C2SMART
CONNECTED CITIES WITH SMART TRANSPORTATION
A USDOT University Transportation Center

New York University

Rutgers University

University of Washington

The University of Texas at El Paso

City College of New York

Vehicle-Traffic Control with Limited-Capacity Connected/Automated Vehicles

December 2020

Vehicle-Traffic Control with Limited-Capacity Connected/Automated Vehicles

Principal Investigator: Jeff Ban
University of Washington

Xuegang (Jeff) Ban
University of Washington
0000-0003-3605-971X

Qiangqiang Guo
University of Washington
0000-0002-6461-5886

Ohay Angah
University of Washington

Zhijun Liu
University of Washington
0000-0002-0071-1406

C2SMART Center is a USDOT Tier 1 University Transportation Center taking on some of today's most pressing urban mobility challenges. Some of the areas C2SMART focuses on include:

Urban Mobility and Connected Citizens

Urban Analytics for Smart Cities

Resilient, Smart, & Secure Infrastructure

Disruptive Technologies and their impacts on transportation systems. Our aim is to develop innovative solutions to accelerate technology transfer from the research phase to the real world.

Unconventional Big Data Applications from field tests and non-traditional sensing technologies for decision-makers to address a wide range of urban mobility problems with the best information available.

Impactful Engagement overcoming institutional barriers to innovation to hear and meet the needs of city and state stakeholders, including School of Engineering, **C2SMART** is a consortium of leading research universities, including Rutgers University, University of Washington, the University of Texas at El Paso, and The City College of NY.

Forward-thinking Training and Development dedicated to training the workforce of tomorrow to deal with new mobility problems in ways that are not covered in existing transportation curricula.

Visit c2smart.engineering.nyu.edu to learn more

Disclaimer

The contents of this report reflect the views of the authors, who are responsible for the facts and the accuracy of the information presented herein. This document is disseminated in the interest of information exchange. The report is funded, partially or entirely, by a grant from the U.S. Department of Transportation's University Transportation Centers Program. However, the U.S. Government assumes no liability for the contents or use thereof.

Acknowledgements

The project team would like to thank the financial and administrative support by the C2SMART UTC. The team also appreciates the support and collaboration with the EcoCar3 team at the University of Washington.

Executive Summary

This project studies methods to control both vehicle and traffic under limited penetration of low-level connected and autonomous vehicles (LCAVs). The investigation includes three major parts: i) the Eco-Driving algorithm for a single CAV with low-level automation; ii) the vehicle in the loop (VIL) simulation platform; and iii) the integrated vehicle/traffic control algorithm tested under the VIL.

A hybrid deep Q-learning and policy gradient (HDQPG) based Eco-Driving algorithm is first developed for a single CAV driving along signalized corridors with low-level automation to learn and control both the longitudinal operations and the lane-changing decisions of the LCAV. We design specific states, actions, and reward functions for the decisions and develop a "checking-feedback-learning" (CFL) framework to ensure driving safety and decision consistency by encouraging the RL algorithms to learn safe driving behaviors and consistent decisions. Numerical results show that HDQPG outperforms existing RL-based Eco-Driving methods and could learn basic fuel-saving strategies that have been proven to be fuel-efficient by model-based methods in the literature, and could also deal with unusual driving conditions.

Second, a vehicle in the loop (VIL) platform is developed to reduce the costs of testing algorithms in real-world. Four key components of VIL are developed or integrated: the microscopic traffic simulation entity (SUMO is used in this project) to simulate the traffic flows and signals, the vehicle simulation/ visualization entity (Unity is used in this project) to simulate vehicle-level dynamics and provide detailed vehicle information and onboard sensor data, a real-world fully controlled autonomous car (an AWS DeepRacer car is used in this project) to receive and conduct the commands generated by the tested algorithms, and the control center that coordinates/communicates with the other three components.

Third, a dynamic HCM method is designed to control signals under the environment of LCAVs. The method first estimates the CAV penetration and calculates the total traffic volume based on the estimated CAV penetration and the CAV volume. Given the volume data, the method calculates the signal phases/timings based on the HCM method, which can be used for the next forward time horizon. We combine this dynamic HCM method with the Eco-Driving algorithm mentioned as a vehicle-traffic integrated control method, and test it on the VIL platform. The results show that the integrated method could reduce fuel consumption of the controlled vehicle and reduce delays of all vehicles of the corridor.

Future research is recommended to expand and enhance the eco-driving algorithm to multi-agent eco-driving method to simultaneously control multiple CAVs, and to communicate/coordinate with traffic signal control directly to develop a fully integrative vehicle-traffic control framework.

Table of Contents

List of Figures

List of Tables

Section 1: Introduction

Connected and Automated Vehicles (CAVs) are believed to have great promises for urban traffic control[(1)-(3)]. Usually, connected vehicles (CVs) refer to vehicles that can communicate with other vehicles (vehicle-to-vehicle, V2V), infrastructure (vehicle-to-infrastructure, V2I), and other traffic participants such as pedestrians and bicyclists (V2X). Fully automated vehicles refer to "the vehicle can do all the driving in all circumstances. The human occupants are just passengers and need never be involved in driving" [(4)]. The U.S. Department of Transportation's National Highway Traffic Safety Administration (NHTSA) defined five levels of automated driving, from driving assistance (Level 1) to fully automated vehicles (Level 5).

Much research has been conducted in the past decade for urban traffic control (UTC) with CAVs, while the primary focus has been on fully automated vehicles and 100% penetration of connectivity and automation (hereafter referred to as fully connected and automated vehicles (FCAVs)). However, it is becoming increasingly clear that it may take a relatively long time for us to reach a high penetration of vehicle connectivity and for full vehicle automation to become mature. In the near future, we will have to deal with a relatively low penetration of CAVs and limited level of vehicle automation (e.g., Level 2 or Level 3), hereafter referred to as "limited CAVs" (LCAVs) in this research.

The research team has worked on developing integrative vehicle-traffic control (IVTC) methods under FCAVs and connected vehicles (CVs) with full or limited penetration for the last decade[(5)-(9)]. The developed methods however have only been tested in traffic simulation. This research aims to extend the CAV-based traffic signal/vehicle control methods the PIs have developed in the past (many were supported by C2SMART) and test them in a vehicle in the loop (VIL) simulation environment to better understand and quantify the benefits of CAV-based control.

The team has collaborated with the EcoCar3 team at the University of Washington (http://ecocar3.org/washington/about-us/), and through that venue to work with a number of industry partners and agency partners, to refine/redevelop the previous methods and conduct VIL testing of vehicle-traffic control with LCAVs. The team also worked with the EcoCar3 VIP program at UW to involve students from different levels (Ph.D., graduate and undergraduate students) and backgrounds (especially those from underrepresented groups) to participate in this multidisciplinary and cutting edge research project.

Section 2: Reinforcement Leaning Based Eco-Driving

Subsection 2.1 2.1 Methods introduction

Improving the fuel efficiency of road vehicles has become a critical issue of the modern society since the transportation sector consumes about two thirds of the petroleum-based energy which has caused severe energy shortage and environmental problems[10][11]. There are generally two types of approaches to reduce the fuel consumption of vehicles on roadways: 1) Develop more fuel-efficient vehicle technologies such as lightweight, hybridization, advanced fuel-saving engines; and 2) Apply Eco-Driving strategies (and other traffic/transportation management strategies) so that vehicles can be driven at economic speed and making use of the kinetic energy[12]. The former has been the focus of fuel-saving technologies for many years and now requires major breakthroughs in new material/mechanical/energy sciences for further improvements, which can be time and capital consuming. Recently, Eco-Driving has received increasing attention due to its high fuel-saving potentials by only relying on existing vehicular technologies.

Traditionally, Eco-Driving was implemented by driver education programs[13][14], where drivers were taught about how to apply fuel-saving operations in their daily driving tasks. It was found that fuel consumption can be reduced right after the education program, which however may return back to normal in the long term due to human complacency and behavioral regression[15]. In the last decades, the connected and automated vehicles (CAVs) have been rising, making it possible to operate Eco-Driving strategies automatically. CAVs can communicate with other vehicles (V2V), roadside infrastructure (V2I), other road users such as pedestrians and bicyclists (V2X), and remote servers, which can provide the information of surrounding vehicles, high-resolution maps, traffic signals, etc. Such information can be processed by vehicle control algorithms to generate Eco-Driving strategies, which can then be operated by the actuators (i.e., the automation feature of CAVs). The U.S. Department of Transportation's National Highway Traffic Safety Administration (NHTSA) defined five levels of automated driving, from driving assistance (Level 1) to fully automated vehicles (Level 5)[4]. While low-level vehicle automation (e.g., Level 1 and Level 2) is maturing and increasingly deployed in vehicles, it is generally agreed now that it may take a relatively long time (10-20 years or more) for high level vehicle automation (e.g. Level 4 or Level 5) to become available and widely deployed in the vehicle fleet. The Eco-Driving methods presented in this paper can apply to low level (e.g., Level 2) automated vehicles, i.e. vehicles equipped with advanced driver assistant systems (ADAS) can control the lateral (i.e., lane-changing) and longitudinal (i.e. acceleration and deceleration) dynamics. Such requirements of low-level automation make the proposed method practical in the near future.

CAV-based Eco-Driving techniques have usually been studied by model-based methods. In the literature, the Eco-Driving problem was often formulated as an optimal control problem (OCP) where fuel

consumption and travel time were considered as the objective, and vehicle dynamics/signal plans/safety considerations were modeled as the constraints. Various solution methods such as dynamic programming (DP) and pseudo-spectral methods have been proposed to solve those OCPs. For examples, for vehicles equipped with continuous variable transmission (CVT), Li and Peng (2012)[16] solved the fuel-minimizing problem under the car-following scenario by Gauss pseudo-spectral method and found that the "pulse and glide"(PnG) operations performed the best. Later, researchers from the same group proposed servo-loop PnG controllers for CVT vehicles[17] and step-gear vehicles [18], which showed great improvement of fuel-saving performance. Hellström et al. (2009)[19] developed a DP-based look-ahead control method for heavy trucks, which could make use of the road slope database and GPS information to generate fuel-efficient speeds. For urban scenarios that consist of signalized intersections, the OCPs become more complex due to the additional constraints introduced by signals. Rakha and Kamalanathsharma (2011)[20] proposed a rule-based Eco-Driving model for a single vehicle driving across signalized intersections assuming no surrounding vehicles. The proposed rules specified that: 1) If the current signal phase is green and the ego vehicle (i.e., the studied vehicle) can accelerate to pass the intersection before the end of the green phase, it will accelerate to certain speed calculated by the kinetic equation; 2) If the ego vehicle has to decelerate to wait during the red phase, the speed profile of that deceleration process is optimized based on a VT-Micro emissions model. Xia, Boriboonsomsin, and Barth (2013)[21] also developed a logic-based dynamic Eco-Driving speed planning methodology for ego vehicles driving along corridors with multiple lanes. This method dynamically generated acceleration and deceleration profiles with trigonometric increments to ensure smooth trajectories. Based on simulation studies on multi-lane corridors with different penetration rates and traffic volumes, the method was shown to be effective for fuel savings.

Readers can refer to Huang et al. (2018)[22] for more detailed reviews on model-based Eco-Driving methods. In summary, model-based Eco-Driving methods need to simplify the dynamics of the environment and usually can only deal with one specific driving scenario (e.g., car-following, intersection passing, etc.). The models will become very complex if the driving scenario gets complex, making it hard to be numerically solved. Current model-based methods usually adopt rule-based schemes to decompose the overall problem into sub-problems which can be solved individually. As a result, the accuracy and generalization ability of these methods need further improvements.

Recently, learning-based, especially reinforcement learning (RL) based Eco-Driving methods have been receiving increasing attention due to its capability to overcome the above disadvantages. RL is an agent-oriented machine learning scheme based on the Markov decision process (MDP), where an agent learns to generate optimal actions to maximize the cumulative rewards based on repeated interactions with the environments[23]. The agent observes current state and reward from the environment, and selects actions based on the principle of maximizing the long-term cumulative reward. The environment receives and conducts the action, and generates new states and rewards, which will be feedback to the

agent. The action sequence under a series of states is the policy of the agent. In terms of improving the fuel efficiency, RL has been applied mainly in two fields: energy management system (EMS) and general Eco-Driving (GED), which are summarized in Table 1. Note that the value-based RL methods refer to those that learn the value functions of either states or actions based on temporal difference learning, and policy-based RL methods directly learn or approximate the optimal policy.

	Papers	Vehicle Type		Driving Scenario			RL Algorithm			Action				
		ICV	HV	CF	SU	ST	VB	PB	HB	LO	LA	IN	CO	DI
EMS	Qi et al. [17]	~	√	~	~	~	√	~	~	~	~	√	~	√
	Qi et al. [18]	~	√	~	~	~	√	~	~	~	~	√	~	√
	Qi et al. [19]	~	√	~	~	~	√	~	~	~	~	√	~	√
	Hu et al. [20]	~	√	~	~	~	√	~	~	~	~	√	~	√
GED	Shi et al. [21]	√	~	~	√	~	√	~	~	√	~	~	~	√
	Gamage and Lee [22]	√	~	~	√	~	√	~	~	√	~	~	~	√
	Gamage and Lee [23]	√	~	√	√	~	√	~	~	√	~	~	~	√
	Qu et al. [24]	~	√	√	√	~	~	√	~	√	~	~	√	~
	Hao et al. [25]	~	√	~	√	√	√	~	~	√	√	~	~	√
	Proposed Method	√	~	~	√	√	~	~	√	√	√	~	√	√

ICV – Internal combustion vehicles; HV – hybrid vehicles; CF – car-following; SU – signalized urban; ST – surrounding traffic; VB – value based; PB – policy-based; HB – hybrid; LO – longitudinal; LA – lateral; IN – internal actions CO – continuous; DI – discrete

Table 1 : Summary of current RL-based energy efficient studies

EMS is mainly designed for hybrid electrical vehicles (HEVs) or plugin hybrid electrical vehicles (PHEVs), which can be considered as an "internal" case of Eco-Driving. For examples, Qi et al. (2016)[24] proposed a tabular Q-learning based EMS for PHEVs to optimize the power-split control in real time. The speed of the ego vehicle, the road grade, percentage of remaining time to destination, the battery pack's state-of-charge, and the available charging gain of the upcoming charging station were used as the states. The discredited ICE power supply level was defined as the action. The reward function was designed to minimize the fuel cost while satisfying the power demand requirement. The numerical experiment with data synthesized from real-world traffic measurements showed that the proposed EMS algorithm could improve the fuel performance up to 12%. Researchers from the same group further modified this framework by adding a deep Q network (DQN) to better approximate the Q value[25][26]. Hu et al. (2018)[27] developed a deep Q-learning based EMS for hybrid electrical vehicles (HEVs). They used the total required torque and the battery state-of-charge (SOC) as the state and discretized the output torque from the ICE as the action. Numerical experiments based on MATLAB and advanced vehicle simulation (ADVISOR) co-simulation showed that the proposed EMS could outperform the rule-based EMS in terms of fuel economy.

EMS focuses on optimizing the internal energy flow of hybrid vehicles (HVs). The state is usually defined from the dynamics of the ego vehicle. GED, on the other hand, assumes there is an explicit relationship between the dynamics of the ego vehicle (e.g., longitudinal acceleration/deceleration) and the energy consumption, and tries to optimize the dynamics of the ego vehicle given the information of the surrounding environment (e.g., the surrounding vehicles, signals, etc.). In terms of GED, the ego vehicle is usually regarded as the agent, the longitudinal (i.e., acceleration and deceleration) and lateral (i.e., lane changing) operations of the ego vehicle can be modeled as actions, the driving scenario, including the dynamics of surrounding vehicles and the physical structure of intersections, can be considered as the environment, and the fuel consumption and travel time can be formulated as the rewards. A few studies in recent years have tried to apply the RL technique to GED. For examples, Shi et al. (2018)[28] proposed a Q-learning based Eco-Driving algorithm for a single vehicle driving on signalized intersections assuming no surrounding vehicles. They predefined the actions (i.e., the longitudinal acceleration) as discrete variables, and used the distance between the vehicle and the intersection, signal status, and instant vehicle speeds as traffic states. The total carbon dioxide was used as the reward. The Eco-Driving strategies learned by Q-learning were found to be able to reduce the fuel consumption as well as to improve the traffic performance. The same framework was used in Gamage and Lee (2016)[29]. Gamage and Lee (2017)[30] extended this framework to the car-following scenario. The state was represented by

the same three parameters as in Shi et al. (2018)[28] with an additional parameter of the headway between the ego vehicle and the leading vehicle. The reward function was constructed as inversely proportional to the cumulative fuel consumption experienced by the vehicle between two successive decision points. However, the positive results generated by the model in the paper was built on the assumption that the leading vehicle was under the free flow condition. Qu et al. (2020)[31] developed a deep deterministic policy gradient based algorithm for electrical CAVs under the car-following scenario to dampen the stop-and-go waves and improve the electric energy consumption. Experiment results showed that the proposed method could improve travel efficiency as well as reduce average electric energy consumption. These algorithms only considered the longitudinal operations while neglecting the lateral operations i.e., lane changing operations. Hao et al. (2020)[32] proposed a deep Q-learning based Eco-Driving framework to deal with the multi-lane scenario where the ego vehicle was surrounded with human-driven vehicles. More specifically, they used the camera data and the signal information gathered from ITS system as the state, and discretized the longitudinal acceleration/deceleration, which is used as the action space together with three discrete lane-changing indicators. They applied the Dueling Deep Q Network (DDQN) along with a long-short term reward (LSTR) function instead of the instant-variable to represent the reward function. Comparing with the intelligent driver model (IDM) and the Speed-First (SF) method, they showed that 12.25%–47.1% of energy savings can be achieved through this approach. In order to apply the value based RL method, they discretized the continuous action space (i.e., longitudinal acceleration/deceleration). Although such discretization could make the value-based RL applicable, several disadvantages were also introduced: 1) the longitudinal acceleration/deceleration may not be smooth, which will increase the fuel consumption due to the extra transient fuel consumption caused by the abrupt change of accelerations; 2) as a result of the first one, the comfortability of drivers will be sacrificed; and 3) the dimension of the discretized action space could be large such that the value-based RL algorithms may be computationally demanding.

In summary, studies that have applied the RL-based methods for GED to improve fuel economy under the signalized intersections and corridors are relatively sparse. Existing RL-based GED methods have some issues. First, they usually over-simplified the problem by e.g., assuming that there is only the ego vehicle driving along a single lane[28] or considering only the car-following scenario while neglecting the lane-changing behavior[30][31]. Second, the study that did consider surrounding vehicles and lateral operations, however, discretized the longitudinal acceleration/deceleration to reduce the dimension of the action space such that the value-based (e.g., Q-learning) RL algorithms can be applied[32], which would increase fuel consumption and reduce comfortability. This paper aims to fill these gaps by developing a new RL-based Eco-Driving algorithm for an ego vehicle driving along signalized roads considering the impacts of surrounding vehicles and both the longitudinal (i.e., acceleration/deceleration) and lateral (i.e., lane changing) operations. Unlike existing studies, we consider the longitudinal acceleration/deceleration as continuous variables (states). We develop a hybrid algorithm that integrates the deep Q-learning and policy gradient (HDQPG) to solve this problem.

