Reinforcement Learning in Eco-driving for Connected and Automated Vehicles

Dissertation

Presented in Partial Fulfillment of the Requirements for the Degree Doctor of Philosophy in the Graduate School of The Ohio State University

By

Zhaoxuan Zhu, M.S.

Graduate Program in Mechanical Engineering

The Ohio State University

2021

Dissertation Committee:

Professor Marcello Canova, Advisor

Professor Abhishek Gupta

Professor David Hoelzle

Abstract

Connected and Automated Vehicles (CAVs) can significantly improve transportation efficiency by taking advantage of advanced connectivity technologies. Meanwhile, the combination of CAVs and powertrain electrification, such as Hybrid Electric Vehicles (HEVs) and Plug-in Hybrid Electric Vehicles (PHEVs), offers greater potential to improve fuel economy due to the extra control flexibility compared to vehicles with a single power source. In this context, the eco-driving control optimization problem seeks to design the optimal speed and powertrain components usage profiles based upon the information received by advanced mapping or Vehicle-to-Everything (V2X) communications to minimize the energy consumed by the vehicle over a given itinerary. To overcome the real-time computational complexity and embrace the stochastic nature of the driving task, the application and extension of state-of-the-art (SOTA) Deep Reinforcement Learning (Deep RL, DRL) algorithms to the eco-driving problem for a mild-HEV is studied in this dissertation.

For better training and a more comprehensive evaluation, an RL environment, consisting of a mild HEV powertrain and vehicle dynamics model and a large-scale microscopic traffic simulator, is developed. To benchmark the performance of the developed strategies, two causal controllers, namely a baseline strategy representing human drivers and a deterministic optimal-control-based strategy, and the non-causal wait-and-see solution are implemented.

In the first RL application, the eco-driving problem is formulated as a Partially Observable Markov Decision Process, and a SOTA model-free DRL (MFDRL) algorithm, Proximal Policy Optimization with Long Short-term Memory as function approximator, is used. Evaluated over 100 trips randomly generated in the city of Columbus, OH, the MFDRL agent shows a 17% fuel economy improvement against the baseline strategy while keeping the average travel time comparable. While showing performance comparable to the optimal-control-based strategy, the actor of the MFDRL agent offers an explicit control policy that significantly reduces the onboard computation.

Subsequently, a model-based DRL (MBDRL) algorithm, Safe Model-based Off-policy Reinforcement Learning (SMORL) is proposed. The algorithm addresses the following issues emerged from the MFDRL development: *a*) the cumbersome process necessary to design the rewarding mechanism, *b*) the lack of the constraint satisfaction and feasibility guarantee and *c*) the low sample efficiency. Specifically, SMORL consists of three key components, a massively parallelizable dynamic programming trajectory optimizer, a value function learned in an off-policy fashion and a learned safe set as a generative model. Evaluated under the same conditions, the SMORL agent shows a 21% reduction on the fuel consumption over the baseline and the dominant performance over the MFDRL agent and the deterministic optimal-control-based controller.

To my family.

Acknowledgments

First and foremost, I would like to acknowledge my advisor Prof. Marcello Canova. He brought me into the research in the automotive field, and since then consistently offer me great trust and advice. I would like to thank Prof. Abhishek Gupta for many insightful and inspiring discussions and Prof. David Hoelzle for his invaluable comments and suggestions.

I would like to thank the colleagues I met during my years at Center for Automotive Research. It is them who make the Ph.D. experience much more interesting. I would like to thank Li Tang, Tianpei Li, Yuxing Liu, Ruochen Yang, Cheng Ye, Mingru Zhao, Xin Jin, Cris Rostiti, Shobhit Gupta, Dennis Kibalama, Polina Brodsky, Ke Pan and Tong Zhao for all the meaningful discussions about research, career and random things. In particular, I want to thank Nicola Pivaro, who helped me develop and integrate many early ideas in the reinforcement learning algorithm. I want to thank Delphi Technologies and Motional for offering me internship opportunities that provide me with industry insights and incite many ideas for my Ph.D. study.

I cannot be grateful enough to my parents Qiuping Hao and Zhiqiang Zhu for their tremendous support and love. Without them, I would not have the patience or courage to start, let alone finish, my Ph.D. journey. Finally, I want to thank my girlfriend Yishen Jin for her patience, accompany, and love throughout.

