 60% compared to control agent with roughly 50%. These findings support the earlier point that the experimental agent generally displays lower accelerations. When comparing the acceleration spread of the control versus the experimental agents, the ranges were found to be 4m/s^2 to 13m/s^2 and 4m/s^2 to -17m/s^2 respectively. These ranges also demonstrate the lower variation in velocity observed in the experimental agent. Last, to analyze these results further, it was found that the control agent not only has a wider acceleration spread, but also produces a higher frequency of extreme acceleration values (high acceleration and deceleration) compared to the experimental agent. These extreme acceleration values would contribute to higher fluctuation in speed, and help explain the greater standard deviation described in Figure 27.

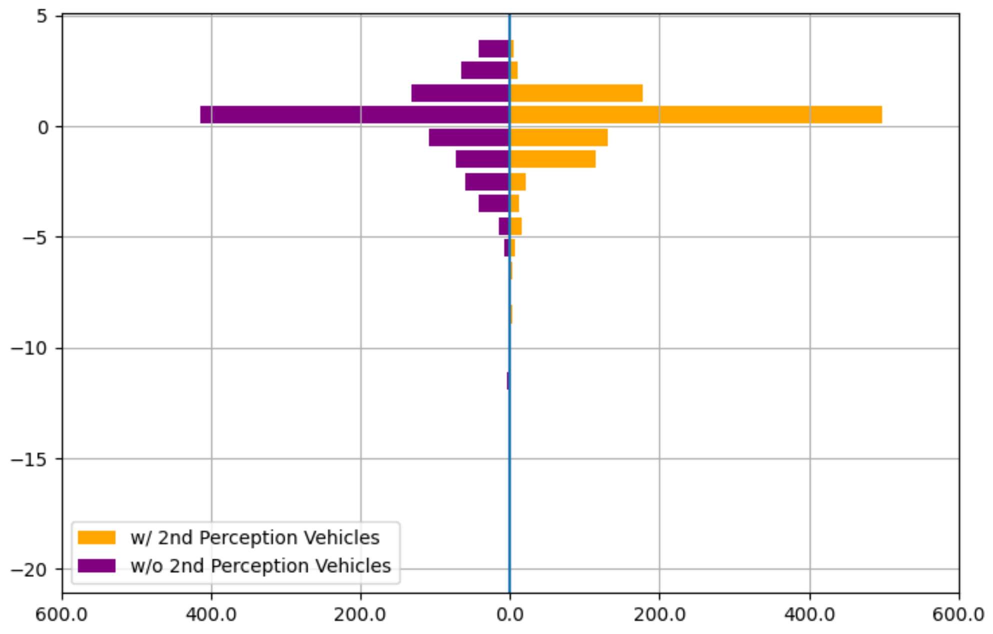


Figure 28: Histogram of acceleration for agent without SPV information (purple) and agent with SPV information (orange) over 1000 time steps.

In addition to overall acceleration, the deceleration behaviours of the two agents were further analyzed. Figure 29 illustrates a histogram of acceleration values below -4 m/s^2 for both the control and experimental agents. In general, high negative acceleration values are typically associated with deceleration for avoiding collisions when FCAS is activated. As can be seen from Figure 29, the control agent produced 3.9% instances of high deceleration while the experimental agent produced 3.2%, an 18% reduction. Additionally, not only does the experimental agent produce fewer instances of high deceleration, it in fact only produced a single instance of deceleration below -10m/s^2, in contrast to the control agent, which produced 10 instances above this point.

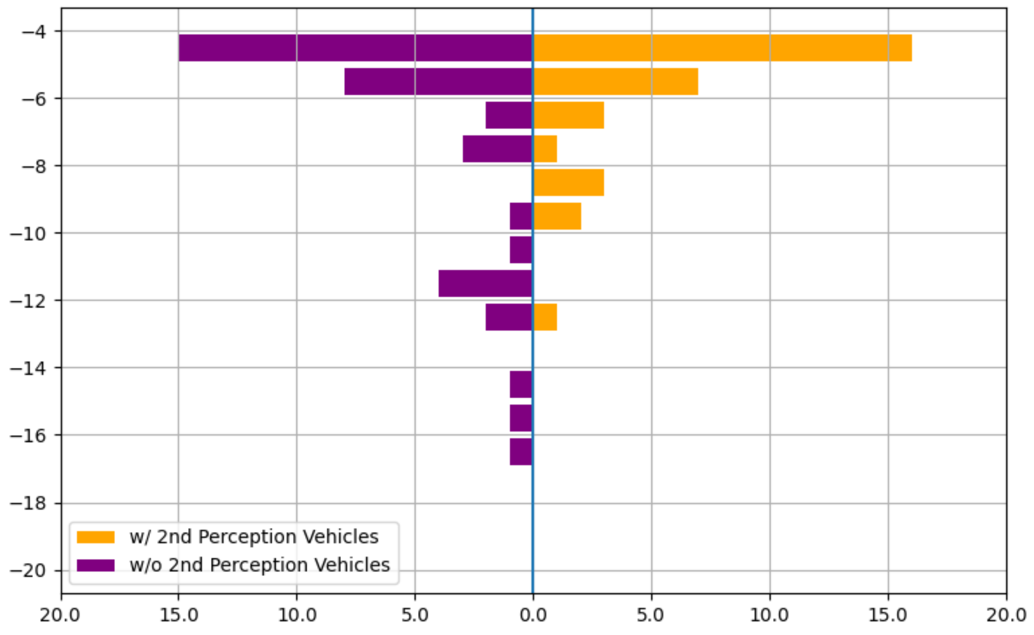


Figure 29: Histogram of high negative acceleration for agent without SPV information (purple) and agent with SPV information (orange) over 1000 time steps.