As the key contribution of this paper, the HDQPG provides a general framework to model the Eco-Driving problem by integrating the continuous longitudinal action and the discrete lateral action (see Section 2.3 for details).

Subsection 2.2 Methodology

2.2.1 Preliminaries of RL

We first present some basic description of RL, which will be helpful when we introduce the Eco-Driving scenario in detail in the next section. RL is an MDP process where the agent learns to optimize his/her actions to maximize the cumulative rewards based on repeated interactions with the environment. The agent observes current state and reward from the environment, and selects actions based on the principle of maximizing the long-term cumulative reward. The environment receives and conducts the action, and generates new states and rewards, which will be feedback to the agent. Consider a general reinforcement learning problem: at time t, an agent observes the state $s_t \in S$ from the environment, performs an action $a_t \in A$ based on certain learning mechanism, and receives a reward $r_{t+1} = r(s_t, a_t)$ when the environment is moved to next state s_{t+1}. The whole process is assumed to be a Markov decision process (MDP). The decision sequence of the agent can be represented by the probability density of taking action a_t given state s_t, denoted as policy $\pi(a_t|s_t)$. Assume the environment (state??) s_t and the agent takes action a_t, the probability of the environment transferring to s_{t+1}, denoted as $p(s_{t+1}|s_t, a_t)$, can represent the dynamics of this MDP. The agent tries to maximize the expected discounted reward $r_t^\gamma \doteq r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k r(s_{t+k+1}, a_{t+k+1})$, where γ is the discount rate with $0 \leq \gamma \leq 1$. The discounted reward represents the present value of future rewards. Adding discounting rate γ can properly convert future rewards to the current value, which also guarantees that the total discounted reward r_t^γ is bounded. The value of a state s_t under a policy π is the expected discounted rewards when starting at state s_t and following the policy π thereafter, denoted as the state-value function for policy π: $V\_\pi\ (s\_t) \doteq E\_\pi\ [r\_t\hat{}\gamma\ |s\_t\ ] = E\_\pi\ [\sum\_(k = 0)\hat{}\infty\blacksquare\ \llbracket \gamma\hat{}k\, r(s\_(t + k + 1), a\_(t + k + 1))|s\_t\ \rrbracket\ ]$. Similarly, the value of taking action a_t at state s_t under policy π is the expected discounted rewards when starting at state s_t, taking action a_t, and following the policy π thereafter, denoted as the action-value (or Q-value) function for policy π: $Q\_\pi\ (s\_t, a\_t) \doteq E\_\pi\ [r\_t\hat{}\gamma\ |s\_t, a\_t\ ] = E\_\pi\ [\sum\_(k = 0)\hat{}\infty\blacksquare\ \llbracket \gamma\hat{}k\, r(s\_(t + k + 1), a\_(t + k + 1))|s\_t, a\_t\ \rrbracket\ ]$. Obviously, $V_\pi(s_t) = \sum_a \pi(a|s_t) Q_\pi(s_t, a)$.

2.2.2 Eco-Driving scenario

As shown in Figure 1, we focus on a CAV that drives across a signalized corridor along with other vehicles, aiming to minimize its fuel consumption and travel time. We call this vehicle the “ego” vehicle. In order to more clearly describe the studied problem, we make the following assumptions:

- The ego vehicle can get access to its own position/speed/acceleration from the controller area network (CAN) or on-board sensors.
- The ego vehicle can either detect the relative distance/speed/acceleration between itself and the surrounding vehicles by on-board sensors or be informed of such information by V2I communications.
- The longitudinal and lateral movements of the ego vehicle can be controlled automatically.
- Signal timing plans of the upcoming intersection can be sent to the ego vehicle through V2I communications.
- We consider multi-lane urban roadways where vehicles can change lanes during driving. For simplicity, three lanes are considered in this paper and the proposed RL framework can be properly modified for other multi-lane scenarios.

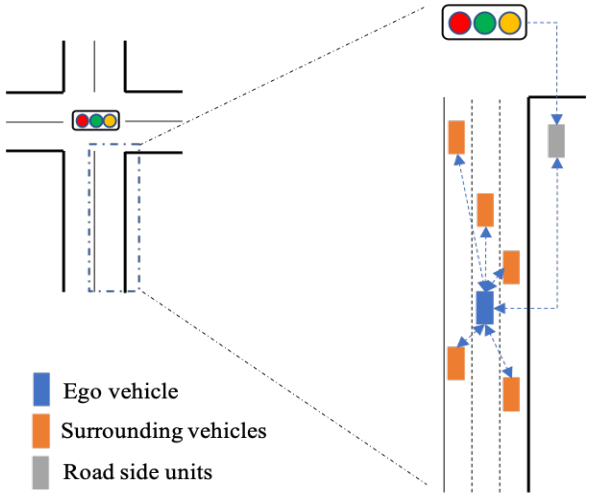

Figure 1: Eco-Driving scenario

Modeled under the RL framework, the ego vehicle is the agent and other vehicles, signals, roads, and the traffic rules together constitute the environment. There are two types of actions that the agent can take, i.e., the longitudinal and lateral accelerations. One can model the action space by three approaches. First, one can define the action space as a two dimensional (i.e., lateral and longitudinal) continuous vector. Second, noticing that people usually take a series of fixed operations after they have decided to change lanes, one can model the action space as a discrete-continuous hybrid space, i.e., one continuous longitudinal action (acceleration/deceleration) and one discrete lateral action (change to the left lane, change to the right lane, or stay at current lane). Third, based on the second approach, one can further discretize the longitudinal acceleration/deceleration so that all action candidates are discrete. The first approach is closest to real-world operations. However, when it comes to the RL-based control, although it is suitable to model the lateral acceleration/deceleration by continuous variables for scenarios without traffic lanes (such as racing competition), for regular driving scenarios with dedicated

lanes and traffic rules, such approach may introduce unpractical and even dangerous operations for lateral operations. For example, the ego-vehicle might be controlled to drive right above or very close to the lane marker for a relative long time, especially during the training process. Although one can design specific reward functions to penalize such “inappropriate” behavior, this could make the reward function in this paper more complex. In real-world driving, one usually takes a series of fixed actions to complete the lane-changing once s/he has decided to change lanes. Therefore, using discrete ``decisions'' to model the lateral operations may be a better choice. The third approach, as used in Hao et al. (2020)[32], can simplify the action space and make it possible to apply one unified value-based RL algorithm to deal with the Eco-Driving problem. However, it suffers from extra fuel consumption and poor convergence of the algorithm. The second approach can be interpreted as a combination of the first and third approach, which still treats the longitudinal acceleration/deceleration as continuous, while using discrete variables to represent whether the ego vehicle should change lane and if so, which lane should be targeted. In this way, the undesired lane changing behavior as described above for the first approach can be avoided. On the other hand, the continuous longitudinal acceleration/deceleration makes it possible to generate smooth acceleration/deceleration trajectories to reduce fuel consumption. Therefore, we adopt the second approach to model the action space in this paper.

By using the discrete-continuous hybrid action space, the Eco-Driving problem can be decomposed into two sub-problems: lateral lane-changing (i.e., deciding to stay on the current lane or change to the right lane or the left lane if needed) and single lane longitudinal operation (i.e., longitudinal acceleration/deceleration). We develop a hybrid framework to model and solve this problem. At a specific time, we first extract information of the lane where the ego vehicle resides (called the “ego lane”), which is used as the state of the longitudinal RL controller. The RL controller generates longitudinal actions (i.e., longitudinal acceleration/deceleration) given current state, which will be applied to the environment. We then assume that there are also two “virtual” ego vehicles, one for each of the two neighboring lanes with the same vehicle state as the real ego vehicle (i.e., distance to next intersection, speed, and acceleration). Similar to the state of the ego lane, the states of two neighboring lanes with the virtual vehicles can also be extracted. Note that although the states of the virtual ego vehicles are the same as the real ego vehicle, the full state of each neighboring lanes is different due to different states of the following vehicles. We then calculate the critic's value of each of the three lanes and use them as part of the state representations for the lateral lane-changing RL controller. Additional information that are useful but excluded from the single lane state (e.g., that related to the following vehicles in two neighboring lanes) are also gathered. Integrating the critic's values of the three lanes and additional lane-changing information, we build the state for the lateral lane-changing RL controller. Given such state, the lateral RL controller decides whether the ego vehicle should change to the right lane, change to the left lane, or stay on the current lane. Compared with directly using information of surrounding vehicles of the ego vehicle as the state of the lateral RL controller, the proposed state

representation, by introducing the concept of virtual ego vehicles, can help reduce the dimension of the state space.

It should be noted that the actions generated by the RL algorithms may not be achievable under the corresponding driving conditions. For example, the lateral RL decides to change to the left lane but it is too close to the front (or following) vehicle on the left lane (also reflected by the distance between the virtual ego vehicle and the front or following vehicle on that lane). To ensure safety, one can design safety-guaranteed RL algorithms to always generate safe decisions or behaviors. This however needs sophisticated RL design schemes such as dynamic output constraints, which can make the RL algorithm very complex. In this paper, we develop a "checking-feedback-learning" (CFL) framework to not only ensure safe (and consistent) driving behavior (or decisions) but also encourage the RL algorithms to learn safe and consistent behavior (via discouraging the learning of unsafe or inconsistent behaviors). Given the actions generated by RL algorithm, CFL conducts safety (or consistency) checks and will not conduct the actions if they are not safe or inconsistent with the original decision (for later control). Meanwhile, if unsafe or inconsistent behavior is detected, a penalty term will be added to the reward to penalize the unsafe/inconsistent decision. In this way, the CFL framework ensures safety and encourages the RL algorithms to learn safe driving behaviors and consistent decisions. In the following section, we show this hybrid RL framework in detail.

Subsection 2.3 Deep Hybrid Q-learning and Policy Gradient (DHQPG) for Eco-Driving

2.3.1 The longitudinal model

In this section, we discuss the longitudinal RL algorithm from four key components in RL: state, action, reward, and the learning algorithm. At time t, the distance between the ego vehicle and the next traffic signal (d_t^e), the ego vehicle's speed (v_t^e) and acceleration (a_t^e), the distance (Δd_t^e), speed difference (Δv_t^e), and acceleration difference (Δa_t^e) between the ego vehicle and the front vehicle, the time to next green phase of the traffic signal (ΔT_t, $\Delta T_t = 0$ if the current phase is green), and the time duration of the next green phase or the remaining green if current phase is green (T_t^g) are used as the state. These variables are selected based on the real driving experience and the assumed technology (i.e., V2I and the sensing capability of AVs). When one drives along a one-lane road, the information s/he usually uses to make driving decisions is related to the ego vehicle, the front vehicle, and the traffic signal. CAVs can obtain such information through advanced on-board sensors and CV technologies. However, extra information, such as the information of vehicles further downstream, is usually hard to collect. Therefore, we use an 8-dimension vector $(d_t^e, v_t^e, a_t^e, \Delta d_t^e, \Delta v_t^e, \Delta a_t^e, \Delta T_t, T_t^g)$ as the (continuous) state of the ego lane. Naturally, the longitudinal acceleration/deceleration serves the action of the single lane Eco-Driving algorithm, which is continuous and bounded.

In order to achieve the Eco-Driving task, three aspects should be considered when designing the reward function:

- First, the fuel consumption should be reduced. Therefore, we add the fuel consumption at the current time step g_t, multiplied by a negative weight $-w_f$, to the reward.

- Second, if minimizing fuel consumption is the only objective, the optimal action of the ego vehicle would be stopping at the origin with the engine shut down. Thus, in order to fulfill the travel needs, travel time should be kept at an acceptable level. The total travel time, which can only be calculated at the end of the travel, is a delayed reward and is hard to be distributed to each step of the whole MDP. In this paper, we use the travel distance (l_t)as a proxy to reflect the travel time. For each time step, since the elapsed time is fixed, the longer the travel distance is, the shorter the travel time would be.

- Third, the CFL framework discussed earlier is applied to ensure driving safety (e.g., avoiding collisions and following traffic lights) for longitudinal control. In order to do so, we introduce a concept of "safety speed" $v_{safe}^{e}(t)$, which is the maximum speed that the ego vehicle can drive at time t without breaking the safety constraints. We calculate the expected speed $\tilde{v}_t^e$ that the ego vehicle should achieve given action a_t at current speed v_t^e (i.e., $\tilde{v}\_t\hat{}e = v\_t\hat{}e + a\_t \times \Delta t$). If the expected speed is larger than the safety speed (i.e., $\tilde{v}\_t\hat{}e > v\_safe\hat{}e\ (t)$), which means the safety constraints are broken, we will modify the action (i.e, original a_t) to $(v\_safe\hat{}e\ (t) - v\_t\hat{}e)/\Delta t$ to make sure the action being input to the environment is legal. Meanwhile, in this case, we add the difference between the expected speed and the safety speed (i.e., $\tilde{v}\_t\hat{}e - v\_safe\hat{}e\ (t)$) to the reward function to penalize such actions that break the traffic rules.

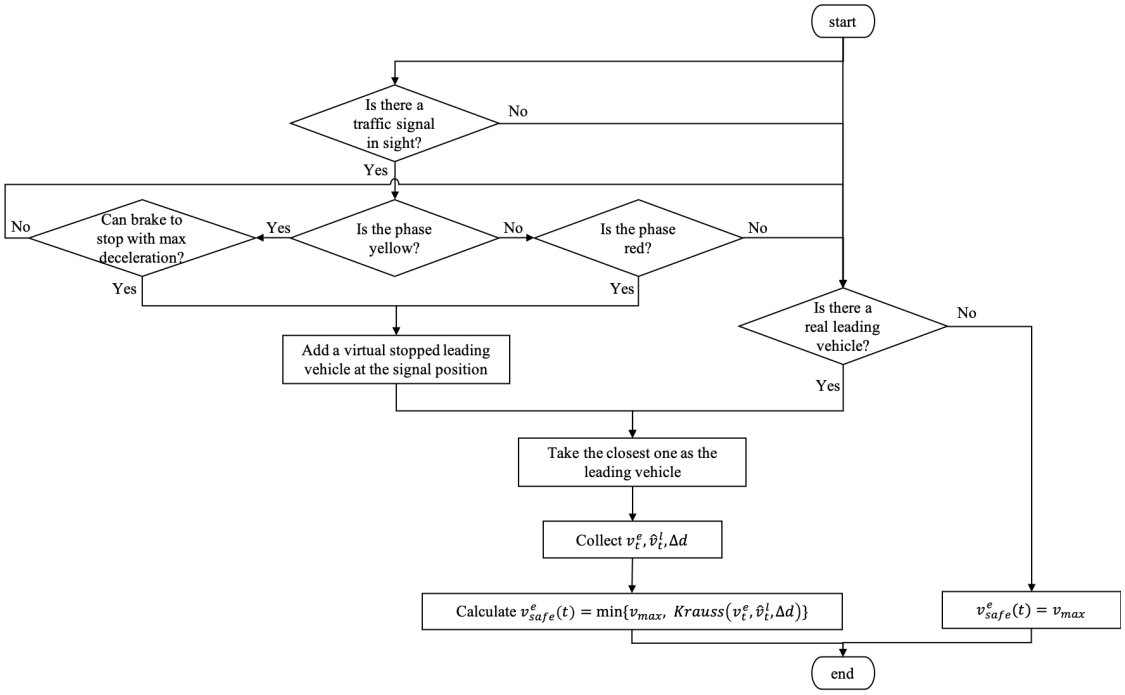

Figure 2: Flowchart of calculating the safety speed

Figure 2 shows the logic of how to calculate the safety speed $\tilde{v}_t^e$. The basic idea is to evaluate the traffic conditions in front of the ego vehicle. Under the simplest condition, where there is no upcoming traffic signal in sight (or there is a signal but with green phase) and no leading vehicle, we simply set the safety speed as the speed limit of the current road. If there is a traffic signal in sight, we check the phase of the upcoming signal. If the phase is red or is yellow but the ego vehicle can brake to stop with maximum deceleration, we'll create a virtual stopped leading vehicle (i.e., a leading vehicle with 0 speed) at the signal position. Meanwhile, we check if there is a real leading vehicle. If yes, we take the closest one from the virtual vehicle (which is generated due to the red/yellow phase of the upcoming signal) and real leading vehicles as the selected leading vehicle for the purpose of calculating $\tilde{v}_t^e$. We then collect the current action a_t, current speed of the ego vehicle v_t^e, current speed of the selected leading vehicle $\hat{v}_t^l$, and the gap between the ego vehicle and selected leading vehicle Δd_t, and use these variables to calculate the safety speed by the Krauss car-following model (Krauss, Wagner, and Gawron (1997); Krauss (1998)). Essentially, the safety speed of Krauss model is calculated as

$$v_{safe}^{Krauss}(t) = \hat{v}_t^l + \left(\Delta d_t - \hat{v}_t^l \tau\right) / \left(\frac{\hat{v}_t^l + \hat{v}_t^e}{2a_{max}} + \tau\right)$$

where τ is the human reaction time and a_{max} is the maximum deceleration of the ego vehicle. In addition, the speed should be bounded by the speed limit. So, we take the minimum of the Krauss safety speed and the speed limit as the final safety state under such conditions.

$$v_{safe}^{e}(t) = min\{v_{max}, v_{safe}^{Krauss}(t)\}$$

In order to make the three different terms comparable as well as to adjust our preference, we add two weight parameters w_f and w_s to the fuel consumption and safety consideration, respectively. These two weights can be interpreted as two parameters that can convert fuel consumption and safety considerations to "distance".

$$r_{lg,t} = -w_f g_t + l_t - w_s max\{0, \tilde{v}_t^e - v_{safe}^e(t)\}$$

The discounted cumulative reward is then defined by

$$r_{lg,t}^{\gamma} = \sum_{k=0}^{\infty} \gamma^k r_{lg,t+k+1}$$

The longitudinal Eco-Driving problem can be regarded as an MDP with a continuous action space. For such problems, of which the continuous action space is a key characteristic, the policy gradient (PG) based RL is usually used as the solution method. From the policy point of view, the agent's goal is to find a policy π that can maximize the cumulative discounted reward from the start state s_0: $J(\pi) = E[r_0^{\gamma}|\pi]$. For the longitudinal Eco-Driving problem, π is a series of longitudinal acceleration/deceleration and $r_0^{\gamma} = r_{lg,0}^{\gamma} = \sum_{k=0}^{\infty} \gamma^k r_{lg,k}$. Given a state s_t, we use a parameterized function to approximate the policy $\pi_\theta(a_t|s_t) \sim \pi(a_t|s_t)$. Note that in this section, we use θ as a common parameter notation for the approximation functions of both stochastic policy π_θ and deterministic policy μ_θ (which will be introduced later). The performance function then becomes $J(\pi) \sim J(\pi_\theta)$. Let $p(s \to s', k, \pi_\theta)$ be the probability of going from state s to state s' after k steps under policy π_θ. Define the discounted state distribution $\rho_{\pi_\theta}(s') = \int_S p_1(s) \sum_{k=0}^{\infty} \gamma^k p(s \to s', k, \pi_\theta)$ where p_1 is the initial state distribution. The performance function can be expressed as

$$J(\pi_\theta) = \int_S \rho^{\pi_\theta}(s) \int_A \pi_\theta(s|a) r(s,a) da ds$$