Vita

March 28, 1993 ..... Born - Shanxi, China

2016 ..... B.S. Mechanical Engineering,
Ohio State University, Columbus, OH, US

2018 ..... M.S. Mechanical Engineering,
Ohio State University, Columbus, OH, US

Publications

Research Publications

Z. Zhu, M. Canova and S. Midlam-Mohler "A Physics-Based Three-Way-Catalytic Converter Model for Real-time Prediction of Temperature Distribution". *IFAC E-COSM*, 2018

Z. Zhu, S. Midlam-Mohler and M. Canova "Development of physics-based three-way catalytic converter model for real-time distributed temperature prediction using proper orthogonal decomposition and collocation". *International Journal of Engine Research, 22(3), pp.873-889*, Mar. 2021

Z. Zhu, Y. Liu and M. Canova "Energy Management of Hybrid Electric Vehicles via Deep Q-Networks". *American Control Conference*, 2020

Z. Zhu, S. Gupta, N. Pivaro, SR. Deshpande and M. Canova "A GPU Implementation of a Look-Ahead Optimal Controller for Eco-Driving Based on Dynamic Programming". *European Control Conference*, 2021

Z. Zhu, S. Gupta, A. Gupta and M. Canova "A Deep Reinforcement Learning Framework for Eco-driving in Connected and Automated Hybrid Electric Vehicles". *arXiv*

preprint arXiv:2101.05372 , 2021 (Submitted to IEEE Transactions on Intelligent Transportation Systems)

Z. Zhu, N. Pivaro, S. Gupta, A. Gupta and M. Canova "Safe Model-based Off-policy Reinforcement Learning for Eco-driving in Connected and Automated Hybrid Electric Vehicle". *arXiv preprint arXiv:2105.11640*, 2021 (Submitted to IEEE Transactions on Intelligent Vehicles)

Fields of Study

Major Field: Mechanical Engineering

Table of Contents

List of Tables

List of Figures

Chapter 1: Introduction

1.1 Eco-driving Problem

A conflicting challenge that the automotive industry always faces is to consistently reduce fuel consumption and emissions while offering an uncompromised driving/riding experience. Over the last two decades, vehicle electrification, hybridization, and engine downsizing and boosting have been shown effective in reducing fuel consumption and emissions [50]. Furthermore, these technologies are also cost-effective and foreseen to have a larger market share in the near future [80].

In the meantime, with the advancement in vehicular connectivity and autonomy, Connected and Automated Vehicles (CAVs) have the potential to operate in a safer and more time- and fuel-efficient manner [105]. As shown in Fig. 1.1, the Vehicle-to-Vehicle (V2V) and Vehicle-to-Infrastructure (V2I) communication offer CAVs access to various real-time look-ahead information of terrain, infrastructure, surrounding vehicles, etc. With connectivity technologies, the ego-vehicle can then plan a speed profile with less unnecessary fluctuations and a power profile with higher efficiency by intelligently passing more signalized intersections without full stop [96] and by cooperatively following/leading other vehicles on the road [111].

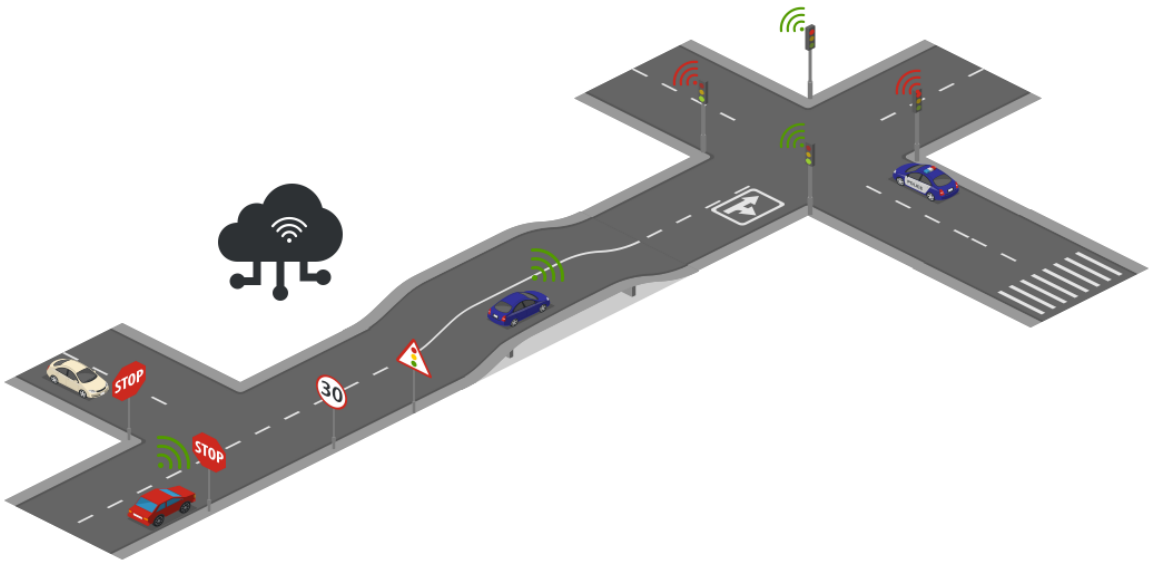


Figure 1.1: Vehicle Connectivity under Complex Driving Scenarios

Integrating Hybrid Electric Vehicle (HEV) and CAV technologies can further improve the fuel economy, as the additional information about the driving conditions helps the HEV controller utilize its multiple power sources more strategically [25, 73].