The improved performance of the experimental agent is proposed to be a result of its ability to better predict the flow of traffic. To assess this, both the control and the experimental agents were tested again with 500 episodes, this time with the variation in traffic speed being reduced by

80%, creating a more constant traffic flow. In situations with greater traffic flow consistency, the ability to predict traffic flow becomes less imperative. Figure 30 illustrates the average speed and standard deviation between the control and experimental agents in this experiment. Comparatively, the average speed between the two agents is similar, and interestingly, the standard deviation is also similar. Recall that the standard deviation was dissimilar between the two groups during fluctuating traffic flow, but this effect is lost during constant traffic flow scenarios, captured in Figure 27 earlier.

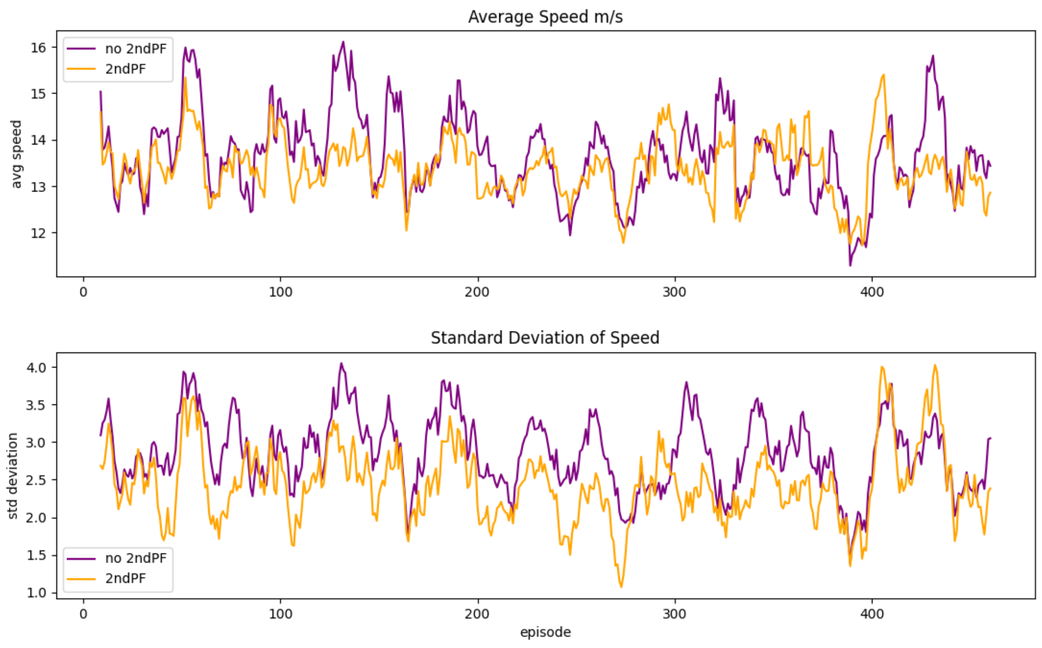


Figure 30: Average speed per episode during testing with constant traffic flow (top) and speed standard deviation per episode during testing with constant traffic flow (bottom). Plots for agent that does not include SPV information (purple) and plot for agent that includes SPV information (orange).

6.4 Discussion

This thesis aimed to examine the effect of including the SPV as an input when training an agent with RL for cooperative autonomous driving. To investigate this, we conducted and presented results for three experiments.

The goal of the first experiment was to reproduce the results from previous research [8], that designed a driving algorithm using DQL, in order to establish a baseline for new research. Key indicators that measured whether the previous model was successfully replicated included 1) training collision rate profile, 2) collision rate and 3) average speed. With regards to the training collision rate profile, results demonstrated considerable similarities between the previous research and our current research, with the only notable difference being the peak collision, which was 5% lower in the current experiment. This difference in peak collision rate can be attributed to differences in initial weight and biases and different scenarios experienced by the agents. When analyzing the collision rate and average speed between the previous research and current research, we found similar results. Given that these key indicators were replicated in the current research, it is concluded that the baseline was successfully established, and furthered the confidence and understanding of the research and design. It is important to note that while the collision rate was successfully replicated, in the context of highway driving, the rate itself is considered high and requires further improvement. While the results indicate further improvements are necessary in the designing of driving algorithms, they also support that RL is a prospective approach for cooperative autonomous driving. Upon having established a baseline design in the first experiment, the goal of the second experiment was to improve the stability, efficiency, and overall design of the driving algorithm. To improve the stability of the agent, a DDQN was implemented and compared with a DQN, with key performance metrics being the collision rate and average speed during testing. Results indicated that implementing a DDQN reduced the collision performance spread across 6 separately trained agents by about 50%, making the collision rate more precise. Additionally, when assessing the effect of using DDQN, it did not affect the stability of average speed, but it did produce less variation in average speed between the 500 episodes. The lower variation observed between episodes for DDQN is indicative of greater generalized learning speed control behaviors, rather than memorization. Taken together, these results demonstrate that DDQN can be used to successfully reduce overestimation bias in this application, resulting in more stable and reliable learning. One limitation of this interpretation is that the size of the dataset used in this experiment was small; only 6 separate driving algorithms were trained with distinct random seeds, which may call into question the reliability of the results.