According to the policy gradient theorem (Sutton et al., 1999), we can calculate the partial derivative of the performance function

$$\nabla_\theta J(\pi_\theta) = \int_S \rho^{\pi_\theta}(s) \int_A \nabla_\theta \pi_\theta(a|s) Q^{\pi_\theta}(s,a)\, da ds = E_{s\sim\rho^{\pi_\theta}, a\sim\pi_\theta}[\nabla_\theta log \pi_\theta(a|s) Q^{\pi_\theta}(s,a)]$$

This is called stochastic policy gradient since π_θ is a probability density function. Practically, the policy usually becomes deterministic as the policy gradient algorithm converges. However, the policy gradient $\nabla \pi_\theta$ fluctuates more rapidly near the mean, making it hard to estimate the stochastic policy [33]. To

overcome this disadvantage, Silver et al. (2014)[34] proposed the deterministic policy gradient algorithm (DPG) by introducing a deterministic policy $\mu_\theta(s): S \to A$ parameterized with $\theta \in R^n$. The performance function becomes

$$J(\mu_\theta) = \int_S \rho^{\mu_\theta}(s) r(s, \mu_\theta(s))\, ds$$

And the deterministic gradient theorem shows that if $p(s'|s,a)$, $\nabla_a p(s'|s,a)$, $\mu_\theta(s)$, $\nabla_\theta \mu_\theta(s)$, $r(s,a)$, $\nabla_a r(s,a)$, $p_1(s)$ are continuous for all parameters, we have

$$\nabla_\theta J(\mu_\theta) = \int_S \rho^{\mu_\theta}(s) \nabla_\theta \mu_\theta(s) \nabla_a Q^{\mu_\theta}(s,a)|_{a=\mu_\theta(s)}\, ds = E_{s\sim\rho^{\mu_\theta}}\left[\nabla_\theta \mu_\theta(s) \nabla_a Q^{\mu_\theta}(s,a)|_{a=\mu_\theta(s)}\right]$$

Given the above theories, the policy gradient can be implemented by various methods, of which the actor-critic is a widely used scheme. The actor tries to find the optimal policy using the policy gradient. The critic estimates the action value $Q^{\pi_\theta}(s,a)$ or $Q^{\mu_\theta}(s,a)$ by certain policy evaluation algorithms such as Monte Carlo method and time differential method. Note that $Q^{\pi_\theta}(s,a)$ and $Q^{\mu_\theta}(s,a)$ are the true action value function, we use a parameterized function $Q^{\omega}(s,a)$ to represent the approximated action value function. Silver et al. (2014)[34] proved that if $Q^{\omega}(s,a)$ is of the form $Q^{\omega}(s,a) = (a-\mu_\theta(s))^{\top} \nabla_\theta \mu_\theta(s)^{\top} w + V^{v}(s)$ and ω minimizes the mean square error between $Q^{\omega}(s,a)$ and $Q^{\mu_\theta}(s,a)$, the function approximator $Q^{\omega}(s,a)$ is compatible, i.e., the gradient $Q^{\mu_\theta}(s,a)$ can be replaced by $Q^{\omega}(s,a)$ without affecting the deterministic policy gradient. In real world problems, however, non-linear approximators are usually needed especially in order to learn and generalize under large state spaces. To solve this problem, Lillicrap et al. (2015)[35] proposed the deep deterministic policy gradient (DDPG) by using neural networks to approximate the action value function for the critic. The numerical experiment results showed that the DDPG could find policies as good as those found by planning algorithm with full access to the dynamics and its derivatives, which means that the DDPG could learn policies that are close to the global optimal policies in an "end-to-end" manner. In order to capture the potential nonlinearity of the actor's policy function and the critic's Q function, we adopt the DDPG method to learn the optimal actions. Although the convergence cannot be guaranteed when using deep neural network approximates, our experimental results demonstrate stable learning. We design two deep neural networks for the actor and critic separately. Similar to Lillicrap et al. (2015)[35], another two soft target networks are used to stabilize the learning process.

2.3.2 The lateral model

The most intuitive way to construct the state for the lateral lane-changing RL model is to directly extract the information of surrounding vehicles, i.e., the position, speed, and acceleration of the vehicles around the ego vehicle. Usually, information of at least five vehicles should be collected: the front

vehicle on the ego lane, and the front and following vehicles of the other two neighboring lanes. Adding more information of the ego vehicle and traffic signal (i.e., time to the next green time and the duration of the green time), the dimension of such a state space is $5 \times 3 + 3 + 2 = 20$. In this paper, we reduce the dimension of the state space by introducing a virtual ego vehicle to each of the two neighboring lanes. For each lane, we can construct the single-lane state. Applying the longitudinal DDPG algorithm to each of the three lanes, we can find i) the longitudinal action for the next time step based on the state of real ego lane; and ii) the critic's value for all three lanes. The critic's value represents the potential cumulative reward of each lane, determined by the (virtual) ego vehicle and the front vehicle. We thus use the critic's value of each lane to replace the states of the front vehicle and the (virtual) ego vehicle in the state representation of the lateral RL controller. This results in four variables to represent the state of each of the two neighboring lanes, i.e., the critic's value and the position/speed/acceleration difference between the virtual ego vehicle and the following vehicle. For the ego lane, only the critic's value is used. Therefore, by introducing the virtual ego vehicles, the dimension of the state space for the lateral RL model would reduce to $4 \times 2 + 1 = 9$, about half of the dimension of the state space if virtual ego vehicles were not used. It should be noted that adding more information from more surrounding vehicles might enable the ego vehicle to learn more useful maneuvers. However, collecting data beyond the above-discussed five vehicles is usually much harder than just collecting data from these five vehicles, especially under current vehicle and traffic technologies. Therefore, we choose to use only the data of the five neighboring vehicles for lateral control in this paper.

The lateral RL controller should be able to identify from the state vector which part is for the information of the ego vehicle, which parts are for the information of surrounding vehicles, and what is the relative location among them. The first two are easy to implement. For example, one can use a vector with nine elements as the state, where the first element represents the ego lane's information, and the rest eight elements represent the information of the other vehicles. However, this approach cannot indicate which lane the ego vehicle resides. To do so and to capture all possible ego lane location scenarios (i.e., left most, middle, right most), we design an encoding method shown in Figure 3 for the state space representation. As aforementioned, the state of the ego lane consists of a single variable (i.e., the critic's value of the ego lane), which is used as the first segment. Then, we add four segment holders for the rest of the two lanes (the dimension of each of the segments is four). The number (-2, -1, 1, 2) indicates the relative position of the segment (lane) with respect to the ego-lane: positive means to the right and negative means to the left. If the ego vehicle locates at the left most lane, which means both the other two lanes are to its right, we place the state of the adjacent right lane to the segment "Right lane +1" and place the state of the right most lane to the segment "Right lane +2". If the ego vehicle locates at the middle lane, we place the state of the adjacent right lane to the segment "Right lane +1" and place the state of the adjacent left lane to the segment "Left lane -1" If the ego vehicle locates at the right most lane, we place the state of the adjacent left lane to the segment "Left lane -1" and place the state of the left most lane to the segment "Left lane -2" In this way, the relative position

of the ego lane and the other two lanes can be compatibly encoded. Notice that although there are four segments (in addition to the first one for the ego-vehicle), for a given scenario, only two of them will be occupied which represent the states for the two neighboring lanes. That is, the state space has a dimension of nine. It is also clear that if there are more or fewer than three lanes, the encoding method shown in Figure 3 can still apply with straightforward modifications.

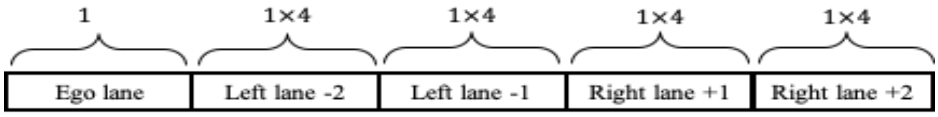

Figure 3: State representation of lateral RL controller

If the lateral RL controller generates a lane-changing decision to "change to the left" lane or "change to the right lane", we will first check whether such a decision can be completed or not, i.e., whether the traffic conditions allow the ego vehicle to change to the target lane safely. If the lane-changing decision is allowed, we will conduct it in the following 3 seconds (we assume that the lane-changing can be completed in 3 seconds with a series of fixed operations). If the lane-changing decision is not allowed since it violates traffic rules or not safe (e.g., the distance to the front or following vehicle on the target lane is too short), the ego vehicle will be kept on the current lane. Meanwhile, a penalty term is added to the reward in this case, which provides feedback to the RL algorithm to discourage its learning of this unsafe behavior (or encourage the RL algorithm to learn safe behaviors). If the lane-changing decision is allowed, the 3 seconds after the lane-changing decision will be a "protected" period. If, subsequently, the RL controller generates any decision that conflicts with the original lane-changing decision during this protected period, the new decision will be regarded as not allowed. If this occurs, we will keep completing the original lane-changing decision and ignore the new decision. Meanwhile, we also add a penalty term to the reward to provide feedback to the RL algorithm which discourages its learning of the inconsistent decision. Here we apply the CFL framework to ensure safety and encourage the lateral RL algorithm to learn safe behavior and consistent decisions.

We use the following reward function for time step t:

$$r_{la,t} = -w_f g_t + l_t - w_l \times 1_A(x_t)$$

where 1_A is an indicator function (i.e., $1_A(x) = 1$ if $x \in A$ and $1_A(x) = 0$ if $x \notin A$). A is the set of events that the lane-changing behavior cannot be completed due to the violation of traffic rules, x_t is the current lane-changing states, w_l is the penalty for generating the lane-changing decisions that cannot be achieved, which encourages the RL algorithm to learn lane-changing oriented traffic rules. Similar to the longitudinal RL controller, the discounted cumulative reward for lateral algorithm is then defined by

$$r_{la,t}^{\gamma} = \sum_{k=0}^{\infty} \gamma^k r_{la,t+k+1}$$

The lateral lane-changing problem has a discrete action space consisting of three variables. We use Q-learning[36] as the basis of the lateral RL algorithm due to its success on dealing with RL problems with small dimension discrete action spaces. At current state s_t, the agent estimates the values of taking every possible action a_t (i.e, change to the left lane, change to the right lane, or stay at current lane), and selects the action that can maximize the long-term reward (i.e., selects the target lane that can minimize the travel time without breaking the safety constraints). The Q-values are learned iteratively by

$$Q_\pi(s_t, a_t) \leftarrow Q_\pi(s_t, a_t) + \alpha(r_{t+1} + \gamma \max_a Q_\pi(s_{t+1}, a) - Q_\pi(s_t, a_t))$$

where α denotes the learning rate. Notice that r_{t+1} is actually $r_{la,t+1}$ for the studied lateral RL controller. Tabular Q-learning can give satisfactory control orders if the state and action spaces are small. However, when dealing with large state and action spaces, Q-learning becomes quite inefficient or even unrealistic due to the large amount of memory required to store/update the table and the large amount of time required to explore possible states. Thus, function approximation methods such as using deep neural network (i.e., the deep Q-network, DQN) to estimate the Q-values are developed[37] to solve this problem. In DQN, the Q-values are approximated by a parameterized neural network (denoted as $Q_\pi(s_t, a_t|\theta^Q)$). The update of Q-values then becomes the update of network parameters θ by minimizing the loss function:

$$L(\theta) \approx (r_{t+1} + \gamma \max_a Q_\pi(s_{t+1}, a|\theta^Q) - Q_\pi(s_t, a_t|\theta^Q))^2$$

There are several key considerations in DQN. First, when optimizing the network parameters θ, most optimization algorithms require that the samples should be independently distributed. The transitions collected along an MDP cannot directly satisfy this requirement due to the temporal correlation. In DQN-based RL, it's common to use replay buffer to solve this problem. The replay buffer is a memory buffer with finite fixed spaces. During the learning process, the transitions $(s_t, a_t, r_{t+1}, s_{t+1})$ are collected and stored in the replay buffer. When the replay buffer is full, the oldest ones will be replaced by the latest ones. At each training step, a minibatch will be sampled uniformly from the replay buffer to update the DQN. Another advantage of using replay buffer is that one can make efficient use of the stored samples and accelerate hardware optimizations[38]. Second, the learning process is proved to be unstable if directly implementing above loss function, because $Q_\pi(s_t, a_t|\theta^Q)$ is used for both calculating the target Q-values for the next step and updating the current Q-network, which may result in fluctuations and even divergence[39]. In this paper, we create a target network $Q'_\pi(s_t, a_t|\theta^{Q'})$ to calculate the target Q-values[40][35]. The loss function then becomes

$$L(\theta) \approx (r_{t+1} + \gamma \max_{a} Q'_{\pi}(s_{t+1}, a|\theta^{Q'}) - Q_{\pi}(s_t, a_t|\theta^{Q}))^2$$

The parameter of the target network $\theta^{Q'}$ is soft updated by $\theta^{Q'} \leftarrow \tau\theta^{Q} + (1-\tau)\theta^{Q'}$ where $\tau \leq 1$, which means the target network will slowly track the learned network regulated by τ. This method has been shown to be able to stable the learning process[35]. For the Eco-Driving problem in this paper, consider that the state of the lateral lane-changing problem is large and continuous, we use two deep neural networks (i.e., one current network and one target network) to approximate the Q-values.

Subsection 2.4 Numerical Experiment

As shown in Figure 4, a corridor that consists of five intersections is built in SUMO[41] to test the DHQPG algorithm. Different distances between adjacent intersections are generated to test the performance of the algorithm. In this section, we first set the number of lanes of each road as 1 (i.e., no lane-changing or by-passing) to test the single lane DDPG algorithm, by which we can show the longitudinal Eco-Driving strategies more clearly. Then, the number of lanes is increased to 3 and the HDQPG is tested. We compare the proposed HDQPG with two existing methods:

1. The default car-following (i.e., Karuss model[42][43]) and lane-changing model (i.e., LC2013, developed by Erdmann (2014)[44]) used in SUMO. This case is used as the baseline case.

2. The DQN based RL method, which is widely used in current studies considering Eco-Driving [28][29][30][32]. The discretized longitudinal acceleration/deceleration, along with three lane-changing indicators, are used as the actions, which means that the third action space modeling approach is used. One drawback of DQN is that, as the action space increases, the algorithm will suffer from instability and even fail to learn the optimal actions. However, the larger the discretization interval is, the larger the gap between reality and the model is, and the less comfort the drivers will be. Based on our pre-experiment, the discretization interval is set to $1m/s^2$. The range of the longitudinal acceleration/deceleration is $[-5m/s^2, 3m/s^2]$. Therefore, the action space is $\{-5m/s^2, -4m/s^2, -3m/s^2, -2m/s^2, -1m/s^2, 0m/s^2, 1m/s^2, 2m/s^2, 3m/s^2, -1, 0, 1\}$, where the first 9 are discretized longitudinal acceleration/deceleration, and the last three are lange-changing indicators (i.e., -1 for changing to the left lane, 0 for stay at current lane, 1 for changing to the right lane). The reward function of the DQN is defined as the sum of the longitudinal (i.e., $r^{dqn} = -w_f g_t + l_t - w_s max\{0, \tilde{v}_t^e - v_{safe}^e(t)\} - w_l \times 1_A(x_t)$.

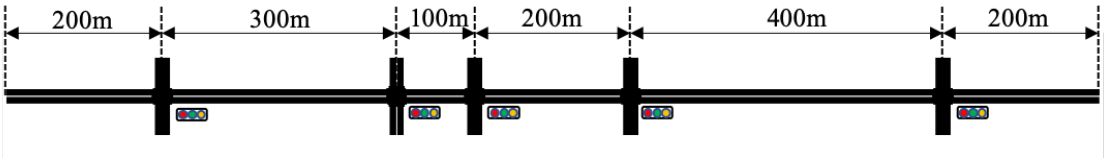

Figure 4: Test scenario

2.4.1 Single lane Eco-Driving

For the DDPG algorithm used to learn the single lane Eco-Driving strategies, we use two $N_i \times 128 \times 64 \times 1$ deep neural networks to approximate the actor and critic functions, where N_i is the dimension of the state of actor and critic, respectively. For the actor network, the activate function of the output layer is set as $tanh$ such that the output value is continuous in $[-1,1]$, which is then transformed to the acceleration value by a linear mapping function based on the bounds of acceleration. The learning rate of the actor is set as 0.001 and the one of the critic is set as 0.002 based on experiences and our preliminary experiments. After trying different discount ratio from 0.8 to 0.99, we set the discount ratio $\gamma = 0.95$. Similarly, different memory capacity values are tested, which is set as 10000 finally. The soft replacement parameter is set as $\tau = 0.001$ and the batch size is set as $m = 64$.

The volume of this corridor is set as 400 veh/h. The ego vehicle is sampled after the first 300 seconds to guarantee that it's not the first departured vehicle in the current episode. When the ego vehicle exits the corridor, the current episode terminates. We train the DDPG algorithm for 1000 episodes, each of which uses different random seeds. Every 10 training episodes, the learned strategies are tested by 5 testing episodes. The baseline case is generated by running the SUMO simulation without any external control, i.e., the ego vehicle is controlled by SUMO's default models. We test the proposed algorithm under two signal settings: 1) non-coordinated signal plan where the phases and offsets along the corridor are not optimized; and 2) coordinated signal plan where the phases and offsets are optimized based on the free flow speed between adjacent intersections to generate green waves.

Notice that we consider both fuel consumption and travel time in the reward function, the performance of which might be quite different under different weights. Increasing w_f should improve the fuel performance but might reduce travel time. Although we focus on Eco-Driving, the travel time should not be sacrificed too much. In order to balance these two objectives, we fix the parameter $w_s = 1$ and train the DDPG algorithm under different w_f. The results are shown in Figure 5.

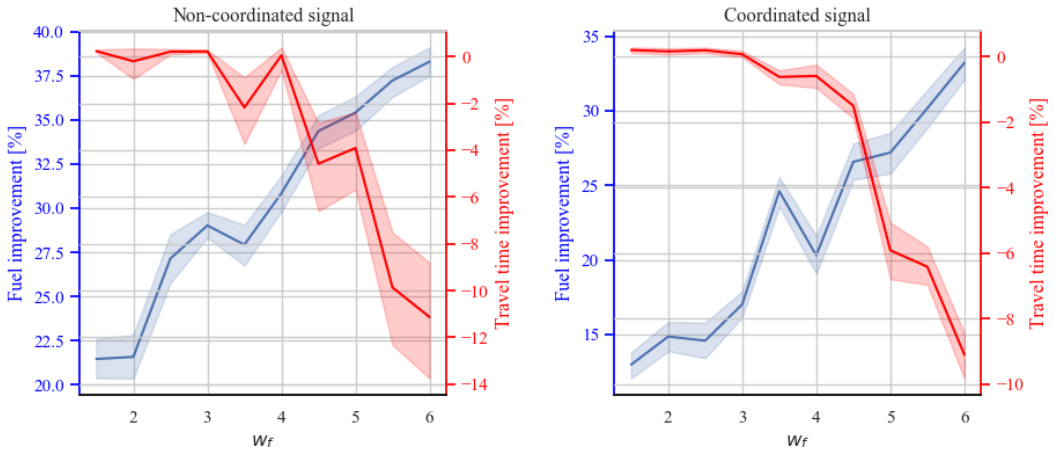

Figure 5: Fuel and travel time improvements under different w_f for non-coordinated and coordinated signal

Under the non-coordinated signal setting, as w_f increases from 1.5 to 4, the performance of fuel consumption improves from 22% to 31% while the travel time maintains at the same level as the baseline case. As w_f keeps increasing, although the fuel consumption could be further reduced, the travel time would dramatically increase. The same trend can be found under the coordinated signal setting. The fuel-saving increases from 13% to 27 % when w_f increases from 1.5 to 4.5, while the travel time increment is limited under 2 %. As w_f keeps increasing, the performance of travel time will decrease dramatically. Therefore, we choose $w_f = 4.5$ as the best balance between the performances of fuel and travel time, since the performance of fuel consumption can be improved largely, and the increment of travel time can be limited under 5 % for both signal settings. A same process for the DQN based Eco-Driving is conducted, and $w_f = 5$ is chosen.