This task is generally referred to as the eco-driving problem or speed trajectory planning problem, and its objective is to co-optimize the speed trajectory and the powertrain control strategy between two designated locations. Multiple objectives could be considered in the optimization problem, most frequently the fuel consumption and the travel time [49, 89, 110].

Fig.1.2 shows an example of hierarchical architecture of CAV control systems with the consideration on fuel economy [96]. The modules highlighted in red are the primary interests of this study. Once the eco-driving module receives the routing information, an optimal speed trajectory is designed based on multiple objectives

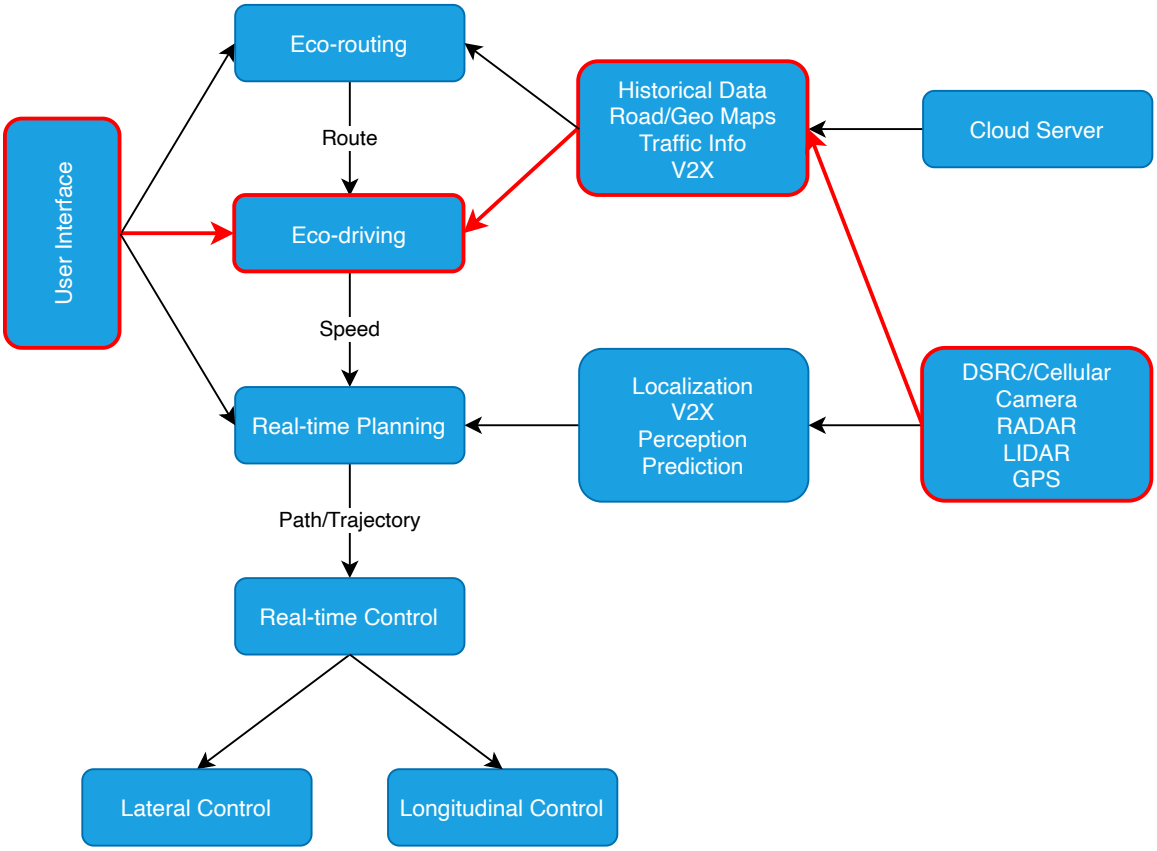


Figure 1.2: CAV Hierarchical Control Architecture [96]

and constraints including driving styles, complex traffic scenarios and powertrain configurations.

1.2 Literature Review

The literature related to the eco-driving problem distinguishes among three aspects, namely powertrain configurations, traffic scenarios, and formulation domain. Regarding powertrain configuration, the difference is in whether the powertrain is

equipped with a single power source [39, 49, 76, 96] or a hybrid electric architecture [7,34,73]. The latter requires more modeling efforts and a more complex control algorithm since the battery SoC needs to be regulated and utilized efficiently.