To improve the efficiency of the driving algorithm, the PER technique was implemented, in which transitions were sampled from a memory queue during training. To evaluate the efficiency during training, metrics of cumulative collision rate and average speed per episode were monitored. Results demonstrated that the agent trained using PER was faster at learning collision avoidance behavior during training, illustrated by the faster decrease in cumulative collision rate. With faster learning, training periods can be reduced, which can help to reduce overall research and development time. In terms of the average speed, results did not indicate faster learning with speed control behavior, however, less variation in the average speed between episodes was observed. This can be interpreted as the agent being able to learn generalization because average speed between episodes (new situations) had less fluctuations. In comparison, UER based agents yielded greater fluctuations in average speed, which can be explained by the fact that UER based agents learn from all transitions, including those that may not matter, and result in memorized learning and potential overfitting. The ability of a model to generalize is central to the success of the model because non-generalized models – whether overfit or underfit – will make inaccurate predictions when given new data. Additionally, it was found that the final cumulative collision rate for the PER based agent was notably lower than that of the UER based agent. While at first glance this trend suggests that the PER based agent performs better than the UER, it is important to consider that the PER based agent learned faster, and therefore experienced less collisions. Therefore, it is reasonable to predict that if trained long enough the UER based agent will catch up in learning because learning becomes saturated. This is confirmed in the testing period of the two agents where the collision rates were similar at approximately 1.8%. Taken together, it is concluded that PER is an effective technique to yield faster and more generalized learning in comparison to UER, which increases the quality and performance of the driving agent.

One final improvement to the baseline model included the use of time headway instead of proximity distance for determining proximity in the reward function, when determining when a target is too close. This enabled the agent to take the relative speed of targets into account when making decisions. The aforementioned improvements to the baseline model were used as the control for the subsequent and final investigation of this thesis.

The central investigation of this research was whether including information from SPVs improves the performance of a driving agent. The current research demonstrated that including SPV information does not improve the overall performance (collision rate and average speed) of the driving algorithm, when compared with the control agent. It is important to note that due to an external FCAS being enabled during the experiments, only lane change collisions are possible in the experiment and therefore SPV information did not have any impact on lane change collisions specifically. Interestingly the experimental agent experienced less activation of FCAS, which suggests that if FCAS was disabled during the experiment, the experimental agent may have had an overall reduction in forward collisions; this is an important area that remains to be investigated. Additionally, it was found that the experimental agent also demonstrated less variation in speed, indicated by a lower standard deviation, lower acceleration spread, greater occurrence of low acceleration, and fewer occurrence of high deceleration, when compared to the control agent. One possible explanation for the experimental agent having a lower acceleration is that having access to information from additional vehicles ahead allows the experimental agent to better predict the flow of traffic, and better learn to modulate its speed accordingly. Being able to intelligently modulate speed in this manner results in 1) lower acceleration because the agent is not always trying to maximize speed, and 2) lower deceleration required to avoid collisions. Taken together, while the SPV inclusion does not have an impact on collision rate and average speed, its ability to better modulate speed suggests that this information can potentially improve fuel economy and drive quality.

Interestingly, although during fluctuating traffic flow, the experimental agent had lower acceleration compared to the control, it was found that when both the experimental agent and the control agent were tested in constant traffic flow scenarios, the effect of reduced acceleration was lost. More specifically, results indicate that with constant traffic flow, the experimental agent did not display any impact on the dynamic behaviors (speed and acceleration) of the driving algorithm. These findings are of particular significance in the real world, given that in situations of fluctuating traffic flow, the ability to predict traffic flow becomes imperative in modulating speed to avoid collisions and improve drive quality, whereas this ability is less important when traffic flow is constant. These findings convey that during situations of fluctuating traffic flow, such as in urban areas and stop and go traffic on highways, RL driving algorithms that use additional information from SPVs can reduce acceleration while still achieving the same average speed when SPV information is excluded. These findings are of particular significance given that lower acceleration means 1) better fuel economy and 2) better ride quality [55]. It is widely understood that acceleration and braking affects fuel economy, and more specifically high acceleration and braking negatively affects fuel economy when compared to cruise-type driving. According to Whitefoot et al. (2017), repeated acceleration and braking behavior can use up to 15% of fuel use in urban driving trips and high accelerations may result in a 10% increase in fuel use in light weight ICE vehicles. Lower acceleration and smoother velocity profiles have also been found to improve fuel economy in EVs [56]. For EVs, positive acceleration above 1.03 m/s^2 is considered aggressive driving and has a range of 10% - 40% more energy consumption compared to economic driving [56]. Additionally, aggressive driving is known to consume more energy even when regenerative braking is considered. In addition to improving fuel economy, lower acceleration improves ride quality, which is the ability to minimize the effects of drive irregularities to the vehicle passengers. A lower acceleration leads to more constant dynamic behavior of the vehicle, resulting in better ride quality. Taken together, the central investigation in this thesis reveals that while information from the SPV does not significantly improve collision rate and average speed, this information can assist in lowering acceleration in situations of fluctuating traffic flow, thereby improving fuel economy and ride quality. Reducing fuel economy and improving ride quality are important goals in the automotive industry, and this novel research helps pave the pathway for how the SPV can be utilized to improve these key performance indicators.