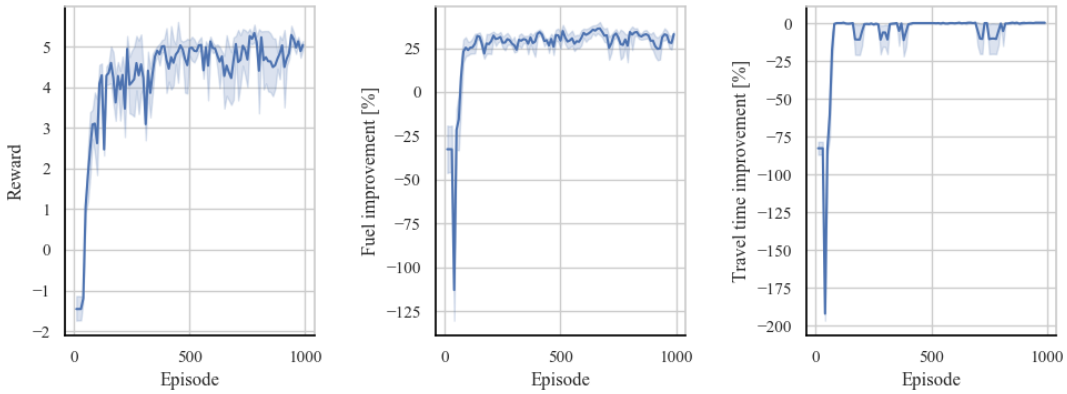

Figure 6: Performance of DDPG-based single lane Eco-Driving under coordinated signal

Figure 6 shows the reward (with the weight parameters $w_f = 4.5, w_s = 1.0$), fuel and travel time improvements of the DDPG-based single lane Eco-Driving compared with the baseline case during the training process (under coordinated signal setting, the same trend is found under non-coordinated signal and is omitted here for brief). It is shown that the reward decreases at the starting period, but then increases dramatically in less than 50 episodes. After that, the reward increases slowly with fluctuation and finally stabilize around 5. The fuel consumption improvement shows a similar trend. The the fuel improvement decreases dramatically at the early training period (i.e., the fuel consumption increases dramatically), but then increases over 0 and finally stabilize around 27%. Meanwhile, the travel time becomes a little worse, but is limited below 5% longer compared with the baseline case.

Table 2 shows the performance of the three control algorithms. It is shown that, under both coordinated and non-coordinated signal settings, both the DQN based and DDPG based RL controllers can improve the fuel performance compared with the baseline case. Meanwhile, the DDPG significantly outperforms the DQN in terms of reducing fuel consumption. In terms of travel time, the DQN outperforms DDPG under the coordinated signal setting. However, under the non-coordinated signal setting, the DQN is unstable with large deviations, while the DDPG can maintain almost the same performance as the baseline case. We then increase the test cases to 100 to check if the above findings are consistent among different test cases. The results are shown in Table 3. We can see that with more test cases, DDPG actually outperforms DQN: the mean and standard deviation of either fuel consumption or travel time are smaller for DDPG for both the coordinated and coordinated settings. DDPG also outperforms the baseline algorithm in terms of fuel consumption with slight degradation (less than 5\%) when travel time is considered.

	Case	Fuel consumption (ml)					Travel time (s)				
		Base	DQN	Imp.	DDPG	Imp.	Base	DQN	Imp.	DDPG	Imp.
Coord.	Case 1	133.4	111.57	16.3%	95.94	28.38%	95	95	0.00%	100	-5.26%
	Case 2	135.94	122.98	9.53%	93.15	31.48%	99	99	0.00%	103	-4.04%
	Case 3	184.46	162.07	12.14%	116.56	36.81%	135	135	0.00%	140	-3.70%
	Case 4	132.37	120.71	8.81%	88.77	32.94%	98	98	0.00%	104	-6.12%
	Case 5	170.16	148.41	12.79%	109.52	35.64%	131	131	0.00%	135	-3.05%
Non-coord.	Case 1	185.48	158.52	14.53%	134.78	27.34%	151	151	0.00%	151	0.00%
	Case 2	198.12	170.86	13.76%	131.66	33.55%	155	155	0.00%	155	0.00%
	Case 3	216.99	214.03	1.36%	144.39	33.46%	155	194	-25.16%	154	0.65%
	Case 4	182.72	169.4	7.29%	118.92	34.91%	152	152	0.00%	152	0.00%
	Case 5	217.36	215.23	0.98%	134.25	38.24%	150	189	-26.00%	149	0.67%

Table 2: Performance of different single lane control algorithms under two signal settings

	Fuel Consumption (mL)			Travel time (s)		
	Base	DQN	DDPG	Base	DQN	DDPG

Coord.	Avg	150.46	147.57	112.82	110.82	116.29	112.32
	Std	21.61	24,35	19.13	17.81	18.81	16.10
Non-coord.	Avg	119.76	214.81	138.3	156.04	167.77	162.68
	Std	24.11	32.57	19.13	16.58	22.87	20.99

Table 3: Performance of different single lane control algorithms under two signal settings (summary of 100 test cases)

Figure 7 shows the spatial-temporal diagrams of the ego vehicle for the first four test cases in Table 2 under the non-coordinated signal setting (Case 5 shows the same pattern as Case 3 and is omitted here due to space limitation). The only difference among the four figures is that they use different random seeds, i.e., the times that the ego vehicle entering the corridor are different. For all the three control methods, under the same case, all components of the simulation including the IDs and entering time of the ego vehicle and its front vehicle are the same. It is shown that the key difference of the driving strategies between the baseline and the two RL-based control is that, if the ego vehicle cannot pass the intersection in current green phase, the RL-based Eco-Driving will slow down the ego vehicle earlier compared with the baseline control. To illustrate it more clearly, we show the spatial-temporal diagram, speed, and accumulated fuel consumption of test case 2 in Figure 8. Under the baseline case, the ego vehicle maintains a high speed to approach the intersection, then decelerate to zero within a relatively short time interval. Under both the DQN and DDPG based RL controllers, the ego vehicle starts to decelerate earlier than the baseline case. Since the fuel consumption rate is zero when decelerating, the two RL controllers could save fuel through this strategy. Compared with the DQN, the speed profile learned by the DDPG is more earlier and smooth without large disruptions. DQN can only generate discrete acceleration/deceleration values, which will make the speed profile non-smooth if the optimal actions are not perfectly learned. Such disturbances will not only increase the fuel consumption, but also introduce delays that can be amplified by traffic signals, which can be shown in test case 3 of Figure 7.

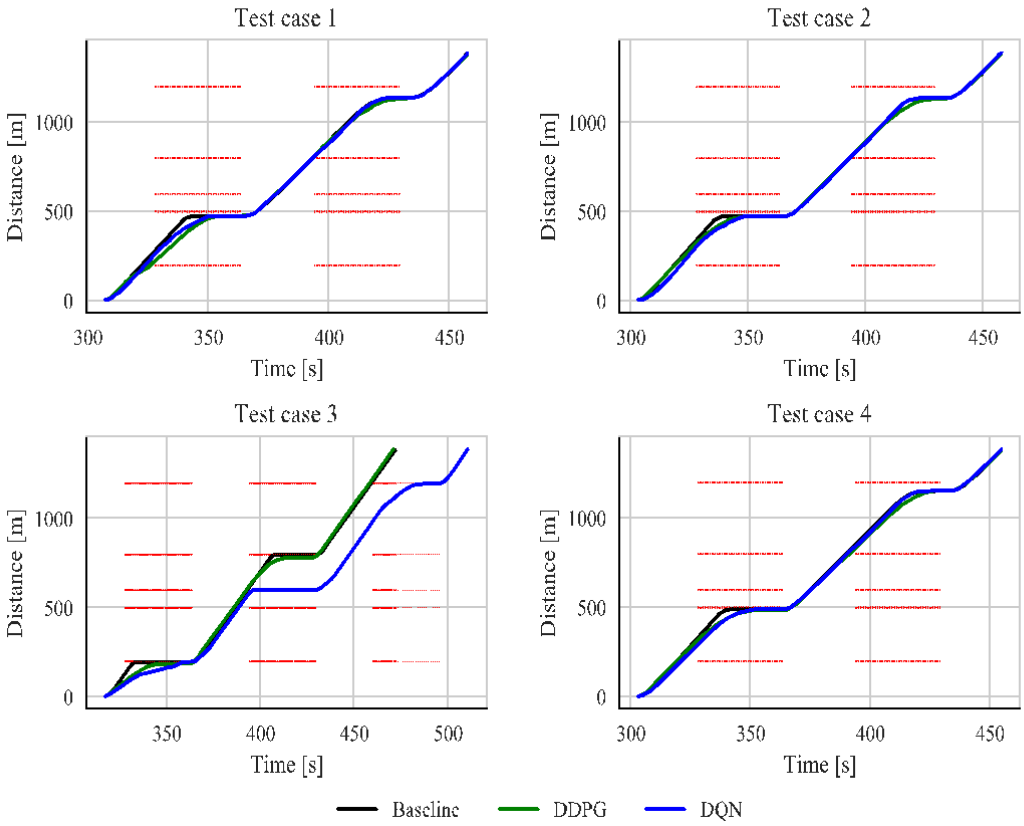

Figure 7: Spatial-temporal diagrams of the ego vehicle under non-coordinated signal setting

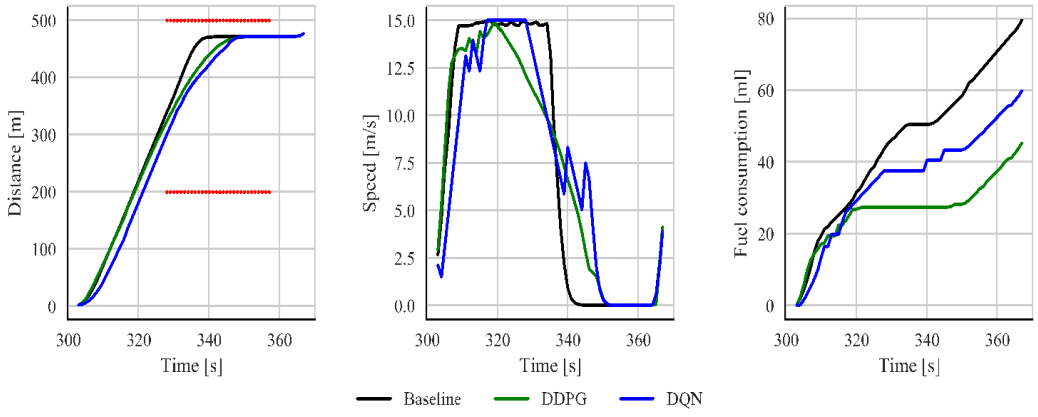

Figure 8: Spatial-temporal diagram, speed, and accumulated fuel of test case 2 under non-coordinated signal setting

In Figure 7 and Figure 8 the signal plans of five intersection on the corridor are not coordinated. The ego vehicle encounters multiple stop-and-goes under every test case, which illustrates the first fuel-saving strategy learned by RL controllers. We then illustrate the second fuel-saving strategy learned by the DDPG-based Eco-Driving algorithm by analyzing the performance under coordinated signal setting.

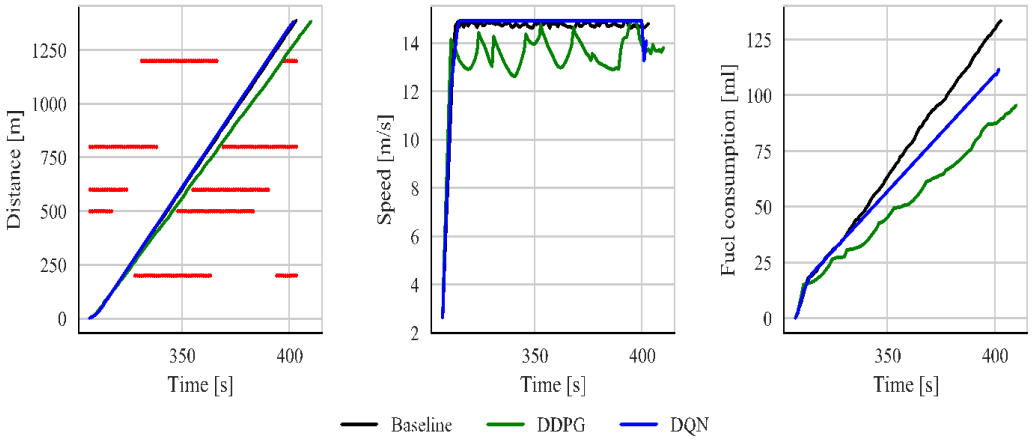

Figure 9: Spatial-temporal diagram, speed, and accumulated fuel of test case 2 under coordinated signal setting

Figure 9 shows the spatial-temporal diagram, speed and accumulated fuel consumption of test case 2 under coordinated signal setting. It is shown that the ego vehicle of baseline case firstly accelerates to a high speed and maintains that speed with slight fluctuations during the whole simulation period. Under DQN, the ego vehicle behaves in the same way. Since DQN can only generate discrete accelerations with $1m/s^2$ interval, the learned acceleration of the ego vehicle during the cruising period is 0. Therefore, the speed profile under DQN is a straight line during most of the cruising period. Such operations can reduce the fuel consumption caused by the speed fluctuations. Under DDPG control, the ego vehicle also firstly accelerates to a high speed, then periodically "pulse and glide" around a high speed. This PnG operation has been proved to be fuel-efficient due to the non-linearity of the fuel consumption characteristics of engines[45][18][17]. This phenomenon becomes more and more clear as the fuel weight w_f increases. Under low w_f, the ego vehicle cares more about the travel time, which encourages high speed. Under higher w_f, saving fuel is given higher priority, the ego vehicle learns the PnG strategy since the fuel consumption during deceleration is 0. In addition, it is found that the larger the amplitude of speed fluctuation is, the lower the average speed (i.e., the longer the travel time), but the more fuel-saving can be achieved. The PnG techniques developed in literature[45][18][17] require sophisticated models, and are usually applied on highways. Our RL controller is purely data driven and can be used in urban traffics.

In summary, there are essentially two strategies learned by the DDPG to save fuel: 1) decelerate earlier before the traffic signal if cannot pass the intersection in current green phase, and 2) use the PnG method to reduce fuel consumption while cruising and car-following. These two strategies are consistent with the fuel-saving operations found by model-based methods as well as learning-based methods in literature. Related studies usually explore these two strategies separately under specific corresponding scenarios, while our method can integrate these two strategies in one pure data driven framework.

2.4.2 Multiple lanes Eco-Driving

The experiment settings of multi-lane Eco-Driving are similar to those used in the single-lane experiment described above, with the extension that each road is now three lanes. The total volume is tripled to 1200 veh/h. Each vehicle enters the corridor from the "best lane" defined by SUMO, and can change lanes based on traffic conditions. In order to evaluate the control performance under unusual traffic conditions, we generate two such cases at the second and fifth road segments (i.e., the two longest segments). When the ego vehicle drives along one of those two segments, if the speed of the ego vehicle is larger than 10m/s and there is a front vehicle on the same lane with speed larger than 10m/s, the front vehicle will stay on the current lane and decelerate to 2m/s within 4 seconds and maintain this speed until exiting the corridor. These ``purposely'' generated scenarios are used to mimic unusual traffic conditions, such as lane drops or traffic accidents that cause the leading vehicles to slow down abruptly. The way how these unusual conditions are generated also imply that for different test cases in this subsection, the locations where this occurs are likely different, instead of fixed at a pre-defined location. This can help test if the algorithms can deal with unusual conditions in general.

An $17 \times 128 \times 3$ neural network with fully connected layers is used to approximate the lateral Q function. The HDQPG Eco-Driving algorithm is tested with the parameter $w_f = 4.5$ and $w_l = 15$, and the DQN-based Eco-Driving algorithm is tested with the parameter $w_f = 5.0$ and $w_l = 15$ after fine-tuning. In order to show the impact of the lateral RL controller, we also test the fourth method: the longitudinal acceleration/deceleration is generated from DDPG and the lateral lane-changing operation is controlled by the SUMO default model. We refer this method as BDDPG (base lateral plus DDPG longitudinal). That is, the proposed HDQPG method and BDDPG differ only in the lateral lane changing control, which can help illustrate the performance of the proposed RL-based lateral control model. Note that we do not test another combination, i.e., SUMO for longitudinal control and RL for lateral control, since the lateral RL control is based on some input (i.e., the critic's value) from the longitudinal RL control.

The detailed results of five test cases are shown in Table 4 and the summary of 100 test cases are shown in Table 5. The first observation from Table 4 is that HDQPG, BDDPG and DQN can all improve the performance under the coordinated signal setting compared with the baseline controller. HDQPG outperforms DQN regarding fuel consumption, but performs slightly worse when it comes to travel time

for the five cases. However, under the non-coordinated signal setting, DQN fails to improve the driving performance for four out of the total five test cases: the fuel consumption and travel time of DQN are much worse than the baseline controller. It is hard for DQN to converge under such multi-lane non-coordinated signal settings, because compared with coordinated signal settings, the number of possible states (under non-coordinated settings) are much larger. Together with the relatively large action space, DQN suffers from inefficient learning. HDQPG, on the other hand, can improve the performance of fuel consumption by up to 35% and 46 %, respectively, under both coordinated and non-coordinated signal settings compared with the baseline controller. One can observe a relatively poor performance of HDQPG under Case 3 in the non-coordinated signal setting, which takes 37% longer time to finish the trip. This is due to the fact that, once the ego vehicle is stopped by a red signal, it has to wait until the signal turns to green. A single stop will make the travel time much longer since the total travel distance of the studied scenario is only 1400 m. We believe that such increment of travel time should be reasonable, especially when fuel consumption can still be reduced by 14%. In addition, under the multi-lane scenario, since there is only one ego vehicle, the following vehicles could change lane to bypass the ego vehicle if they feel that the ego vehicle is too slow. Therefore, the travel time increment only holds for the ego vehicle, while other vehicles usually do not suffer long extra travel times. With 100 test cases in Table 5, similar results can be observed. In particular, HDQPG outperforms the other three algorithms in terms of fuel consumption with much smaller average and smaller (or comparable) standard deviation. For travel times, HDQPG outperforms DQN but is sligtly worse than the baseline algorithm and BDDPG.