Meanwhile, the difference also exists in the complexity of the driving scenarios. Ozatay et al. [76] proposed a framework providing advisory speed profile using online optimization conducted on a cloud-based server without considering the real-time traffic light variability. Olin et al. [73] implemented an eco-driving framework to evaluate real-world fuel economy benefits from in-vehicle testing. As traffic lights are not explicitly considered in these studies, the eco-driving controller is required to be coupled with other decision-making agents, such as human drivers or rule-based controllers [4]. Other studies have explicitly modeled and considered Signal Phase and Timings (SPaTs). Jin et al. [49] formulated the problem as a Mixed Integer Linear Programming (MILP) for conventional vehicles with Internal Combustion Engine (ICE). Asadi et al. [6] used traffic simulation models and proposed to solve the problem considering probabilistic SPaT with Deterministic Dynamic Programming (DDP). Sun et al. [96] formulated the eco-driving problem as a distributionally robust stochastic optimization problem with collected real-world data. Guo et al. [34] proposed a bi-level control framework with a hybrid vehicle. Guo and Wang [33] developed a multi-step cooperative optimization of speed trajectory and traffic signal priority.

Finally, the optimization problem formulation for eco-driving can also be classified into time-domain [5, 34, 49] and spatial domain [7, 73, 76, 96]. While the formulation in the time domain is more straightforward, the formulation in the spatial domain is advantageous in two aspects. First, the problem can be formulated with finite steps

in spatial domain since the control task is to reach the destination from the origin, whereas the total number of steps in time domain depends on the vehicle velocity. Second, when constraints such as speed limits and the traffic light conditions are set in the Model Predictive Control (MPC) horizon, they are inherently related to the vehicle position along the trip, whereas in the time domain, they need to be set according to the vehicle velocity dynamically.

1.3 Motivation

In this dissertation, the eco-driving problem of Connected and Automated Hybrid Electric Vehicles (CAHEVs) with the capability of passing traffic lights autonomously is studied. The problem is identified as a reference-free stochastic nonlinear Optimal Control Problem (OCP). Specifically, the following technical challenges are identified.

First, unless with significant simplifications, the mathematical model of the system is a) nonlinear and non-convex, due to the fuel consumption maps of the internal combustion engine, b) hybrid, due to the integer nature of the gear shift, and c) stochastic, due to the uncertain nature of real-world driving. Furthermore, the problem requires searching for a global optimal trajectory rather than tracking an existing reference, resulting in a much larger search space. The problem also might limit the choice of solvers, as gradient-based methods might converge to local optima.

Second, the eco-driving problem is subject to several constraints, including those from powertrain components, driving comfort, and vehicle and traffic safety. While some constraints can be converted to soft constraints, the presence of safety-critical constraints requires the designed controller to be robust.

Furthermore, both the speed and battery State of Charge (SoC) plannings require the controller to make long-term strategic decisions. Control strategies that are short-sighted can be less performant and run into feasibility issues.

Finally, the design of the algorithm and the selection of the hardware need to be considered synergistically for real-time implementation.

In the existing eco-driving studies, two prevailing optimization and control techniques are used to mitigate the aforementioned challenges. [32, 33, 49] seek to either simplify the powertrain model or decouple the speed planning and powertrain control such that real-time controllers can be developed. The advantage of these studies is that the proposed control strategies are computationally efficient. However, the methodology can be limited to certain assumptions, and performance can be lost during simplification and decoupling. On the other hand, [6, 73, 76, 96] use DDP to solve the coupled problem with little simplification to powertrain and vehicle dynamics. While DDP offers the global optimal solution to problem formulation, it suffers from the curse of dimensionality, and is computationally expensive, especially for automotive onboard controllers.

This work presents two original implementations and extensions of the theory of Reinforcement Learning (RL) to solve the eco-driving problem in a computationally efficient fashion without aggressive model simplifications that can potentially deteriorate the performance. The objective of RL is to train a decision-making agent (control strategy) via interaction with an environment. As shown in Fig. 1.3, RL, in general, uses a train-offline-execute-online framework. Here, the environment encodes the physics of the problem to be studied and provides feedback to the learning agent. In practice, it can either be a reactive simulator or data from the real world. The