7 Conclusions and Future Work

The central goal of this research was to investigate the effect of including information from SPVs on the performance of a driving algorithm. Of the three experiments conducted, the first two aimed to reproduce and improve the DQN agent used to navigate a multi-lane highway from previous research. In the first experiment a DQN driving algorithm was designed with an external forward collision avoidance system to make decisions to change lanes, accelerate or decelerate using vehicle information from PPVs, with the goal of avoiding collisions and maximizing speed within the given speed limit. During training and testing, the driving algorithm showed comparable performance to previous research with a collision rate range of 1.6% to 5.6% and an average speed of 13.6 m/s. The AV performed as expected by overtaking slow moving vehicles and allowing faster moving vehicles to overtake, supporting the prospect that reinforcement learning with cooperative driving can control a vehicle effectively.

The goal of the second experiment was to improve the stability and efficiency of the DQN agent. Stability was improved using a DDQN to reduce overoptimistic bias resulting in a 50% increase in stability. More specifically, it was found that the performance range across six separately trained DDQN agents was half that of DQN agents. Improvement in stability optimizes the development of such agents, as less variation in performance enables better collaborative work across the field and more efficient validations. To improve efficiency, PER was implemented in the algorithm to sample experiences for training. Use of PER resulted in improved training efficiency as it helped the agent prioritize experiences that are deemed important during training in contrast to sampling uniformly. Using PER, the agent showed faster convergence to optimum performance during training and also more generalized learning when compared with UER. This improvement was significant because it allows for a shorter training period, which can significantly reduce development time of driving algorithms.

The final experiment in this research investigated the effect of including SPV information as input to the DQN driving algorithm. Including such information resulted in no difference in overall performance, namely collision rate and average speed when compared to agents lacking SPV information. Interestingly, more detailed analysis of speed variation revealed that agents trained with SPV information had less variation in speed and therefore less acceleration. This effect was pronounced during scenarios with high variation in traffic speed such as urban areas or stop and go highway traffic, but minimized with scenarios that had low variation in traffic speed such as open highway driving. These results support the assertion that including SPV information helps agents better predict traffic speed in scenarios with fluctuating traffic conditions. This ability to predict traffic speed results in improved ability to modulate speed in order to achieve the same overall speed, but with less acceleration. Results also demonstrated that the external FCAS was activated less often when SPV information was included, implying that SPV information can help reduce forward collision rates.

In conclusion, using RL in cooperative driving is a prospective approach and while information from the SPV does not significantly improve collision rate and average speed, this information can assist in lowering acceleration in situations of fluctuating traffic flow, thereby improving fuel economy and ride quality. Reducing fuel economy and improving ride quality are important goals in the automotive industry, and this novel research helps pave the pathway for how SPVs can be utilized to improve these key performance indicators.

7.1 Future Work

While the presented findings expand our understanding of the impact of SPV information in an autonomous cooperative driving simulation, there are key areas of improvement to note. First, it is recommended that future research investigates the effect of reducing the time step duration. The time step duration of one second used in this research is considered high latency for an autonomous driving scenario, which is problematic for mission critical activity. Reduced latency of time steps is not only more realistic in current autonomous driving research and development, it will also ensure the vehicle receives information about its surroundings with greater frequency and speed, which can potentially improve the performance of the driving algorithm. One caveat with reduced latency, however, is that it can significantly increase training time.

The current research utilized traffic modeling that included only two variations in speed, namely "FastCar" and "SlowCar". Another recommendation to improve upon this research is to use more sophisticated traffic modeling software to include vehicles with more variations in speed. It is predicted that this improvement will better simulate traffic flow that is reminiscent of real world conditions. An additional aspect of the speed parameters used in this research relates to the lack of flexibility in the speeding up and speeding down behaviour of the AV. It is therefore recommended that variation in the speeding up and speeding down decisions is implemented such that the AV can have more control and flexibility in its performance.

This research aimed to investigate the impact of including speed and distance information from both PPVs and SPVs to make driving decisions. As stated previously, SPV information helps the AV better predict the behaviour of PPVs. However, the current research is limited in that it does not collect information about acceleration. It is possible that collecting acceleration of PPVs can offset the need for information from SPVs altogether – an area that remains to be investigated. Removing SPV information and instead collecting PPV acceleration would mean that the AV is tasked with collecting and processing less data, which can improve performance and efficiency of the driving algorithm.

Finally, another limitation of this research is the use of a forward collision activating system that was external to the driving algorithm, which may be impacting performance of the driving algorithm. The FCAS system prevented rear-end collisions, and therefore only lane change collisions are possible in the experiments. Hence, it is recommended that a multi-objective network is used for steering and acceleration such that the need for an external collision management system is removed altogether. This is a key area of improvement that can ensure that the driving algorithm itself is capable of navigating a multi-lane highway by way of cooperative driving, including avoiding rear end collisions without help from an external collision avoidance system. This improvement would significantly improve the quality and reliability of the results.

References

[1] B. R. Kiran, I. Sobh, V. Talpaert, P. Mannion, A. A. A. Sallab, S. Yogamani, and P. Pérez, "Deep reinforcement learning for autonomous driving: A survey," 2021.

[2] N. Media, "Nhtsa releases 2019 crash fatality data," Dec 2020. [Online]. Available: https://www.nhtsa.gov/press-releases/nhtsa-releases-2019-crash-fatality-data

[3] Y. Ye, X. Zhang, and J. Sun, "Automated vehicle's behavior decision making using deep reinforcement learning and high-fidelity simulation environment," 2018.