		Fuel Consumption (ml)					Travel Time(s)				
	Case	Base	DQN	Imp.	DDPG	Imp.	Base	DQN	Imp.	DDPG	Imp.
Coord.	Case 1	133.4	111.57	16.3%	95.94	28.38%	95	95	0.00%	100	-5.26%
	Case 2	135.94	122.98	9.53%	93.15	31.48%	99	99	0.00%	103	-4.04%
	Case 3	184.46	162.07	12.14%	116.56	36.81%	135	135	0.00%	140	-3.70%
	Case 4	132.37	120.71	8.81%	88.77	32.94%	98	98	0.00%	104	-6.12%
	Case 5	170.16	148.41	12.79%	109.52	35.64%	131	131	0.00%	135	-3.05%

Non-coord.	Case 1	185.48	158.52	14.53%	134.78	27.34%	151	151	0.00%	151	0.00%
	Case 2	198.12	170.86	13.76%	131.66	33.55%	155	155	0.00%	155	0.00%
	Case 3	216.99	214.03	1.36%	144.39	33.46%	155	194	-25.16%	154	0.65%
	Case 4	182.72	169.40	7.29%	118.92	34.91%	152	152	0.00%	152	0.00%
	Case 5	217.36	215.23	0.98%	134.25	38.24%	150	189	-26.00%	149	0.67%

Table 3: Performance of different multi-lane control algorithms under two signal settings

		Fuel Consumption (mL)				Travel time (s)			
		Base	DQN	BDDPG	HDQPG	Base	DQN	BDDPG	HDQPG
Coord.	Avg	175.42	169.46	136.91	120.05	129.97	164.84	129.11	131.76
	Std	44.71	79.32	32.65	29.15	42.39	87.03	40.90	34.90
Non-coord.	Avg	218.36	254.26	154.8	133.87	168.59	236.93	174.67	179.87
	Std	32.49	65.00	26.54	31.28	32.71	75.10	33.77	39.55

Table 4: Performance of different single lane control algorithms under two signal settings (summary of 100 test cases)

In order to further demonstrate how the lateral RL controller improves the performance, we next focus on HDQPG and BDDPG. For BDDPG, the lateral lane-changing behavior is controlled by the default model of SUMO, while the longitudinal acceleration/deceleration is generated by the single lane DDPG. Table 4 shows that both HDQPG and BDDPG can improve the performance of the baseline controller under either the coordinated or the non-coordinated signal setting. Note that in the baseline controller, the ego vehicle can also change lanes to avoid congestion when the front vehicle "purposely" decelerates. The travel times of both HDQPG and BDDPG are similar to those of the baseline controller (except Case 3 under non-coordinated signal, due to the reason discussed above), under either the coordinated or non-coordinated signal setting. In terms of fuel consumption, under both coordinated and non-coordinated signals, HDQPG performs significantly better than BDDPG. Such difference between BDDPG and HDQPG are mainly resulted from the different objectives of these two lateral controllers: the SUMO lane-changing model of BDDPG aims to improve the speed gain, while HDQPG tries to reduce both travel time and fuel consumption.

It is hard to compare in detail the differences between the lane-changing operations of HDQPG and BDDPG. One reason lies in the difficulty of capturing a scenario where all surrounding environments of the ego vehicle under HDQPG and BDDPG are exactly the same. In addition, since HDQPG considers the long-term fuel performance, the lane-changing decision at a specific time might not be the best for just that particular time, which however may help achieve the greater long-term benefit. Here we show a scenario where the surrounding environments of the ego vehicle under HDQPG and BDDPG are almost the same. We qualitatively compare the lane-changing operation differences between HDQPG and BDDPG. Figure 10 shows such a scenario, which happens on the fifth road segment in Case 2 under the non-coordinated signal setting. As mentioned earlier, on this segment, the ego vehicle will encounter a manually created "slow down", i.e., the front vehicle abruptly decreases to a very low speed (2m/s) and maintains that speed until the end of the simulation. In Figure 10, the red solid horizontal lines in the upper left plot are used to indicate the duration of the red signal phase, the dashed cyan and black vertical lines show when the front vehicle starts to abruptly decelerate under BDDPG and HDQPG respectively, and the dashed magenta and yellow lines show when the ego vehicle changes lane under BDDPG and HDQPG respectively. The solid blue and green lines show, under BDDPG and HDQPG respectively, the distance (upper left plot) and speed (upper right plot) of the ego-vehicle, and the distance to the front vehicle (lower left plot) and the speed difference between the front vehicle and the ego vehicle (lower right plot). Note that the trajectories of the ego vehicle under both BDDPG and HDQPG are aligned together in terms of longitudinal travel distances.

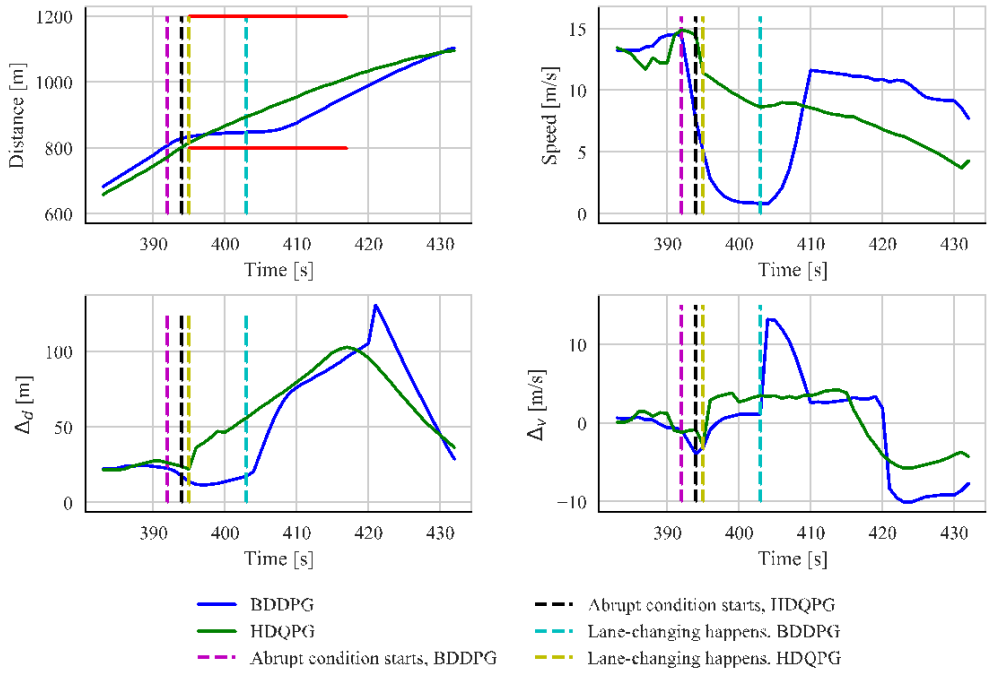

Figure 10: Lane-changing behavior of BDDPG and HDQPG

We can observe that, before this abrupt condition occurs, the ego vehicles under BDDPG and HDQPG drives at a similar environment, i.e., each of them follows the front vehicle with similar speed difference as well as similar distance, which can be shown from the lower two plots in Figure 10. When the front vehicle starts to abruptly decelerate, the ego vehicle under HDQPG changes lane right after the abrupt deceleration starts, which enables it to maintain a relatively high speed to gain more both travel time and fuel consumption benefits. Under BDDPG, the ego vehicle first slows down sharply to a low speed, following the front vehicle; and then after about 11 seconds, the ego vehicle changes lane to gain more speed benefit. The fuel improvements of BDDPG and HDQPG are 28% and 43% respectively compared with the baseline controller, while the travel time improvements of BDDPG and HDQPG are both around -1%. There are two reasons that may help explain the relatively long delay (i.e., 11 seconds) before the ego vehicle makes the lane changing under BDDPG. First, the tactical lane-changing model used in SUMO uses the accumulative speed gain probability to prevent oscillations. Thus, it takes time (11 seconds in this test case) to accumulate the speed gain probability to reach the pre-defined threshold. Second, we cannot guarantee the traffic conditions of adjacent lanes are exactly the same for BDDPG and HDQPG. BDDPG may be prohibited to change lane due to traffic conditions of adjacent lanes even the speed gain probability has reached the threshold. The scenario shown in Figure 10 is part of Case 2 under the non-coordinated signal setting (see Table 4). For other cases, the patterns shown in Figure 10 are common, although it is possible that both BDDPG and HDQPG may decelerate to low speeds first since it is not safe to change lane immediately.

Subsection 2.5 Discussion and conclusion

In this section, we developed a hybrid deep Q-learning and policy gradient (HDQPG) based Eco-Driving model for vehicles driving along signalized corridors under the environment of connected and automated vehicles (CAVs). Both longitudinal acceleration/deceleration and lateral lane-changing operations were considered. The information of surrounding vehicles and upcoming traffic signals were collected by on-board sensors, V2V, and V2I techniques. We also designed specific states, actions, and reward functions for the longitudinal and lateral RL models respectively, and developed a "checking-feedback-learning" (CFL) framework to ensure driving safety and decision consistency by encouraging the RL algorithms to learn safe driving behaviors and consistent decisions.

A deep deterministic policy gradient algorithm was designed to learn the longitudinal operations and a deep Q-learning neural network was developed to make the lane-changing decisions. We tested the proposed HDQPG model on a three-lane five-intersection corridor with different traffic conditions. The results showed that HDQPG outperforms existing RL-based Eco-Driving methods and could learn two basic fuel-saving strategies (i.e., decelerating earlier if the ego vehicle cannot pass the upcoming intersection in current green phase; and using pulse and glide (PnG) operation to make efficient use of

kinetic energy when cruising), which have been proven to be fuel-efficient by model-based methods in the literature. The HDQPG algorithm could also deal with unusual driving conditions like abrupt deceleration of the front vehicle by changing lanes at proper times. Compared with the baseline control method, under both coordinated and non-coordinated signal settings, the fuel performance could be improved 27%-35% and 14%-46%, respectively.

There are several limitations of the proposed HDQPG model, which merits further investigations. First, we only test the proposed model and algorithm on a traffic corridor in simulation, which needs to be further tested using more case studies and ideally on real-world traffic corridors or test beds. In order to implement the proposed algorithm in real-world, a lower level regulator might be needed to smooth the acceleration profile generated by the RL controller and generate the final commands to the actuators (e.g., throttle angle of the engine and angles of the braking/acceleration pedal). Second, we only use the data from five surrounding vehicles of the ego-vehicle to design the RL algorithms. Using data of more vehicles to design more efficient and robust Eco-Driving algorithms is an interesting future research direction once the V2V and V2X techniques become more widely deployed. Third, we did not consider the routing problem in this paper since the focus is on signalized corridors. Designing RL-based algorithms for a transportation network in which the ego vehicle can learn and find the best routes to follow and then the best lanes for left-turn/right-turn/go-straight when approaching an intersection is an interesting and challenging topic. Fourth, the fuel consumption model in SUMO is based on the 3rd degree polynomials over speed and acceleration. Although this polynomial function was fit to detailed engine models based on the data of Handbook Emission Factors for Road Transport (HBEFA), it still may not be very accurate, especially when the lateral operations are considered. Integrating a more accurate fuel consumption model as well as a more accurate vehicle dynamics model is needed in future research.

Section 3: Vehicle in the Loop (VIL) Simulation

Subsection 3.1 Introduction

Today, purely real-world experiments and purely virtual simulation are still the major ways of conducting automotive and transportation related testing [46]. However, testing in real-world is usually done along with high-budget and time-costly pre-processes to build the testing environment [46, 48]. Moreover, testing in purely virtual simulation-based forms may not reflect the real-world conditions [49]. Vehicle in the Loop (VIL) simulation may help address these issues. The concept of VIL combines the advantages of testing in the real world and the flexibility of applying virtual simulation [46]. Specifically, VIL provides a platform for testing new vehicle and traffic control algorithms in a virtual environment with just the key hardware components (i.e., vehicle(s) in this case). By adopting the concept of VIL, environmental conditions can be easily changed. The pre-process of building the testing environment is no longer time consuming, and it can therefore significantly reduce the costs.

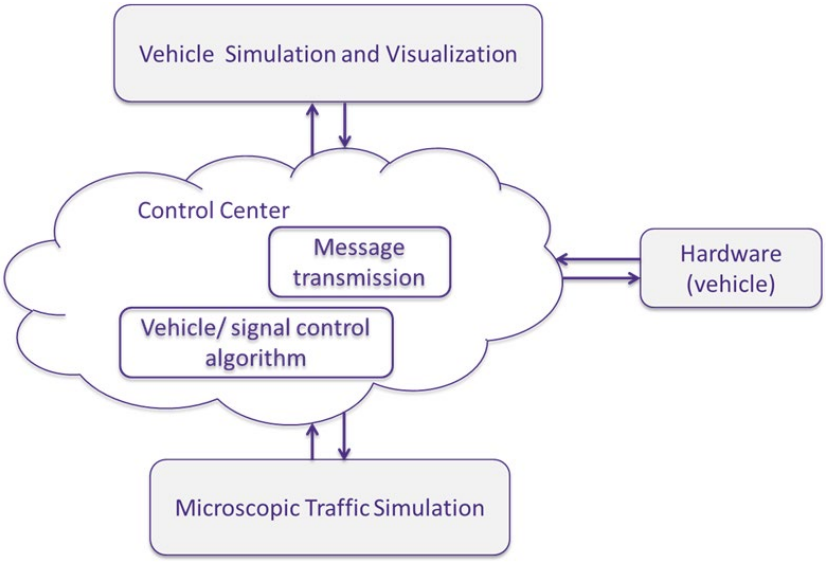

Figure 11: Structure for the VIL platform simulation

Figure 11 shows the basic structure of the VIL simulation. The VIL consists of four main elements: control center, microscopic traffic simulation entity, vehicle simulation and visualization entity, and the real-world hardware (i.e., vehicle) entity.

- The control center plays two important roles in the VIL. First, it manages the communications between the microscopic traffic simulation, vehicle simulation and visualization, and the real-world vehicle. Second, it collects data from the other three entities and generates the control

commands (e.g., traffic signal phases, vehicle acceleration, etc.) by applying the control algorithms that need to be tested.

- The microscopic traffic simulation entity simulates the traffic flows and the traffic signals. From the perspective of a controlled vehicle, the microscopic traffic simulation entity can capture the high-level information of its surrounding vehicles (e.g., relative position, speed, and acceleration), the status of signals (e.g., phases and timings) and the trip information (e.g., path travel times), which can be sent to the control center and be used by the tested control algorithms.
- The vehicle simulation and visualization entity simulates the vehicle-level dynamics, which can provide richer information than the microscopic traffic simulation entity. For example, the microscopic traffic simulation software (e.g., SUMO) usually do not provide image data of the onboard cameras, but vehicle simulation software (e.g., Unity) can. In addition, the detailed simulation makes it possible to better visualize the simulated vehicle.
- The real-world hardware (i.e., vehicle) entity is the controlled subject. It receives commands from the control center and implement them in real-world. Meanwhile, it sends its status information to the control center, which will be used by the tested control algorithms as well as mapped to the microscopic traffic simulation and the vehicle simulation and visualization entity.

In the following of this Section 3, we show the development details of the VIL.

Subsection 3.2 Methodology

In this section, we first show the framework of VIL in detail. Then, we show how to build each module. Finally, we present an example of the VIL.

3.2.1 VIL framework

Figure 12 shows the detailed flowchart of the VIL. The control center manages the communications between the microscopic traffic simulation, vehicle simulation and visualization, and the real-world vehicle. The traffic signal information, network traffic information (e.g., path travel times), and the high-level vehicle information (e.g., position/speed/acceleration of surrounding vehicles) are collected from the microscopic traffic simulation entity. The detailed vehicle information (e.g., engine status, torque force, frictions against ground floor, tires status) and sensor data (e.g., onboard cameras) are collected from the vehicle simulation and visualization entity. These data can be used by two kinds of control algorithms integrated in the control center: the signal control algorithms and the vehicle control algorithms. The tested signal control algorithms generate the optimal signal phases/timings in real-time or for a forward time horizon, which are sent to the microscopic traffic simulation entity and implemented. The tested vehicle control algorithms generate the corresponding actions (e.g., acceleration, lane-changing decision), which are sent to the real-world vehicle. The real-world vehicle

implements those actions and transit to next status. Such status is then collected by the control center and be sent to microscopic traffic simulation and vehicle simulation and visualization entities to reflect the actual status of the controlled vehicle in those simulations. Meanwhile, the visualization function of the vehicle simulation and visualization entity can display all the information including the real vehicle details, real vehicle actions, and the microscopic traffic simulation results. After finishing all the visualization at the current time step, the vehicle simulation and visualization entity will send a ready message to the control center. This triggers the next iteration of the VIL simulation.

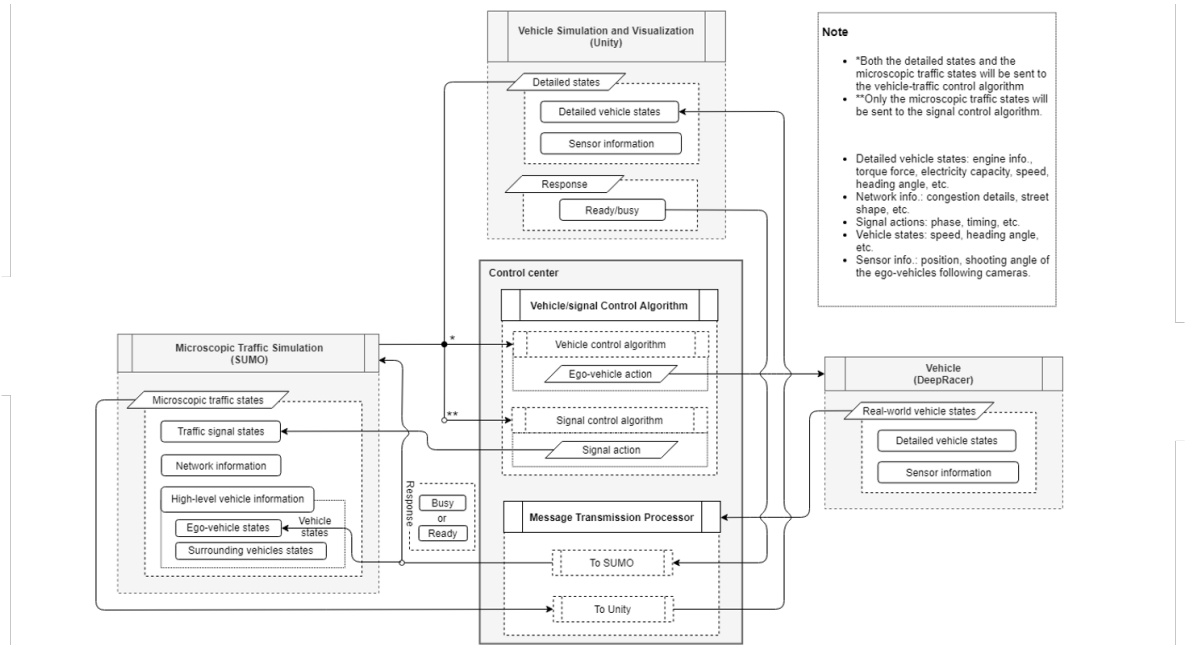

Figure 12: Detailed Flowchart of the VIL Simulation

In this study, we use i) Unity 3D [50] as the vehicle simulation and visualization module; ii) SUMO as the microscopic traffic simulation module; and iii) a fully autonomous scaling-down race car, the Amazon Web Services (AWS) DeepRacer [51], as the model of the real-world vehicle. Figure 13 shows SUMO, Unity 3D, and AWS DeepRacer used in our design. Unity is a powerful game engine that can be used to develop 2D and/or 3D dynamic environments (e.g., video games and vehicles) interacting with other platforms. SUMO is an open-source microscopic traffic simulator used to simulate, analyze, and evaluate traffic systems. In addition, it provides an interface library, i.e., Traffic Control Interface (TraCI), allowing users to interact with SUMO in real time with programming languages (i.e., Python, C++, etc.). AWS DeepRacer is a 1/18th-scale fully autonomous race car controlled by the AWS DeepRacer web panel.