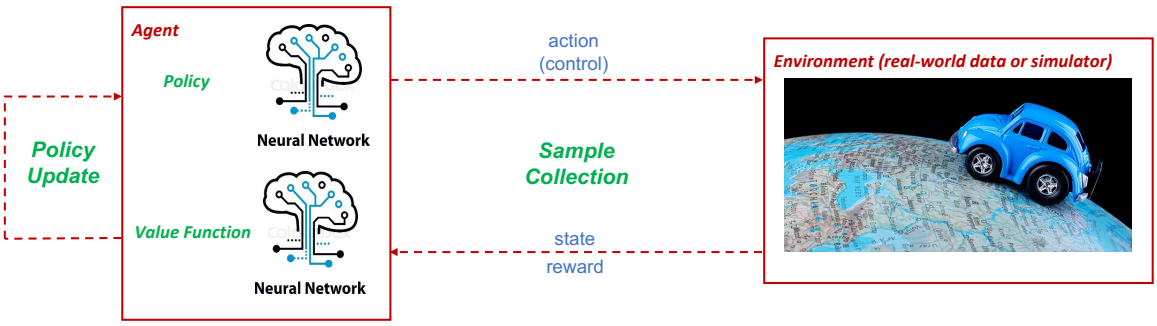


Figure 1.3: General Reinforcement Learning Framework

learning agent is the one to be trained, and depending on the choice of algorithms, either an explicit policy can be learned to directly map system state to action, or a value function can be learned to help make decisions that are strategic in long term. In either case, a more performant control strategy can be learned iteratively via the interaction with the environment.

RL can be broadly categorized into model-free RL (MFRL) and model-based RL (MBRL). In MFRL, the agent learns its policy purely by the interaction with the environment without either the knowledge or trying to build the knowledge about the environment. In Chapter 5, an MFRL agent will be developed. With the advantage being its onboard computational efficiency and the disadvantages being the long training time and the lack of safety guarantee, the MFRL agent shows a significant improvement over the baseline strategy. On the other hand, MBRL either uses the existing knowledge about the environment or learns the environment models on the fly to improve the sample efficiency of the learning algorithm and to impose constraints. In Chapter 6, an MBRL agent that combines RL and MPC will be presented. The

algorithm explicitly imposes the safety constraints related to the problem, reduces the training time and complexity, and improves the overall performance metrics.

1.4 Contributions

The focus of this dissertation is to develop a train-offline-execute-online RL framework to address the computational and performance-wise limitations to the existing approaches for the eco-driving problem. Compared to the literature reviewed in Section 1.2, the contributions are as follows.

First, a comprehensive eco-driving environment is developed for both training and testing purposes. The simulation environment integrates two key components, namely an experimentally validated powertrain and vehicle dynamics model and a traffic model wrapped around an open-source microscopic traffic simulator Simulation of Urban Mobility (SUMO) to generate different driving scenarios in an arbitrarily large map on a large scale.

Then, a parallel deterministic dynamic programming (PDDP) solver for the eco-driving problem is developed. The computationally efficient implementation enables *a*) fast access to the deterministic optimal solution given the constraints and reference of the entire itinerary, and *b*) a real-time implementable trajectory optimization solver that reliably provides the global optimal solution within the rollout length.

Next, the eco-driving problem is formulated as a Partially Observable Markov Decision Process (POMDP), and a control policy is trained using a state-of-the-art (SOTA) MFRL algorithm, Proximal Policy Optimization (PPO) along with Long Short-Term Memory (LSTM). The resulting explicit policy is computationally efficient in real-time as it does not require any online optimization. show that the

performance of the MFRL is significantly better than the baseline strategy, and it is comparable with the more computationally demanding optimal control strategy.

Finally, to simplify the reward design while ensuring constraints satisfaction, a more sophisticated MBRL algorithm, Safe Model-based Off-policy RL (SMORL), is proposed. In this method, the reinforcement learning framework is integrated with an efficient trajectory optimization solver and a generative model approximating the robust control invariant set. Compared to the MFRL approach, constraint satisfaction and feasibility are explicitly considered, and the performance of the trained policy is significantly improved.

1.5 Dissertation Outline

This dissertation is organized as follows. In Chapter 2, the necessary background information regarding the techniques utilized in this work will be reviewed. In Chapter 3, the RL environment is introduced. The mathematical problem formulation, the PDDP solver and the three benchmarking strategies are presented in Chapter 4. The details of the MFRL and MBRL algorithms are presented in Chapter 5 and Chapter 6, respectively. Finally, conclusions and future work are discussed in Chapter 7.

Chapter 2: Background

2.1 Dynamic Programming

2.1.1 Notation

Consider the stationary discrete-time dynamic system [12]

$$x_{k+1} = f(x_k, u_k, w_k), \quad k = 0, 1, \ldots, \tag{2.1}$$

where for all k, the state x_k is an element of a space $\mathcal{X}$, the control u_k is an element of a space $\mathcal{U}$, and the random disturbance w_k is an element of a space $\mathcal{W}$, which is assumed to be a countable set. The control u_k is constrained in a nonempty subset $U(x_k)$, and w_k follows the conditional probability distribution $P(\cdot|x_k, u_k)$.