[4] T. Reed, "Inrix global traffic scorecard," Jan 2019. [Online]. Available: https://trid.trb.org/view/1456836

[5] W. Schwarting, J. Alonso-Mora, and D. Rus, "Planning and decision-making for autonomous vehicles," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 1, no. 1, pp. 187–210, 2018. [Online]. Available: https://doi.org/10.1146/annurev-control-060117-105157

[6] M. N. Tahir, K. Maenpaa, and T. Sukuvaara, "Analysis of safecop features in v2i and v2v communication," in *2019 IEEE 89th Vehicular Technology Conference (VTC2019-Spring)*, April 2019, pp. 1–6.

[7] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: A Bradford Book, 2018.

[8] A. Chandramohan, M. Poel, B. Meijerink, and G. Heijenk, "Machine learning for cooperative driving in a multi-lane highway environment," in *2019 Wireless Days (WD)*, April 2019, pp. 1–4.

[9] R. Chopra and S. S. Roy, "End-to-end reinforcement learning for self-driving car," in *Advanced Computing and Intelligent Engineering*, B. Pati, C. R. Panigrahi, R. Buyya, and K.-C. Li, Eds. Singapore: Springer Singapore, 2020, pp. 53–61.

[10] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. A. Riedmiller, A. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529–533, 2015.

[11] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, ser. NIPS'12. Red Hook, NY, USA: Curran Associates Inc., 2012, p. 1097–1105.

[12] S. Ishikawa and S. Arai, "Cooperative learning of a driving strategy to suppress phantom traffic jams," in *2016 IEEE International Conference on Agents (ICA)*, Sep. 2016, pp. 90–93.

[13] X. Li, X. Xu, and L. Zuo, "Reinforcement learning based overtaking decision-making for highway autonomous driving," in *2015 Sixth International Conference on Intelligent Control and Information Processing (ICICIP)*, Nov 2015, pp. 336–342.

[14] S. Bacha, R. Saadi, M. Y. Ayad, A. Aboubou, and M. Bahri, "A review on vehicle modeling and control technics used for autonomous vehicle path following," in *2017 International Conference on Green Energy Conversion Systems (GECS)*, 2017, pp. 1–6.

[15] "Sae international releases updated visual chart for its "levels of driving automation" standard for self-driving vehicles," https://www.sae.org/news/press-room/2018/12/sae-international-releases-updated-visual-chart-for-its-

[16] "3 types of autonomous vehicle sensors in self-driving cars," https://www.itransition.com/blog/autonomous-vehicle-sensors.

[17] J. Hallam and C. Malcolm, "Behaviour: perception, action and intelligence — the view from situated robotics," *Philosophical Transactions of the Royal Society of London. Series A: Physical and Engineering Sciences*, vol. 349, pp. 29 – 42, 1994.

[18] N. Suganuma and Y. Hayashi, "Development of autonomous vehicle - overview of autonomous driving demonstration in its world congress 2013," in *2014 11th International Conference on Informatics in Control, Automation and Robotics (ICINCO)*, vol. 02, 2014, pp. 545–549.

[19] S. Campbell, N. O'Mahony, L. Krpalcova, D. Riordan, J. Walsh, A. Murphy, and C. Ryan, "Sensor technology in autonomous vehicles : A review," in *2018 29th Irish Signals and Systems Conference (ISSC)*, 2018, pp. 1–4.

[20] "The role of machine learning in autonomous vehicles," https://www.electronicdesign.com/markets/automotive/article/21147200/nxp-semiconductors-the-role-of-m

[21] "Radar and ultrasonic sensors strengthen adas object detection," https://www.electronicdesign.com/markets/automotive/article/21805470/radar-and-ultrasonic-sensors-streng

[22] L. Liu, S. Lu, R. Zhong, B. Wu, Y. Yao, Q. Zhang, and W. Shi, "Computing systems for autonomous driving: State of the art and challenges," *IEEE Internet of Things Journal*, vol. 8, no. 8, pp. 6469–6486, 2021.

[23] S. Lee, J. Kim, J. S. Yoon, S. Shin, O. Bailo, N. Kim, T.-H. Lee, H. S. Hong, S.-H. Han, and I. S. Kweon, "Vpgnet: Vanishing point guided network for lane and road marking detection and recognition," in *2017 IEEE International Conference on Computer Vision (ICCV)*, 2017, pp. 1965–1973.

[24] A. P. Dhongade and M. Khandekar, "Gps and imu integration on an autonomous vehicle using kalman filter (labview tool)," in *2019 International Conference on Intelligent Computing and Control Systems (ICCS)*, May 2019, pp. 1122–1125.

[25] P. Daruthep and N. Sutthisangiam, "Development of automated processing for high-definition mapping system," in *2020 22nd International Conference on Advanced Communication Technology (ICACT)*, Feb 2020, pp. 507–510.

[26] Y. Wang, C. Wang, W. Zhao, and C. Xu, "Decision-making and planning method for autonomous vehicles based on motivation and risk assessment," *IEEE Transactions on Vehicular Technology*, vol. 70, no. 1, pp. 107–120, Jan 2021.

[27] L. Ma, J. Yang, and M. Zhang, "A two-level path planning method for on-road autonomous driving," in *2012 Second International Conference on Intelligent System Design and Engineering Application*, Jan 2012, pp. 661–664.

[28] X. Zhang, S. Pan, and Q. Miao, “Adaptive beamforming-based gigabit message dissemination for highway vanets,” *IEEE Transactions on Intelligent Transportation Systems*, pp. 1–14, 2021.