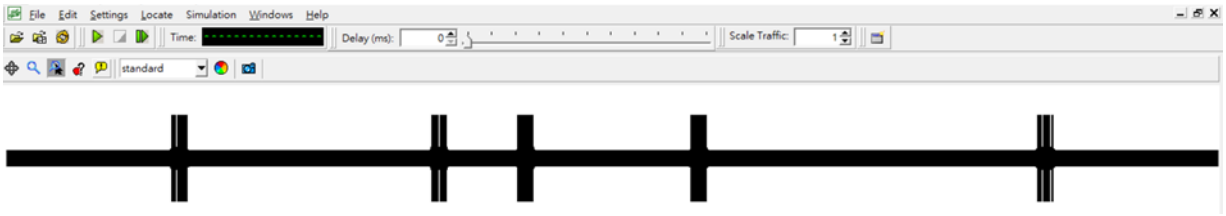

Figure 13a: SUMO Simulation Environment

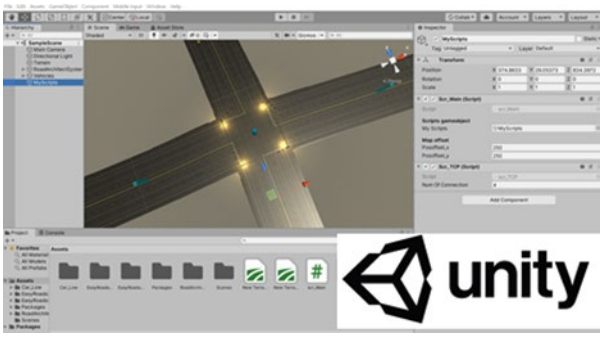

Figure 13b: Unity 3D Game Engine

Figure 13c: AWS DeepRacer

Figure 13: Tools Used in VIL Simulation Experiment

3.2.2 Assumptions

Developing the entire VIL shown in Figure 12 under different traffic scenarios is difficult. To reduce the complexity and meanwhile maintain the key features of VIL, we make several simplifications and assumptions:

- The vehicle simulation and visualization module (i.e., Unity 3D) is only used for visualization. Nevertheless, it can be extended to simulate the detailed vehicle dynamics.
- Passenger vehicles are the only type of vehicles considered in the VIL platform. Three types of passenger vehicles are considered: human-driven vehicles, CAVs, and CAVs under control. The network information only focuses on the type of road. Other information (e.g., path travel times) of the microscopic traffic simulation module (i.e., SUMO) is out of the interest.
- There are only two types of "road systems": straight roads with three lanes and signalized intersections with four legs, twelve movements, and three lanes. Under this simplification, we designed a modular method to efficiently generate road systems in Unity. We build to modules (i.e., an extremely long straight road module and an intersection module) and store them in the Unity library. When we try to visualize the interested target vehicle, we will first check (in SUMO) which road system the target vehicle locates at and how long it is from the current position to the next road system. Then, if the target vehicle is far away from next road system,

e.g., the target vehicle drives along a straight road and there is a long way to arrive next intersection, we will keep using the current road system (i.e., straight road module) in Unity. If the target vehicle is approaching a next different road system and the distance is less than a threshold, e.g., the target vehicle drives along a straight road and it is approaching an intersection, we will connect the next road system to the current road system in the threshold distance. In this way, we can flexibly deal with different road lengths instead of manually building every straight road model.

Based on above assumptions, the simplified VIL flowchart is shown in Figure 14.

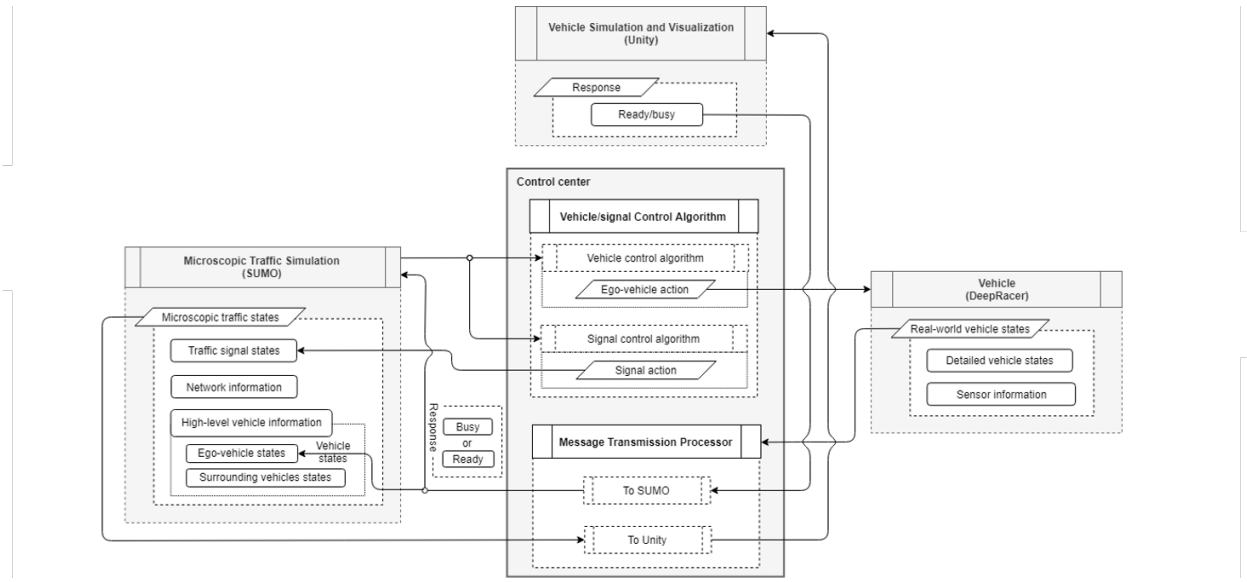

Figure 14: Detailed Flowchart of the Simplified VIL Simulation

In the following of Section 3.2, we show the development of the four key components: the control center, the microscopic traffic simulation module (i.e., SUMO), the vehicle simulation and visualization module (i.e., Unity 3D), the real-world vehicle (i.e., AWS DeepRacer).

3.2.3 Control Center

The major steps in the control center are shown in Figure 15. First, it starts with launching the control center at the specific IP and port. If the launching is successful, the control center starts to listen for the other three clients, SUMO, Unity, and DeepRacer, to join the IP and port. If the other three clients successfully join the IP and port, the control center checks the step number in SUMO. If the step number in SUMO is the initial step, the control center will request the initial traffic information from SUMO (by getBeginSUMO). Otherwise, the control center will first check whether Unity is ready to take another message (by getResponseFmUnity). If Unity is still working on the last message, the control center will

wait until the Unity work is done. Then, the control center will request the current traffic information from SUMO (by getNextSUMO). Then, the vehicle control algorithm and signal control algorithm in the control center will generate the actions for the DeepRacer and the signals in SUMO using the collected data. The actions generated by the vehicle control algorithm will be sent to the DeepRacer (by msgToDeepRacer). The DeepRacer conducts the action and sends its updated status to SUMO (by msgToSUMO). After receiving the new information from DeepRacer, SUMO will move one step forward. Meanwhile, this process is also visualized by Unity. The control center then starts a new loop.

The details of functions checkStepSUMO, getBeginSUMO, getResponseFmUnity, and getNextSUMO are listed in Appendix A. It is noted that all the communications between the three parts are based on Python3.8 Transmission Control Protocol (TCP) and the Internet Protocol (IP), i.e., the TCP/IP protocol.

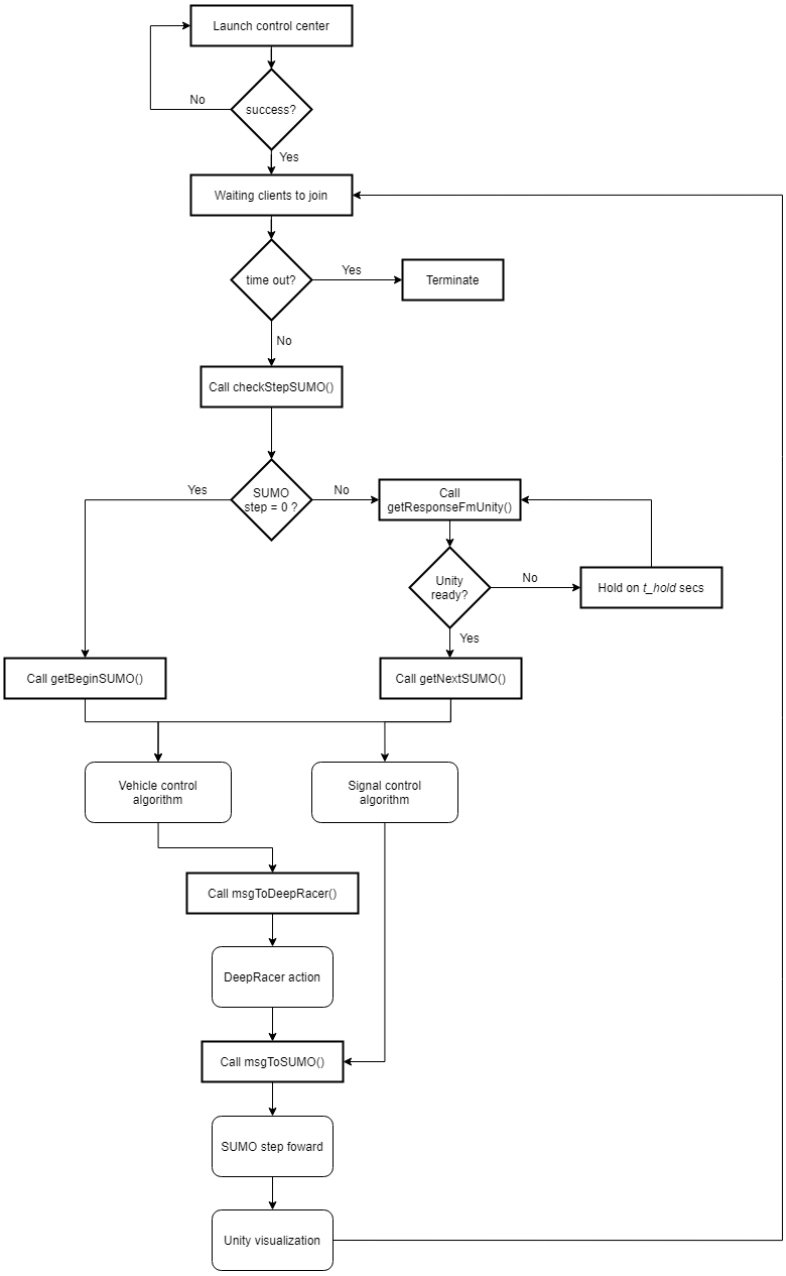

Figure 15: Detailed Flowchart of Control Center

3.2.4 The microscopic traffic simulation module: SUMO

Since the control center requires different types of information from SUMO, several classes are designed in SUMO to handle the details of these requests. They are class vehicle, class network, and class signal:

- vehicle: stores the information of route, road, vehicle length, vehicle width, vehicle position, speed, heading, whether it is using brake pedal

- network: stores the information of edges, end nodes of each edge, node type, and lane number
- signal: stores signal phase information piece, including red light, yellow light, and green light

When the control center calls the function getBeginSUMO or getNextSUMO, SUMO will check the type of the current road system and the incoming road system for the target vehicle, which, as discussed in Section 3.2.2, is used to help generate the visualization environment in Unity by the modular method. To check the road system type in SUMO, we first collect the information of the road system where the target vehicle locates at. If the target vehicle is on a straight road, we will check the end node of that road to see if there is a "traffic_light" in that node. If the target vehicle is on an intersection, we will retrieve the route of the target vehicle and get the next straight road that the target vehicle will go. These information are then sent to the control center to help generate the visualization environment in Unity by the modular method.

3.2.5 The vehicle visualization module: Unity

The major steps in Unity can be seen in Figure 16, which begin with launching Unity and then try to let Unity join the same IP and port of the control center. Once the Unity is successfully launched, it will send a response to indicate it is ready to take the next message from SUMO. After sending the ready response, Unity starts to receive the message. Before working on the received message, Unity will check whether the received message is a new message. If the message is the same as the last message, Unity will ignore it and get ready to receive the next new message. If the received message is the new one, Unity will start visualizing the information of that message.

To visualize the message, as shown in Figure 16, Unity splits the received text into three parts: network, traffic light, and vehicle. The message is processed by using the function CookNetwork, CookTrafficLight, and CookVehicle. The network processor CookNetwork in Unity is responsible for duplicating either the intersection module or the straight module to the corresponding location in the SUMO simulation. Which module shall be duplicated is based on the type of road system. The pre-built road systems (i.e., straight road and intersection) are shown in Figure 17. For the straight road system, using a three-lane road as an example, we apply white dashed lines to separate the lanes and the double yellow line to divide the two directions. Once the number of displaying road systems is more than a threshold number, Unity will remove the very first road system to retain its computing and memory capacity. In the traffic light processor CookTrafficLight, the traffic signal sets are sorted into a clockwise order from the northern side. The sorting is based on each of the traffic signals' location. As an example, there are twelve signals in total if the road system has three lanes at each leg, three signals is a set which has one location information, and the traffic light processor will sort them into a clockwise order shown Figure 18. Next, Unity will assign each signal phase set to the corresponding traffic light set. As for the vehicle processor CookVehicle, it splits the vehicle message into existing vehicle information, left vehicle

information, and the newly arrived vehicle information. This processor accordingly removes the left vehicles, creates the arrived vehicles, and moves the existing vehicles to the latest locations with their respective heading angles. Finally, a camera is set to track the target vehicle. The camera keeps a distance of one and a half vehicle-length away from the target vehicle with a distance of one and a half vehicle-height above ground. In addition, the camera is rotated 10 degrees downward from the horizontal line, and it keeps its view toward the tracked vehicle while displaying. The processes of the functions CookNetwork, CookTrafficLight, and CookVehicle are also listed in Appendix A.

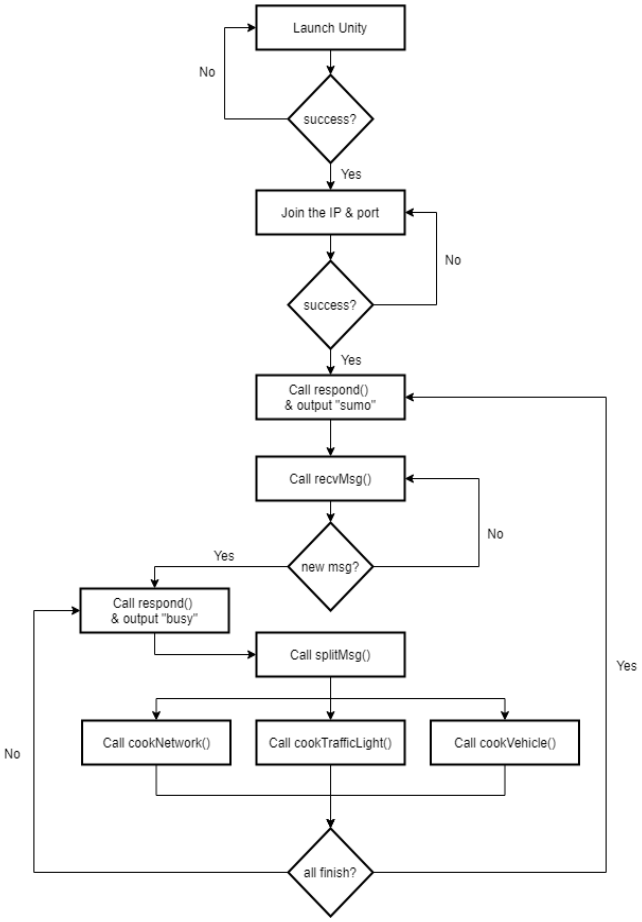

Figure 16: Detailed Flowchart of Unity

Figure 17a) Intersection module

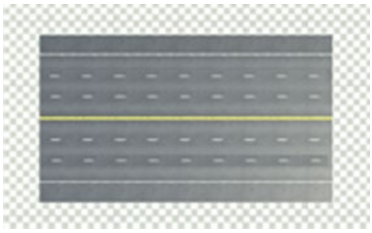

Figure 17b) Straight module

Figure 17: Pre-built Modules in Unity

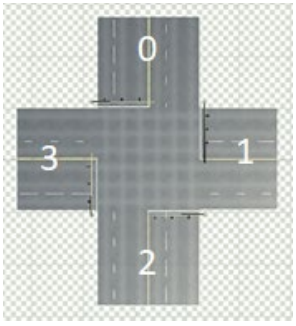

Figure 18: Traffic Signal Sets Sorting

3.2.6 DeepRacer

DeepRacer is controlled by the panel shown in Figure 19 in the Amazon DeepRacer console website. The control steps are shown in Figure 20. After joining the same IP and port in the control center, it tries to log into the Amazon DeepRacer console web using the pre-assigned DeepRacer account and password. Then, the action message is received by the function `recvMsg` and is processed by the function `controlDR`. In the function `controlDR`, the received speed will be normalized to the range -1.0 to 1.0. Similarly, the heading angle will be normalized to the same range. According to the calibration of our DeepRacer, the real speed range falls in 0.0 to 2.1 m/s, and the heading angle range falls in -21 to 22 degrees from the vertical line. The function will send the speed and heading angle commands to the panel to control the DeepRacer. The processes of the functions `recvMsg`, `controlDR`, and `sendMsg` can be seen in Appendix A.