Given an initial state x_0, the objective is to find the optimal policy

$$\pi = \{\mu_0, \mu_1, \ldots\}, \tag{2.2}$$

where $\mu_k : \mathcal{X} \to \mathcal{U}, \mu_k \in U(x_k)$, for all $x_k \in \mathcal{X}, k = 0, 1, \ldots$, that minimizes the cost function

$$J_\pi(x_0) = \lim_{N\to\infty} \mathop{\mathbb{E}}_{w_k \; k=1,2,\ldots} \left\{ \sum_{k=0}^{N-1} \gamma^k g(x_k, \mu_k(x_k), w_k) \right\}, \tag{2.3}$$

subject to the Eqn. (2.1). In general, $\mathbb{E}_{x\sim P(x)}[f(x)]$ means the expected value of the function $f(x)$ with respect to the random variable x following the distribution of $p(x)$. $g : \mathcal{X} \times \mathcal{U} \times \mathcal{W} \to \mathbb{R}$ is the cost per stage. $\gamma \in (0, 1]$ is the discount factor.

When the state space $\mathcal{X}$ is finite or countable, the aforementioned system can be equivalently formulated as a finite state Markov chain. Here, states can be denoted by $i = 1, 2, \ldots, n$, and the system dynamics can be described by the transition probability matrix $p_{ij}(u)$:

$$p_{ij}(u) = P(x_{k+1} = j | x_k = i, u_k = u), \quad i, j \in \mathcal{X}, u \in \mathcal{U}. \tag{2.4}$$

Note that the disturbance $w_k \sim P(\cdot|x_k, u_k)$ is now embedded in the transition probability matrix.

2.1.2 Finite Horizon Problem

Suppose the problem has a fixed number of stages N, the optimal policy $\pi^* = \{\mu_1, \mu_2, \ldots, \mu_N\}$, along with the optimal cost-to-go function

$$J_k = \underset{w_k,\, k=0,1,\ldots}{\mathbb{E}} \left\{ \gamma^N J(x_N) + \sum_{l=k}^{N-1} \gamma^l g(x_l, \mu_l(x_l), w_l) \right\}, \tag{2.5}$$

can be determined using Stochastic Dynamic Programming (SDP) via backward recursion:

$$J_{N-k}(x) = \min_{u \in U(x)} \underset{w_k,\, k=0,1,\ldots}{\mathbb{E}} \left\{ \gamma^{N-k} g(x, u, w) + J_{N-k+1}\left(f(x, u, w)\right) \right\}, \tag{2.6}$$

with the initial condition

$$J_N(x) = \gamma^N J(x). \tag{2.7}$$

A special type of problem is when the system is deterministic, and the disturbance $w_k = 0, k = 0, 1, \ldots, N$. In the deterministic setup, Eqn. (2.6) can be simplified as

$$J_{N-k}(x) = \min_{u \in U(x)} \gamma^{N-k} g(x, u) + J_{N-k+1}\left(f(x, u)\right). \tag{2.8}$$

To convert the constrained optimization problem to a unconstrained one and to allow step-varying state dynamics and constraints, Eqn. (2.8) is reformulated as

$$\mathcal{J}_N(x) = g_N(x) + \phi_N(x), \tag{2.9}$$

$$\mu_k^* = \underset{\mu_k}{\operatorname{argmin}}\, \mathcal{J}_k(x) = \underset{\mu_k}{\operatorname{argmin}} \left[g_k(x, \mu_k(x)) + \phi_k(x) + \mathcal{J}_{k+1}(f_k(x, \mu_k(x))) \right]. \tag{2.10}$$

where g_k and g_N are the discrete stage and final cost function, respectively. ϕ_k and ϕ_N are penalty functions introduced to ensure that the trajectory stay feasible.

Solving such optimization problems analytically or by gradient methods is not always feasible. In [97, 98], the state and control spaces are discretized, and the optimization is conducted by numerically comparing the exhaustive combination of state and control at each stage of backward recursion. Nevertheless, such method significantly increases the computational demand and limits the method to problems with state and action in low dimensional space. To overcome the excessive computational cost, In Chapter 4, a practical Parallel Deterministic Dynamic Programming (Parallel DDP, PDDP) solver that significantly reduces the computational time will be presented.

2.1.3 Infinite Horizon Problem

When the problem has infinite horizon, instead of calculating the optimal cost-to-go and optimal policy directly from backward recursion, as in Eqn.(2.10), the optimal policy is typically obtained iteratively. Here, the markovian formulation defined by Eqn. (2.4) is used for more concise representation.

In this section, two algorithms, namely Value Iteration (VI) and Policy Iteration (PI) will be briefly presented. Detailed proofs regarding their convergence and properties are referred to [12].