[29] S. Hegde, L. Shi, N. J. Hernández Marcano, R. Shrivastava, O. Blume, and R. H. Jacobsen, “Sidelink group resource scheduling for platoons in cellular vehicle-to-vehicle communications,” in *2021 IEEE 93rd Vehicular Technology Conference (VTC2021-Spring)*, April 2021, pp. 1–5.

[30] E. Kural, T. Hacibekir, and B. A. Güvenç, “State of the art of adaptive cruise control and stop and go systems,” *CoRR*, vol. abs/2012.12438, 2020. [Online]. Available: https://arxiv.org/abs/2012.12438

[31] R. Utriainen, M. Pöllänen, and H. Liimatainen, “The safety potential of lane keeping assistance and possible actions to improve the potential,” *IEEE Transactions on Intelligent Vehicles*, vol. 5, no. 4, pp. 556–564, Dec 2020.

[32] O. Amimi, A. Mansouri, S. Dose Bennani, and Y. Ruichek, “Stereo vision based advanced driver assistance system,” in *2017 International Conference on Wireless Technologies, Embedded and Intelligent Systems (WITS)*, April 2017, pp. 1–5.

[33] L. Xiao and F. Gao, “A comprehensive review of the development of adaptive cruise control systems,” *Vehicle System Dynamics*, vol. 48, no. 10, pp. 1167–1192, 2010. [Online]. Available: https://doi.org/10.1080/00423110903365910

[34] C. Wu, Z. Xu, Y. Liu, C. Fu, K. Li, and M. Hu, “Spacing policies for adaptive cruise control: A survey,” *IEEE Access*, vol. 8, pp. 50 149–50 162, 2020.

[35] Z. Wang, G. Wu, and M. J. Barth, "A review on cooperative adaptive cruise control (cacc) systems: Architectures, controls, and applications," in *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, Nov 2018, pp. 2884–2891.

[36] K. Cumali and E. Armagan, "Steering control of a vehicle equipped with automated lane centering system," in *2019 11th International Conference on Electrical and Electronics Engineering (ELECO)*, Nov 2019, pp. 820–824.

[37] G. Lei, W. Jianqiang, and L. Keqiang, "Lane keeping system based on thasv-ii platform," in *2006 IEEE International Conference on Vehicular Electronics and Safety*, Dec 2006, pp. 305–308.

[38] J. F. B. Araújo, H. M. Pedrosa, M. C. B. P. Rodrigues, and P. G. Barbosa, "Real-time "hardware-in-the-loop" simulation of components of an electric vehicle powertrain: Modeling and implementation," in *2016 12th IEEE International Conference on Industry Applications (INDUSCON)*, Nov 2016, pp. 1–7.

[39] L. Traub, V. Butakov, and R. Simpson, "Parameter identification for a multi-body vehicle model," in *2016 IEEE Intelligent Vehicles Symposium (IV)*, June 2016, pp. 521–526.

[40] R. Hamilton, H. Seager, K. P. Divakarla, A. Emadi, and S. Razavi, "Modeling and simulation of an autonomous-capable electrified vehicle: A review," in *2018 IEEE Electrical Power and Energy Conference (EPEC)*, Oct 2018, pp. 1–7.

[41] M. Olofsson and J. Pettersson, "Parameterization and validation of road and driver behavior models for carmaker simulations and transmission hil-rig," Master's thesis, 2015.

[42] W. Xu, W. ZhaYao, H. Zhao, and H. Zha, "A vehicle model for micro-traffic simulation in dynamic urban scenarios," in *2011 IEEE International Conference on Robotics and Automation*, May 2011, pp. 2267–2274.

[43] D. Gruyer, S. Pechberti, and S. Glaser, "Development of full speed range acc with sivic, a virtual platform for adas prototyping, test and evaluation," in *2013 IEEE Intelligent Vehicles Symposium (IV)*, June 2013, pp. 100–105.

[44] T.-C. Lin, S. Ji, C. E. Dickerson, and D. Battersby, "Coordinated control architecture for motion management in adas systems," *IEEE/CAA Journal of Automatica Sinica*, vol. 5, no. 2, pp. 432–444, Mar 2018.

[45] K. L. Masita, A. N. Hasan, and T. Shongwe, "Deep learning in object detection: a review," in *2020 International Conference on Artificial Intelligence, Big Data, Computing and Data Communication Systems (icABCD)*, Aug 2020, pp. 1–11.

[46] L. Cardamone, D. Loiacono, and P. L. Lanzi, "Evolving competitive car controllers for racing games with neuroevolution," 01 2009, pp. 1179–1186.

[47] K.-J. Kim, J.-H. Seo, J.-G. Park, and J. C. Na, "Generalization of torcs car racing controllers with artificial neural networks and linear regression analysis," *Neurocomputing*, vol. 88, pp. 87–99, 2012, intelligent and Autonomous Systems. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0925231212000094

[48] M. S and P. M, "An analysis of q-learning algorithms with strategies of reward function," *International Journal on Computer Science and Engineering*, vol. 3, 02 2011.

[49] "Q-learning," Jul 2021. [Online]. Available: https://en.wikipedia.org/wiki/Q-learning

[50] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural Networks*, vol. 61, pp. 85–117, 2015. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0893608014002135

[51] "Sumo at a glance - sumo documentation," https://sumo.dlr.de/docs/SUMO_at_a_Glance.html.