Figure 19: DeepRacer Control Panel

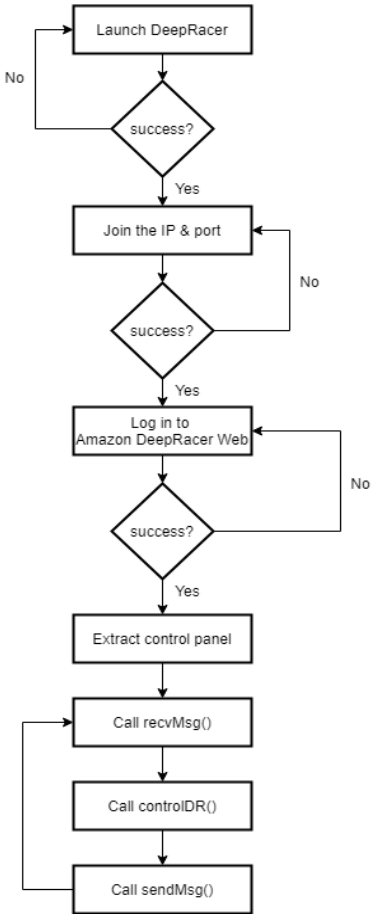

Figure 20: Detailed Steps in DeepRacer

Subsection 3.3: VIL Experiment

3.3.1 Experiment settings

We designed a five-intersection corridor, which is also used in the Eco-Driving algorithm in Section 2, to test the VIL platform. The network is shown in Figure 21. Every road segment has three lanes for each direction. The length of the horizontal straight road from West to East is, respectively, 200.0, 300.0, 100.0, 200.0, 400.0, and 200.0 meters. The North-South road segments have the same length of 50.0 meters. Vehicles traveling on the East-West horizontal roads are only allowed to go straight at the intersections (i.e., no left turns or right turns are allowed). The same restrictions also apply to southbound vehicles on the North-South vertical roads. On the other hand, the northbound vehicles on the North-South vertical roads are allowed to go straight or turn right (no left turns). Vehicles are also allowed to take a U-turn by the end of any road segment in this network.

3.3.2 Results and Discussion

Figure 21 (b) shows the connection between SUMO, Unity 3D, and DeepRacer. With the camera following setting in Unity 3D, it can be easily observed that the traffic system changes in real-time. More than one perspective in this simulation can be observed by setting up multiple camera angles in Unity 3D.

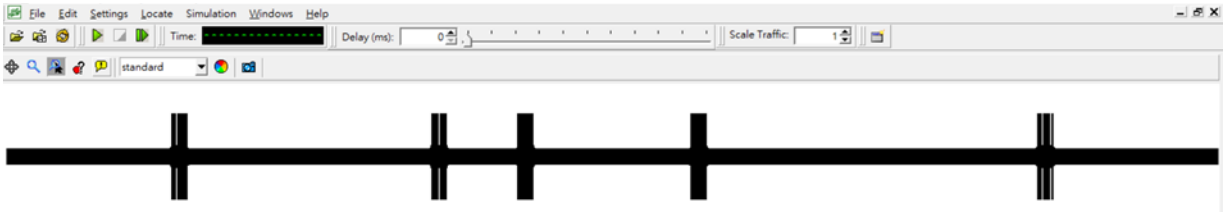

Figure 21a: 3rd network environment in SUMO

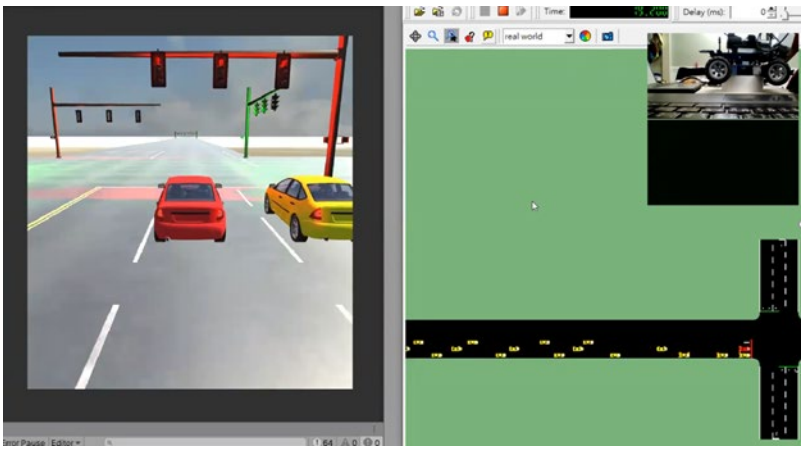

Figure 21b: VIL simulation by applying 3rd network environment

Figure 21: Real-time Simulation and Visualization Results of the 3rd Network Environment

The advantages of the VIL environment (i.e., the integration of SUMO, Unity 3D, and DeepRacer in this project), can be summarized as: (1) it easy to observe the same scenario from multiple perspectives by putting multiple cameras in the visualization module (i.e., Unity), and (2) the visualization module (e.g., the Unity 3D) may have the ability to help detect infeasible vehicle motions that might not be easily discovered (and thus accepted) in traffic simulation. For example, a vehicle cannot move from position X to Y in a short period of time if the distance between X and Y is large enough. In SUMO, however, this unrealistic movement may be simulated by using the function `moveToXY`. The physical environment in Unity 3D needs to be set up before conducting the simulation, and as a result, this unrealistic movement can be easily caught in Unity and will be displayed as an error in the visualization system.

Although the framework of the VIL platform has been successfully developed, the implementation of the VIL remains an open field. For examples, due to the space limitation, we did not build the real testing field (i.e., the five-intersection corridor) in real-world. Instead, the target vehicle is "running in the air". We did not build the detailed vehicle model in Unity, due to which the Unity is only used for visualization. In fact, Unity can simulate microscopic vehicle dynamics such as engines, transmission system and tire dynamics, which can make the platform more practical. In addition, the onboard sensor data like image from camera can help design better vehicle and traffic control algorithms. These topics are left for future improvement of the VIL platform and further studies of the vehicle/traffic control algorithms.

Section 4: Integrative Vehicle-Signal Control with VIL Simulation

In this Chapter, we test a vehicle-signal control algorithm that integrates a vehicle control algorithm and a signal control algorithm. The vehicle control algorithm is the Eco-driving algorithm presented in Section 2. The signal control algorithm is described in Section 4.1. In the following of this section, we first introduce the signal control algorithm, and show the experiment results of this integrated vehicle-signal control method. Note that in this section we do not include DeepRacer, only SUMO (for traffic simulation) and Unity (for visualization) are used.

Subsection 4.1 The dynamic signal control algorithm

According to the HCM2000 (Highway Capacity Manual 2000), the fixed-time signal control method is computed based on the hourly maximum traffic volume throughout a day to satisfy the peak-hour situation. For each signal phase, the green time duration is designed to accommodate the maximum traffic volume. Both the phase plan and green duration are fixed in a fixed-time signal control, which may cause excessive waiting time when the traffic flow varies dramatically within an entire day.

The alternative control option is actuated signal control, which determines the phases and green time duration based on the actual traffic volumes from all incoming approaches of an intersection. In the actuated signal, three essential parameters are needed for designing the green duration: the minimum green interval, the maximum green interval, and the extended interval. Sensors, such as loop detectors, are also needed to detect traffic volume information in real-time. Under this control strategy, each signal phase has a minimum green interval and accumulated extended interval (i.e., when detectors detect a coming vehicle, the green time duration will increase by a unit of the passage time) and terminates when the duration exceeds the maximum green interval. The traffic delay issue can be mitigated to certain extent using this control method. It is nevertheless detector-sensitive, which means that the performance of the signal control depends heavily on the availability and quality of detector data.

In this research, a dynamic signal control algorithm is proposed to address both traffic volume fluctuation and detector-dependency problems. Using the trajectory data of limited CAVs, the dynamic signal control method can provide the HCM2000-based signal timing plan every 15 minutes, which can be then updated based on the most recent traffic information from CAVs. The overall process of operating dynamic signal control consists of four steps in each period (every 15 minutes): penetration estimation, traffic volume estimation, signal phase determination, and signal phase duration calculation. Assuming that the fluctuation of traffic volumes between adjacent time periods is relatively small in the real-world scenario, the dynamic signal control applies the estimated traffic volume in the current time period to calculate the signal phase plan for the next time period.

4.1.1 Penetration rate estimation

The first step of the dynamic signal control algorithm is to estimate the penetration rate of the limited CAVs under the mixed traffic stream of human driving vehicles and CAVs. The single-source data penetration rate[52] (SSDPR) estimation method is adopted in this step, which is a non-parametric and unbiased method for estimating penetration rate solely based on CAV data. SSDR is also applicable to overflow or oversaturated conditions. Using SSDR, the penetration rate of each lane is computed based on the CVs data collected via SUMO and is used as an input to the traffic volume estimation step.

4.1.2 Traffic volume estimation

The traffic volume of each incoming approach is computed as the volume of the CAVs of a certain approach divided by the average penetration rate of the corresponding approach. Since the given CAVs are generated randomly, the traffic volume may fluctuate over time. Thus, the estimated traffic volume may differ from the true value especially when the penetration of CAVs is low (i.e., less than 30%). To this end, the Kalman filter (i.e., linear quadratic estimation) is applied to smooth the estimated result so as to minimize the difference between the estimated result and the true value. The Kalman filter covariance is set as 1 in the testing to optimize the estimation process.

4.1.3 Signal phase plan determination

The signal phase plan will be chosen among three main signal phase plans: the basic phase plan (i.e., left-turn phase and straight/right turn phase in North/South and West/East direction), the lead/lag phase plan, and the overlapping phase plan. These three phase plans are shown in Figure 22 – Figure 24.

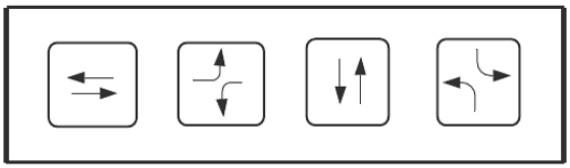

Figure 22: The Basic Phase Plan

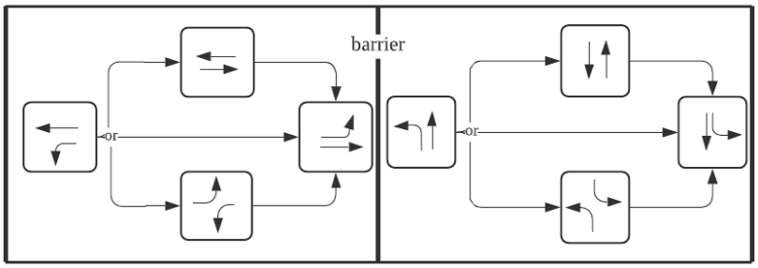

Figure 23: Lead/lag Phase Plan

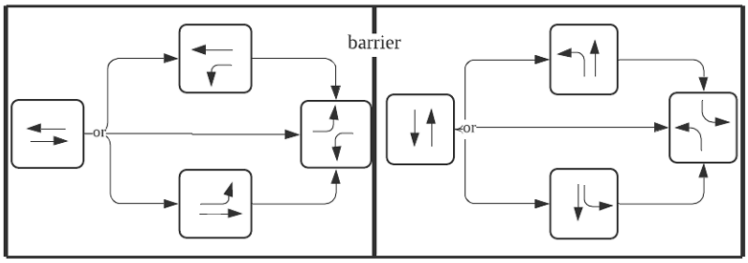

Figure 24: Overlapping Phase Plan

Different phase plans are suitable for different traffic flow patterns. Phase plan options provided in this dynamic algorithm cover most of the possible traffic flow patterns in urban signalized intersections: symmetrical traffic flow on each incoming approach, and relatively large traffic flow on the main street and low volume on the minor street.

4.1.4 Signal phase plan selection and phase duration calculation

With the estimated traffic volume computed in step 2 above, the cycle length and green time of each plan are calculated for each of the three phase plans discussed above using the method in HCM2000. In particular, the green duration of each phase is computed in terms of the ratio of its corresponding v/s (i.e., volume/saturation flow rate) ratio. Next, the average delay of each phase plan is calculated by the HCM2000 proposed methods and the optimal phase plan is selected as the one with the minimum average delay. The dynamic signal control addresses the detector-dependency problem by estimating the traffic volume solely using the CVs data and can better adapts the fluctuations of traffic volumes in real-time by providing the optimal signal timing plan every 15 minutes.

Subsection 4.2 Testing Results and Discussions

We test the integrated vehicle-signal control method with the same corridor used in Section 2 and Section 3. Since the Eco-Driving algorithm is designed for a single vehicle, at a specific time instant, only one CAV in the network is allowed to be controlled. To do this, we randomly pick one of the CAVs that just enter the corridor as the controlled vehicle, and only control this vehicle by the Eco-Driving algorithm until it exits the corridor. Once the controlled vehicle exits the corridor, we then randomly pick one of the newly arrived CAVs as the next controlled vehicle. In this way, the Markov property is not violated and the Eco-Driving algorithm can be applied. Meanwhile, the dynamic HCM signal control algorithm collects the data of all CAVs to calculate the signal timing/phases every 15 minutes. Different CAV penetrations, i.e., 5%, 10%, 30%, 60%, and 100%, were tested. The volume of the main corridor road is set in a periodic pattern: during the first 2 hours, the volume increases linearly from 900 veh/h to 1200 veh/h; in the next 2 hours, the volume decreases linearly from 1200 veh/h to 900 veh/h; during the next 4-hour period, same volumes patterns are used.

We compare the performance of such an integrated vehicle-signal control method with SUMO default vehicle control model (as discussed in Section 2) and pre-timed signal plans. To design the pre-timed signal plans, we assume that the peak hour volumes of the corridor network are known and calculate the corresponding signal plans for each intersection by the HCM method. Such pre-timed signal plans are implemented during the entire simulation period. For each CAV penetration, we train the integrated vehicle-signal control algorithm for 10 hours and collect data of the following 2 hours to evaluate its

performance. We use the average fuel consumptions of controlled CAVs, average travel times of the controlled CAVs, and average delay of all vehicles (controlled CAVs, uncontrolled CAVs, and human-driven vehicles) as the performance measurement, where the first two indexes measure the performance of the Eco-Driving algorithm, and the last index indicates the performance of the dynamic HCM signal control method. The SUMO-Unity connection is shown in Figure 25 and the experiment results are shown in Table 6.

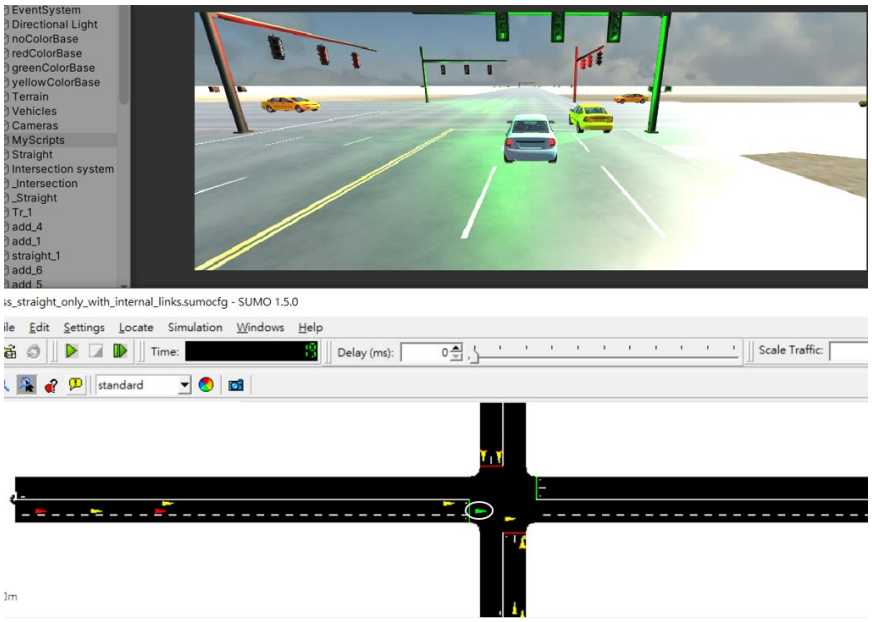

Figure 25: Vehicle-traffic control (Scenario I) in VIL Simulation

Penetration		5%	10%	30%	60%	100%
Width control	Avg. fuel (g)	129.65	134.18	124.96	125.97	115.70
	Avg. time (s)	167.40	159.30	158.20	159.60	151.0
	Avg. delay (s)	22.59	22.73	22.67	23.03	23.14
Baseline	Avg. fuel (g)	133.38				

Avg. time (s)	121.80
Avg. delay (s)	44.17

Table 6: Performance of the integrated vehicle-signal control algorithm and the baseline

In Figure 25, the green vehicle (marked with a circle in SUMO) is the selected controlled CAV, the red vehicles are uncontrolled CAVs, and the yellow vehicles are human-driven vehicles. In this experiment, there is only one green vehicle in the entire corridor. Figure 25 shows that Unity can successfully connect with SUMO and depict the traffic conditions in SUMO with 3-D view.

Table 6 shows the performances of the integrated vehicle-signal control algorithm and the baseline. The average delay of all vehicles is 44.17s for the baseline case and 22.59s to 23.14s for the integrated vehicle-signal control. This shows that the dynamic HCM signal control method can dramatically improve the corridor performance. The dynamic HCM method changes the signal plan dynamically based on current traffic volumes thus can better reflect the real-time traffic conditions.

When the pre-timed signal plans are used and there is no Eco-Driving control, in average, connected vehicles (i.e. CAVs) consume 133.38g fuel and take 121.80 seconds to complete their trips to cross the entire corridor. Under the integrated vehicle-signal control, the travel times of the controlled vehicles are always greater than those of the baseline under every penetration scenario. This is due to the fact that, as discussed in Section 2, the Eco-Driving algorithm tends to decelerate the vehicle earlier if the algorithm predicts that controlled vehicle cannot pass the next intersection within current green phase. In addition, one stop at an intersection can lead to large increment of the travel time since the vehicle has to wait for an entire signal cycle to be able to move again. Under the dynamic HCM signal control, the signal plans change every 15 minutes, which makes it hard for the Eco-Driving algorithm to precisely predict whether or not the controlled vehicle can pass the next intersection in current green phase. Once such prediction is not accurate, the travel time could be largely increased.

The average fuel consumption of the controlled vehicles varies among different CAV penetrations. Most cases (i.e., 5%, 30%, 60%, 100%) show a decrement while one case (i.e., 10%) shows an increment. This is due to the same reason discussed above. The original Eco-Driving algorithm (as shown in Section 2) is designed under pre-timed signal plan, which might become unstable under the dynamic HCM signal control method. Another observation is that the fuel improvements, even the largest case (i.e., 100% CAV penetration, fuel consumption reduces from 133.38g to 115.70g), is smaller than those observed from Section 2 (i.e., from 175.42g to 120.05g). In fact, the fuel consumption of the baseline case here is smaller than the one in Section 2. There are two reasons for this observation. First, we added to

"unusual" traffic conditions (i.e., the front vehicle suddenly decelerate, see Section 2.4.2) in the experiment of original Eco-Driving algorithm, which will increase the travel time as well as fuel consumption of the controlled vehicles. Second, the volume used in this section varies from 900 veh/h to 1200 veh/h and the results shown in Table 6 are the average performance.

In summary, the integrated signal-vehicle control can improve the overall performance of the corridor by reducing the delays. However, those improvements are mainly contributed by the dynamic HCM signal control method. The travel time of the vehicles controlled by the Eco-Driving algorithm is even greater than the baseline case. Nevertheless, the Eco-Driving algorithm can reduce the fuel consumption of the controlled vehicles. It should be noted that the integration of the Eco-Driving algorithm and the dynamic HCM signal control remains to be improved. As discussed above, the dynamic HCM signal control might reduce the performance of the Eco-Driving algorithm. The main reason is that there is no real-time cooperation between those two control algorithms. This will be an interesting research direction in the future. In addition, we only control one vehicle in the corridor to ensure the Markov property of the Eco-Driving algorithm, which is not efficient enough to improve the fuel performance of all CAVs. We have tried the method of simply apply the Eco-Driving algorithm to multiple vehicles, and the results showed that the performance is much worse than the baseline. The main reason is that there is no communication among the controlled vehicles and the Markov property is violated for a specific controlled vehicle. Multi-agent RL method may solve this problem, which is another future research direction.