First, two operators T and T_μ are defined as follows:

$$(TJ)(i) = \min_{u \in U(i)} \sum_{j=1}^{n} p_{ij}(u) \left(g(i,u,j) + \gamma J(j) \right), \quad i \in \mathcal{X} \tag{2.11}$$

$$(T_\mu J)(i) = \sum_{j=1}^{n} p_{ij}(\mu(i)) \left(g(i,\mu(i),j) + \gamma J(j) \right), \quad i \in \mathcal{X} \tag{2.12}$$

Here, J is initialized as a random vector of n elements.

In VI, the optimal cost function $J^*(\dot{})$ is obtained by

$$\lim_{k \to \infty} (T^k J)(i) = J^*(i), \tag{2.13}$$

where $T^k J$ means iteratively applying the operator T for k times. J^* needs to be computed iteratively because the operator T is nonlinear.

In PI, an initial policy μ^0 is required apart from J_0 to start the algorithm. The optimal cost function is then obtained by iteratively applying the following two steps:

Policy Evaluation: Given the stationary policy μ^k, compute the corresponding cost function J_{μ^k} from the linear system of equations

$$J_{\mu^k} = T_{\mu^k} J_{\mu^k}. \tag{2.14}$$

Policy Improvement: Obtain a new stationary policy μ^{k+1} satisfying

$$J_{\mu^{k+1}} = T J_{\mu^k}. \tag{2.15}$$

The iteration stops when the following condition is satisfies:

$$J_{\mu_{k+1}} = T J_{\mu_k} \tag{2.16}$$

Although being able to offer solution(s) to the infinite horizon problem, VI and PI have several limitations. First, the aforementioned methods can only handle problems with discrete state and action space. Second, when the dimension of the action

space is large, the minimization operation becomes very expensive for real-time applications. Third, the method requires knowledge of both the state dynamics $f(x, u, w)$ and the distribution of the disturbance w. This requirement, for many engineering applications, especially those in high dimensions, demands an excessive amount of modeling efforts.

2.1.4 The Use of Dynamic Programming in Eco-driving

DP is primarily utilized in the study of the Energy Management (EM) problem and eco-driving problem in the following three ways.

1. In many applications of optimal control, DDP is used to solve finite horizon problems to provide a benchmark for optimality, rather than designing a causal suboptimal controller [97]. The problem formulation assumes the reference and the constraints of the entire horizon are known a *priori* (deterministically). As a result, the wait-and-see (WS) solution [14] serves as the optimal solution to a given realization of the general stochastic process.

2. When Model Predictive Control (MPC) is formulated such that the reference and constraints are assumed to be accessible and accurate within the shorter receding horizon, DDP is used as the solver to the finite horizon trajectory optimization problem [24, 25, 73]. Compared to other gradient-based MPC solvers [108, 119], the accuracy of DDP is less affected by the non-linearity or convexity of the problem, but more by the size of grids of the state and control spaces. When the dimension of the optimization problem is low, DDP provides a more robust solution.

3. When a causal policy that takes uncertainties into account is desired, VI and PI have been adopted under the infinite horizon formulation. For instance, studies [61, 71] have applied SDP to the energy management problem to explicitly consider uncertainties from driving behaviors. While this study does not directly apply SDP to the eco-driving problem, its background offers some fundamental concepts to the proposed method.

2.2 Reinforcement Learning

The subject of Reinforcement Learning (RL) [100] aims to solve a similar problem as DP, specifically the decision-making under uncertain environments. Many techniques in RL extend the fundamental techniques from DP and aim to address the computational and performance-related limitations of DP. In recent years, Deep Reinforcement Learning (DRL), namely an RL algorithm coupled with various Neural Networks (NN) as function approximators, has drawn significant research, development and public attention due to its application to many problems in industry, economics, and society at large [11, 69, 92, 109].

In this section, the common notation in RL will be first introduced. Subsequently, common techniques and concepts of DRL will be then reviewed.

2.2.1 Markov Decision Process

In a Markov Decision Process (MDP), sequential decisions are made in order to maximize the discounted sum of the rewards. An MDP can be defined by a tuple $\langle \mathcal{S}, \mathcal{A}, P, \rho_0, r, \gamma \rangle$, where $\mathcal{S}$ and $\mathcal{A}$ are the state space and the action space, respectively; $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is the transition dynamics distribution; ρ_0 is the initial distribution of the state space. The reward function $r : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is a

function that maps the tuple (s_t, a_t, s_{t+1}) to an instantaneous reward. Finally, γ is a discount factor that prioritizes the immediate reward and ensures that the summation over an infinite horizon is finite.