[52] OpenAI, "A toolkit for developing and comparing reinforcement learning algorithms." [Online]. Available: https://gym.openai.com/docs/

[53] S. Thrun and A. Schwartz, "Issues in using function approximation for reinforcement learning," in *Proceedings of the 1993 Connectionist Models Summer School*, D. T. J. E. M. Mozer, P. Smolensky and A. Weigend, Eds. Erlbaum Associates, June 1993.

[54] C. Ismay and A. Y. K. F. by Kelly S. McConville, "Statistical inference via data science," Jul 2021. [Online]. Available: https://moderndive.com/8-confidence-intervals.html

[55] K. S. Whitefoot, M. L. Fowlie, and S. J. Skerlos, "Compliance by design: Influence of acceleration trade-offs on co2 emissions and costs of fuel economy and greenhouse gas regulations," *Environmental Science & Technology*, vol. 51, no. 18, pp. 10 307–10 315, 2017, pMID: 28825797. [Online]. Available: https://doi.org/10.1021/acs.est.7b03743

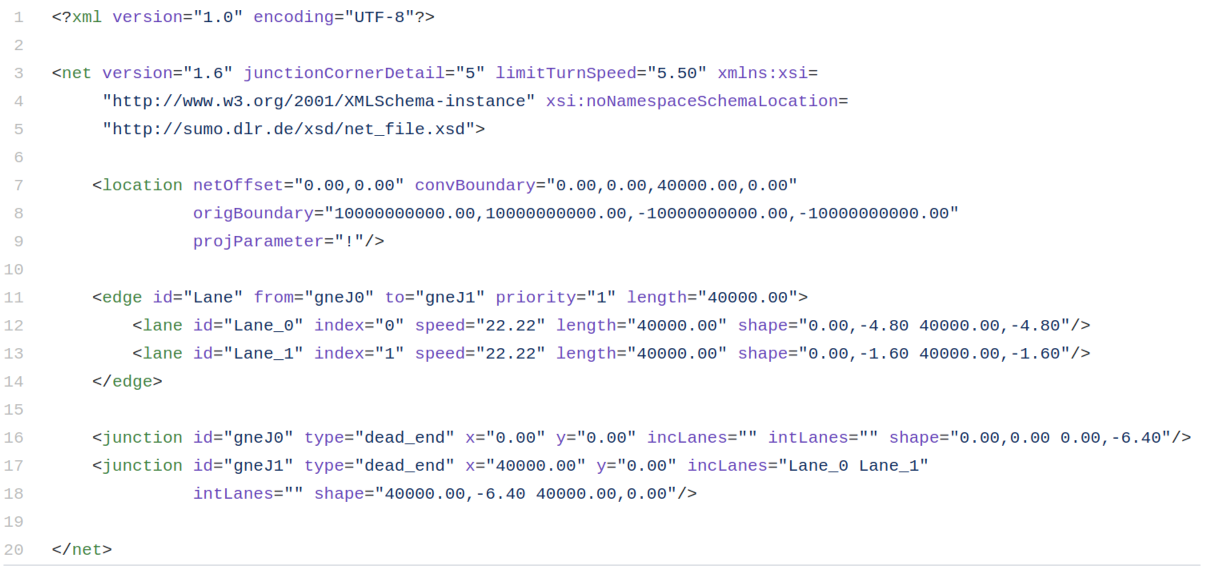
[56] F. Badin, F. Le Berr, H. Briki, J.-C. Dabadie, M. Petit, S. Magand, and E. Condemine, "Evaluation of evs energy consumption influencing factors, driving conditions, auxiliaries use, driver's aggressiveness," in *2013 World Electric Vehicle Symposium and Exhibition (EVS27)*, Nov 2013, pp. 1–12.

[57] [Online]. Available: https://sumo.dlr.de/docs/TraCI/Vehicle_Value_Retrieval.html

[58] K. Team, "Keras documentation: The model class." [Online]. Available: https://keras.io/api/models/model/

Appendix A

SUMO Configuration Files



```xml
<?xml version="1.0" encoding="UTF-8"?>

<net version="1.6" junctionCornerDetail="5" limitTurnSpeed="5.50" xmlns:xsi=
     "http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation=
     "http://sumo.dlr.de/xsd/net_file.xsd">

    <location netOffset="0.00,0.00" convBoundary="0.00,0.00,40000.00,0.00"
              origBoundary="10000000000.00,10000000000.00,-10000000000.00,-10000000000.00"
              projParameter="!"/>

    <edge id="Lane" from="gneJ0" to="gneJ1" priority="1" length="40000.00">
        <lane id="Lane_0" index="0" speed="22.22" length="40000.00" shape="0.00,-4.80 40000.00,-4.80"/>
        <lane id="Lane_1" index="1" speed="22.22" length="40000.00" shape="0.00,-1.60 40000.00,-1.60"/>
    </edge>

    <junction id="gneJ0" type="dead_end" x="0.00" y="0.00" incLanes="" intLanes="" shape="0.00,0.00 0.00,-6.40"/>
    <junction id="gneJ1" type="dead_end" x="40000.00" y="0.00" incLanes="Lane_0 Lane_1"
              intLanes="" shape="40000.00,-6.40 40000.00,0.00"/>

</net>
```