Section 5: Conclusions

Subsection 5.1: Summary of research activities and findings

This project presented the investigation efforts and results related to vehicle/traffic control under limited penetration of low-level connected and autonomous vehicles (LCAVs). The investigation includes three major parts: i) the Eco-Driving algorithm for a single CAV with low-level automation; ii) the vehicle in the loop (VIL) simulation platform; and iii) the integrated vehicle/traffic control algorithm tested under the VIL.

First, this project developed a hybrid deep Q-learning and policy gradient (HDQPG) based Eco-Driving algorithm for single CAV driving along signalized corridors with low-level automation. The information of surrounding vehicles and upcoming traffic signals were assumed to be collected by on-board sensors, V2V, and V2I techniques. A deep deterministic policy gradient algorithm was designed to learn the longitudinal operations and a deep Q-learning neural network was developed to make the lane-changing decisions. We designed specific states, actions, and reward functions for the longitudinal and lateral RL

models respectively and developed a "checking-feedback-learning" (CFL) framework to ensure driving safety and decision consistency by encouraging the RL algorithms to learn safe driving behaviors and consistent decisions. We tested the proposed HDQPG model on a three-lane five-intersection corridor with different traffic conditions. The results showed that HDQPG outperforms existing RL-based Eco-Driving methods and could learn two basic fuel-saving strategies (i.e., decelerating earlier if the ego vehicle cannot pass the upcoming intersection in current green phase; and using pulse and glide (PnG) operation to make efficient use of kinetic energy when cruising), which have been proven to be fuel-efficient by model-based methods in the literature. The HDQPG algorithm could also deal with unusual driving conditions like abrupt deceleration of the front vehicle by changing lanes at proper times. Compared with the baseline control method, under both coordinated and non-coordinated signal settings, the fuel performance could be improved 27%-35% and 14%-46%, respectively.

Second, we developed a VIL platform to reduce the costs of testing algorithms in real-world. Four key components of VIL were developed. The control center was designed to manage the communications among different simulation/hardware entities as well as to collect data and run the tested algorithms to generate commands to different entities. The microscopic traffic simulation entity (SUMO in this project) was designed to simulate the traffic flows and signals, which could provide high-level information of vehicles (e.g., position, speed and acceleration), the status of signals, and network information (e.g., path travel times). The vehicle simulation/visualization entity (Unity in this project) simulated the vehicle-level dynamics and provided detailed vehicle information (e.g., engine torque, tires status) and onboard sensor data (e.g., camera data). The real-world hardware (an AWS DeepRacer car in this project) is deployed to receive and conduct the commands generated by the tested algorithms. The VIL platform was tested under two traffic environments. The results show that the platform can help test a real-world hardware. We were able to easily deploy a new traffic environment if the road modules have been included in the vehicle simulation/visualization simulation model, and the model also allows us to track a vehicle to observe the traffic system at a microscopic to sub-microscopic perspective.

Third, we developed a dynamic HCM method to control the signals under the environment of LCAVs. The method would first estimate the CAV penetration and calculate the total traffic volume based on the estimated CAV penetration and the CAV volume. Then, given the volume data, the method would calculate the signal phases/timings based on the HCM method, which would be used for the next forward time horizon (e.g., 15 minutes). In addition, we combined this dynamic HCM method with the Eco-Driving algorithm mentioned above as a vehicle-traffic integrated control method, and tested it using the VIL platform. The results showed that the integrated method could reduce the fuel consumption of the controlled vehicle and reduce delays of all vehiclesof the corridor. However, the performance will decrease dramatically if we directly apply it for multiple vehicles.

Subsection 5.2: Future research directions

There are several future research directions that can improve the current Eco-Driving algorithm, the VIL platform, and the integrated vehicle-traffic control method. For the Eco-Driving problem, we designed the HDQPG algorithm only for a single vehicle. Extending such algorithm to deal with multi-agent Eco-Driving problem is an interesting topic. To do this, multi-agent RL structures especially the communication topology and the learning mechanism should be well designed. For the VIL platform, we have successfully developed the framework, while many extensions could be done based on the current framework. The first extension is that we can build a detailed vehicle model in Unity that considers the dynamics of engine, transmission system, tire, suspension system, etc. In this way, the simulation of a single vehicle could be more realistic. The second extension is to develop onboard sensor modules in Unity to simulate the sensors equipped on CAVs, e.g., camera, radar and lidar. These data can be used to test many other CAV related techniques like computer vision (CV) algorithms and Simultaneous localization and mapping (SLAM). The third extension is to develop different modules for different types of vehicles. In this project, we only used the passenger car module. Other types like bus, truck, and bicycle can also be developed based on specific requirements. Fourth, with the capability of simulating network-wide traffic, SUMO can provide network-level information like path travel times. This feature makes it possible to test network flow management algorithms like navigation and ride-hailing algorithms. The current VIL platform serves as the blueprint of various potential applications, which will be implemented in the future. Lastly, the integrated vehicle-traffic control method can be enhanced by adding coordinating between signal control and vehicle control directly, probably under the RL framework. This is expected to further improve the control performances.

References

- Hamilton, A., Waterson, B., Cherrett, T., Robinson, A., Snell, I., 2013. The evolution of urban traffic control: changing policy and technology. Transportation planning and technology 36, 24–43.
- Li, L., Wang, F.-Y., 2018. A review of past 100-year and perspective of next 50-year development of ground traffic control, Automatica Sinica, 44 (4), 577-583 (in Chinese).
- Mahmassani, H.S., 2016. 50th anniversary invited article—autonomous vehicles and connected vehicle systems: Flow and operations considerations. Transportation Science 50, 1140–1162.
- NHTSA, Automated Vehicles for Safety, 2016. https://www.nhtsa.gov/technology-innovation/automated-vehicles-safety#issue-road-self-driving
- Li, W., Ban, X., 2017. Traffic signal optimization under the connected vehicle environment. In Proceedings of the Intelligent Vehicles Symposium, Los Angeles, CA.

- Li, W., Ban, X., 2018a. Connected vehicle based traffic signal optimization. IEEE Transactions on Intelligent Transportation Systems, accepted.
- Li, W., Ban, X., 2018b. Connected vehicle based traffic signal coordination. Submitted to Engineering.
- Xu, B., Ban, X., Bian, Y., Wang, J., Li, K., 2017. V2I based Cooperation between Traffic Signal and Approaching Automated Vehicles. In Proceedings of the IEEE Intelligent Vehicle Symposium, June 11-14, 2017, Redondo Beach, CA. (Selected for the Best Paper Award (2nd Prize) of the Symposium)
- Zhao, J., Li, W., Wang, J., Ban, X., 2016. Dynamic Traffic Signal Timing Optimization Strategy Incorporating Various Vehicle Fuel Consumption Characteristics. IEEE Transactions on Vehicular Technology 65 (6), 3874-3887.
- Borken, Jens, Heike Steller, Tamás Meretei, and Filip Vanhove. 2007. "Global and Country Inventory of Road Passenger and Freight Transportation: Fuel Consumption and Emissions of Air Pollutants in Year 2000." *Transportation Research Record* 2011 (1): 127–36.
- Yan, Fang, Ekbordin Winijkul, Soonkyu Jung, Tami C Bond, and David G Streets. 2011. "Global Emission Projections of Particulate Matter (Pm): I. Exhaust Emissions from on-Road Vehicles." *Atmospheric Environment* 45 (28): 4830–44.
- Walnum, Hans Jakob, and Morten Simonsen. 2015. "Does Driving Behavior Matter? An Analysis of Fuel Consumption Data from Heavy-Duty Trucks." *Transportation Research Part D: Transport and Environment* 36: 107–20.
- Row, Shelley J. 2010. "Intellidrive: Safer, Smarter, Greener." Public Roads 74 (1).
- Barić, Danijela, Goran Zovak, and Marko Periša. 2013. "Effects of Eco-Drive Education on the Reduction of Fuel Consumption and Co2 Emissions." *Promet-Traffic&Transportation* 25 (3): 265–72.
- Zarkadoula, Maria, Grigoris Zoidis, and Efthymia Tritopoulou. 2007. "Training Urban Bus Drivers to Promote Smart Driving: A Note on a Greek Eco-Driving Pilot Program." *Transportation Research Part D: Transport and Environment* 12 (6): 449–51.
- Li, S Eben, and Huei Peng. 2012. "Strategies to Minimize the Fuel Consumption of Passenger Cars During Car-Following Scenarios." *Proceedings of the Institution of Mechanical Engineers, Part D: Journal of Automobile Engineering* 226 (3): 419–29.
- Li, Shengbo Eben, Huei Peng, Keqiang Li, and Jianqiang Wang. 2012. "Minimum Fuel Control Strategy in Automated Car-Following Scenarios." *IEEE Transactions on Vehicular Technology* 61 (3): 998–1007.
- Li, Shengbo Eben, Qiangqiang Guo, Long Xin, Bo Cheng, and Keqiang Li. 2016. "Fuel-Saving Servo-Loop Control for an Adaptive Cruise Control System of Road Vehicles with Step-Gear Transmission." *IEEE Transactions on Vehicular Technology* 66 (3): 2033–43.

- Hellström, Erik, Maria Ivarsson, Jan Åslund, and Lars Nielsen. 2009. "Look-Ahead Control for Heavy Trucks to Minimize Trip Time and Fuel Consumption." *Control Engineering Practice* 17 (2): 245–54.
- Rakha, Hesham, and Raj Kishore Kamalanathsharma. 2011. "Eco-Driving at Signalized Intersections Using V2i Communication." In *2011 14th International Ieee Conference on Intelligent Transportation Systems (Itsc)*, 341–46. IEEE.
- Xia, Haitao, Kanok Boriboonsomsin, and Matthew Barth. 2013. "Dynamic Eco-Driving for Signalized Arterial Corridors and Its Indirect Network-Wide Energy/Emissions Benefits." *Journal of Intelligent Transportation Systems* 17 (1): 31–41.
- Huang, Yuhan, Elvin CY Ng, John L Zhou, Nic C Surawski, Edward FC Chan, and Guang Hong. 2018. "Eco-Driving Technology for Sustainable Road Transport: A Review." *Renewable and Sustainable Energy Reviews* 93: 596–609.
- Sutton, Richard S, and Andrew G Barto. 2018. *Reinforcement Learning: An Introduction*. MIT press.
- Qi, Xuewei, Guoyuan Wu, Kanok Boriboonsomsin, Matthew J Barth, and Jeffrey Gonder. 2016. "Data-Driven Reinforcement Learning–Based Real-Time Energy Management System for Plug-in Hybrid Electric Vehicles." *Transportation Research Record* 2572 (1): 1–8。
- Qi, Xuewei, Yadan Luo, Guoyuan Wu, Kanok Boriboonsomsin, and Matthew J Barth. 2017. "Deep Reinforcement Learning-Based Vehicle Energy Efficiency Autonomous Learning System." In *2017 Ieee Intelligent Vehicles Symposium (Iv)*, 1228–33. IEEE.
- Qi, Xuewei, Yadan Luo, Guoyuan Wu, Kanok Boriboonsomsin, and Matthew Barth. 2019. "Deep Reinforcement Learning Enabled Self-Learning Control for Energy Efficient Driving." *Transportation Research Part C: Emerging Technologies* 99: 67–81.
- Hu, Yue, Weimin Li, Kun Xu, Taimoor Zahid, Feiyan Qin, and Chenming Li. 2018. "Energy Management Strategy for a Hybrid Electric Vehicle Based on Deep Reinforcement Learning." *Applied Sciences* 8 (2): 187.
- Shi, Junqing, Fengxiang Qiao, Qing Li, Lei Yu, and Yongju Hu. 2018. "Application and Evaluation of the Reinforcement Learning Approach to Eco-Driving at Intersections Under Infrastructure-to-Vehicle Communications." *Transportation Research Record* 2672 (25): 89–98.
- Gamage, Hasitha Dilshani, and Jinwoo Lee. 2016. "Machine Learning Approach for Self-Learning Eco-Speed Control."
- Gamage, Hasitha Dilshani, and Jinwoo Brian Lee. 2017. "Reinforcement Learning Based Driving Speed Control for Two Vehicle Scenario." In *Australasian Transport Research Forum (Atrf), 39th.*
- Qu, Xiaobo, Yang Yu, Mofan Zhou, Chin-Teng Lin, and Xiangyu Wang. 2020. "Jointly Dampening Traffic Oscillations and Improving Energy Consumption with Electric, Connected and Automated Vehicles: A Reinforcement Learning Based Approach." *Applied Energy* 257: 114030.

- Hao, Peng, Zhensong Wei, Zhengwei Bai, and Matthew J Barth. 2020. "Developing an Adaptive Strategy for Connected Eco-Driving Under Uncertain Traffic and Signal Conditions."
- Zhao, Tingting, Hirotaka Hachiya, Gang Niu, and Masashi Sugiyama. 2011. "Analysis and Improvement of Policy Gradient Estimation." In *Advances in Neural Information Processing Systems*, 262–70.
- Silver, David, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. 2014. "Deterministic Policy Gradient Algorithms."
- Lillicrap, Timothy P, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. 2015. "Continuous Control with Deep Reinforcement Learning." *arXiv Preprint arXiv:1509.02971*.
- Watkins, Christopher JCH, and Peter Dayan. 1992. "Q-Learning." *Machine Learning* 8 (3-4): 279–92.
- Mnih, Volodymyr, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, et al. 2015. "Human-Level Control Through Deep Reinforcement Learning." *Nature* 518 (7540): 529–33.
- Lin, Long-Ji. 1992. "Self-Improving Reactive Agents Based on Reinforcement Learning, Planning and Teaching." *Machine Learning* 8 (3-4): 293–321.
- Tsitsiklis, JN, and B Van Roy. 1996. "An Analysis of Temporal-Difference Learning with Function approximationTechnical." *Report LIDS-P-2322). Laboratory for Information and Decision Systems, Massachusetts Institute of Technology, Tech. Rep.*
- Mnih, Volodymyr, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. 2013. "Playing Atari with Deep Reinforcement Learning." *arXiv Preprint arXiv:1312.5602.*
- Lopez, Pablo Alvarez, Michael Behrisch, Laura Bieker-Walz, Jakob Erdmann, Yun-Pang Flötteröd, Robert Hilbrich, Leonhard Lücken, Johannes Rummel, Peter Wagner, and Evamarie Wießner. 2018. "Microscopic Traffic Simulation Using Sumo." In *The 21st Ieee International Conference on Intelligent Transportation Systems*. IEEE. <https://elib.dlr.de/124092/
- Krauß, Stefan. 1998. "Microscopic Modeling of Traffic Flow: Investigation of Collision Free Vehicle Dynamics." PhD thesis.
- Krauß, Stefan, Peter Wagner, and Christian Gawron. 1997. "Metastable States in a Microscopic Model of Traffic Flow." *Physical Review E* 55 (5): 5597.
- Erdmann, Jakob. 2014. "Lane-Changing Model in Sumo." *Proceedings of the SUMO2014 Modeling Mobility with Open Data* 24: 77–88.
- Xu, Shaobing, Shengbo Eben Li, Xiaowu Zhang, Bo Cheng, and Huei Peng. 2015. "Fuel-Optimal Cruising Strategy for Road Vehicles with Step-Gear Mechanical Transmission." *IEEE Transactions on Intelligent Transportation Systems* 16 (6): 3496–3507.

- Horváth, M., Lu, Q., Tettamanti, T., Török, Á., Szalay, Z., (2019). Vehicle-In-The-Loop (VIL) and Scenario-In-The-Loop (SCIL) Automotive Simulation Concepts from the Perspectives of Traffic Simulation and Traffic Control. Transport and Telecommunication Journal. 20. 153-161. 10.2478/ttj-2019-0014.
- Tettamanti, T., Szalai, M., Vass S., and Tihanyi, V., (2018). Vehicle-In-the-Loop Test Environment for Autonomous Driving with Microscopic Traffic Simulation. *2018 IEEE International Conference on Vehicular Electronics and Safety (ICVES)*, Madrid. 1-6, doi: 10.1109/ICVES.2018.8519486.
- Butenuth, M., Kallweit, R. & Prescher, P., (2017). Vehicle-in-the-Loop Real-world Vehicle Tests Combined with Virtual Scenarios. *ATZ Worldw* 119, 52–55. https://doi.org/10.1007/s38311-017-0082-4
- Winner, H., (2013). Challenges of Automotive Systems Engineering for Industry and Academia. In: Maurer, M.; Winner, H. (Ed.): Automotive Systems Engineering. Berlin/Heidelberg: Springer-Verlag, 3–15.
- Haas, J. K. (2014). A history of the unity game engine. Retrieved from https://digitalcommons.wpi.edu/iqp-all/3207.
- Balaji, B., Mallya, S., Genc, S., Gupta, S., Dirac, L., Khare, V., Roy, G., Sun, T., Tao, Y., Townsend, B., Calleja, E., Muralidhara, S., Karuppasamy, D., (2019). DeepRacer: Educational Autonomous Racing Platform for Experimentation with Sim2Real Reinforcement Learning.
- Wong, W., Shen, S., Zhao, Y. and Liu, H.X., 2019. On the estimation of connected vehicle penetration rate based on single-source connected vehicle data. Transportation Research Part B: Methodological, 126, pp.169-191.

Appendix A: Commands used in Figure 15, 16, and Figure 20

- checkStepSUMO:
    - output: step number in the SUMO simulation
- getBeginSUMO:
    - output:
        - current road system: the current road system the controlled vehicle is on
        - current signal phase: if the current road system is an intersection, then return the current signal phase
        - information of all vehicles on the current road system: location, speed and heading angle of each vehicle
- getResponseFmUnity:
    - output: if the Unity is ready, then return “sumo”; otherwise return “cook”
- getNextSUMO:

- output:
  - current road system: the current road system the controlled vehicle is on
  - incoming road system: the next road system the controlled vehicle will be on
  - information of all vehicles on the current road system: location, speed and heading angle of each vehicle
  - current or next signal phase:
    - if the next road system is an intersection and is close enough to the controlled vehicle, then return the next signal phase; otherwise use the current traffic signal phase if the current road system is an intersection
    - if the current road system is an intersection, then return the current signal phase

- CookNetwork:
  - check road type
    - if straight
    - check whether it is in a new direction
      - if yes: generate a new straight road
      - if no: do nothing
    - if intersection: generate a new intersection
- CookTrafficLight:
  - sort the locations of twelve traffic light movements
  - put the phases to the movements one by one by their locations clockwisely
- CookVehicle:
  - if existing: change the position and heading angle of the vehicle
  - if newly-arrived: create a new vehicle based on its vehicle category
- recvMsg:
  - receive driving actions (i.e., speed and steering) from control center
- controlDR:
  - send the driving commands to AWS DeepRacer control panel to change the speed and heading angle of the DeepRacer
- sendMsg:
  - send driving actions (i.e., speed and steering) to SUMO