Figure 31: SUMO road network configuration file.

```xml
<?xml version="1.0" encoding="UTF-8"?>

<routes xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:noNamespaceSchemaLocation="http://sumo.dlr.de/xsd/routes_file.xsd">
    <vType id="Auto" length="3.00" minGap="3.00" maxSpeed="55.55" color="blue"
           accel="30" decel="30" tau="1" sigma="0" speedFactor="1.2"/>
    <vType id="FastCar" length="3.00" minGap="3.00" color="magenta"
           lcKeepRight="100" accel="30" decel="30" tau="3" sigma="0.5"/>
    <vType id="SlowCar" length="3.00" minGap="3.00" maxSpeed="11.10"
           color="red" accel="30" decel="30" tau="3" sigma="0.5"/>
    <route edges="Lane" color="yellow" id="straight"/>
    <flow id="FastCar" type="FastCar" begin="0.00" route="straight"
          end="200.00" probability="0.01"/>
    <flow id="SlowCar" type="SlowCar" begin="0.00" color="red"
          route="straight" end="200.00" probability="0.1"/>
    <vehicle id="Auto" type="Auto" depart="60.00" color="blue"
             route="straight" departSpeed="11.1"/>
</routes>
```

Figure 32: SUMO vehicle route configuration file

```xml
<?xml version="1.0" encoding="UTF-8"?>
<configuration xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
               xsi:noNamespaceSchemaLocation="http://sumo.dlr.de/xsd/sumoConfiguration.xsd">
    <input>
        <net-file value="2Lane_StraightRoad.net.xml"/>
        <route-files value="StraightRoad.rou.xml"/>
    </input>
    <time>
        <begin value="0"/>
        <step-length value="1.0"/>
    </time>
    <processing>
        <collision.action value="remove"/>
        <collision.mingap-factor value="0"/>
        <lanechange.overtake-right value="false"/>
    </processing>
    <gui_only>
        <quit-on-end value="true"/>
        <start value="true"/>
        <scale value="1.0"/>
    </gui_only>
    <random_numberType>
        <random value="true"/>
    </random_numberType>
</configuration>
```

Figure 33: SUMO simulation configuration file.

Appendix B

External APIs used

Details about the APIs from TraCI and Keras used in this research is givien in this chapet

APIs from TraCI

Table B below shows the APIs from TraCI used in this research for controlling the simulation environment and for accessing the vehicle information form the simulation and to give out the actions to the autonomous vehicle. The entire list of all the APs available in TraCI can be found in [57].

Table 14: APIs from TraCI used in this research.

S.No	Name of the Function	Description
1	traci.close()	Close interface with SUMO
2	traci.start(sumoCommand)	Start SUMO and load the SUMO configuration.
3	traci.load(sumoCommand)	Load SUMO configuration.
4	traci.simulationStep()	Run one time step in the simulation.

Continued on next page

Table 14 – *Continued from previous page*

S.No	Name of the Function	Description
5	traci.vehicle.getIDList()	Returns the ids of all the vehicles currently active in the simulation.
6	traci.simulation.getCurrentTime()	Returns the current simulation time in ms.
7	traci.vehicle.subscribe(vehID, parameters to subscribe)	The parameters subscribed by this command for the vehicle id is updated after each simulation step and can be got with the next function. The parameters subscribed are: -VehSpeed (m/s), VehPosition (x,y), VehLaneIndex, VehTravelDistance (m).
8	traci.vehicle.getSubscriptionResults()	Returns the latest values of the subscribed parameters of all the subscribed vehicles.
9	traci.vehicle.getSpeedMode()	Used to find the car following model, returns a 5 bit number.

Continued on next page

Table 14 – *Continued from previous page*

S.No	**Name of the Function**	**Description**
10	traci.vehicle.setSpeedMode(VehID, modeValue)	To set the car following model for the VehID. For disabling the automatic collision avoidance system, the 3^{rd} and 4^{th} bit of modeValue should be 0.
11	traci.vehicle.setLaneChangeMode(vehID, modeValue)	This is used for disabling automatic lane change of the autonomous vehicle by the simulator as this action needs to be done by the driving algorithm. For this the modeValue needs to be 0.
12	traci.vehicle.changeLane(VehID, laneIndex, time)	Moves the vehicle with id vehID to the lane given by laneIndex for the time provided. The time is given to be very long so that the vehicle remains in that lane until another action changes it to a different lane.
13	traci.vehicletype.getMaxSpeed(vehID)	Gets the maximum speed the vehicle can travel at.

Continued on next page

Table 14 – *Continued from previous page*

S.No	Name of the Function	Description
14	traci.vehicle.slowDown(VehID, targetSpeed, time)	Accelerates or decelerates the vehicle to the target speed(m/s) in the time(ms) provided.
15	traci.lane.getMaxSpeed(laneID)	Returns the speed limit of the lane.

APIs from Keras

Table 15 below gives the functions from the Keras library used in this research for the driving agent. The list of all APIs available in Keras in given in [58].

Table 15: APIs from Keras used in this research.

S.No	Name of the Function	Description
1	model.Add(LayerProperties)	This is used to add a layer to the neural network. The LayerProperties consistes of the number of nodes in teh layer, its activation method and the initial weights for the nodes.
2	model.compile(lossFunction, optimizer)	Configures the model for training.
3	model.predict(stateSet)	Generate the output Q values for each of the state from the current State.
4	model.fit(stateSet, TargetValues, epochs)	Trains the neural network for the given number of epochs.
5	model.load_weights(filePath)	Loads the weights of the nodes in the neural network from an existing file so that the agent need not be retrained during each simulation.
6	model.save_weights(filePath)	Save the weights of the nodes in the neural network to filePath.