Let $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$ be a randomized policy and Π be the set of all randomized policies. The objective of MDP is to find the optimal policy π^* that maximizes the expectation of the discounted sum of the rewards defined as follows:

$$\begin{gathered} \pi^* = \underset{\pi \in \Pi}{\operatorname{argmax}}\, \eta(\pi), \text{ where} \\ \eta(\pi) = \mathbb{E}_{s_{t+1} \sim P(\cdot|s_t, a_t)} \left[\sum_{t=0}^{\infty} \gamma^t r\left(s_t, a_t\right) \right], \\ \text{where } s_0 \sim \rho_0(\cdot), \quad a_t \sim \pi(\cdot|s_t). \end{gathered} \tag{2.17}$$

In the remaining work, the expectation under the state trajectory $\mathbb{E}_{s_{t+1} \sim P(\cdot|s_t, a_t)}[\cdot]$ will be written compactly as $\mathbb{E}_{\pi}[\cdot]$.

Note that the policy can also be deterministic $\pi : \mathcal{S} \to \mathcal{A}$, in which case the expectation of the discounted sum of the rewards becomes

$$\eta(\pi) = \mathbb{E}_{\pi} \left[\sum_{t=0}^{\infty} \gamma^t r\left(s_t, \pi(s_t)\right) \right]. \tag{2.18}$$

For any policy π, the value function $V^{\pi} : \mathcal{S} \to \mathbb{R}$, the Q function $Q^{\pi} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ and the advantage function $A^{\pi} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ are defined as follows:

$$V^{\pi}(s_t) = \mathbb{E}_{\pi} \left[\sum_{i=t}^{\infty} \gamma^{i-t} r\left(s_i, a_i\right) | s_t \right], \tag{2.19}$$

$$Q^{\pi}(s_t, a_t) = \mathbb{E}_{\pi} \left[\sum_{i=t}^{\infty} \gamma^{i-t} r\left(s_i, a_i\right) | s_t, a_t \right], \tag{2.20}$$

$$A^{\pi}(s_t, a_t) = Q^{\pi}(s_t, a_t) - V^{\pi}(s_t). \tag{2.21}$$

Intuitively, the value function $V^{\pi}(s_t)$ represents the average cumulative rewards the agent collects starting from state s_t and following policy π. The Q function $Q^{\pi}(s_t, a_t)$ represents the average cumulative rewards the agent collects starting from

state s_t, applying a_t as the action at the current step and subsequently following policy π. The advantage function $A^\pi(s_t, a_t)$ is the difference between the Q function and the value function, and it measures how advantageous is to apply a_t at the current step than following π from the current step.

2.2.2 Function Approximators

In Sec. 2.1, the cost function J is tabular and defined for the discrete state $x_k \in \mathcal{X}$. The tabular setting, however, requires an excessive amount of memory to store for problems with large state space. In addition, the tabular setup works only if the state and action spaces are discrete.

One key feature that made RL methods so widely applied in recent times is the use of function approximators [13, 100]. In the rest of the work, the combination of RL and function approximators, in particular Deep Neural Networks (Deep NNs, DNNs), will be referred to as Deep Reinforcement Learning (DRL).

Two types of DNNs will be used in work, namely Feedforward Networks (also known as Multilayer Perceptrons, MLPs) and Recurrent Neural Networks (RNNs). For brevity, the introduction to these concepts and the practical concerns in implementation are referred to [31].

MLP is a type of nonlinear function approximator that approximates the true function $y = f(x)$ by $f_\theta(x)$, where θ is a set of parameters (or weights). Training an MLP requires one to optimally tune the weights θ, using a technique known as backpropagation. The advantage of MLP compared to other types of function approximators, such as State Vector Machine (SVM) or Gaussian Process (GP), is that it can approximate very complex functions as long as the networks are sufficiently

large [44]. On the other hand, MLPs typically require a large amount of data, and techniques preventing overfitting, such as regularization, dropout [95] and batch normalization [46], are commonly needed to give a satisfying performance.

Compared to MLP, RNNs specialize on the prediction of time-series data. Defining u_t to be the inputs to the system at index t, and x_t to be the hidden state of the system at index t, RNN can be described as

$$x_t = f_\theta(x_{t-1}, u_t). \tag{2.22}$$

Here, the hidden state is delayed with one time step, and fed back to the network recurrently. The training technique used for RNNs is called Backpropagation Through Time (BPTT). When the gradient is backpropagated through a long sequence, the training is subject to gradient vanish/explosion [78]. Long Short-term Memory (LSTM) [43] and Gated Recurrent Unit (GRU) [18] are two gating mechanisms that provide better training stability and long-term dependency.

Specific to RL in the context of this dissertation, the value function, Q function and policy will be approximated by different types of DNNs represented by a set of parameters. For example, $V_\theta^{\pi_\xi}$ means the value function under the policy π_ξ is parameterized by θ, and the policy π_ξ is also a function approximator parameterized by ξ. During the training of the algorithm, the parameters are updated such that the objective function, which in general represents how close is the parameterized function to the true underlying function, is minimized. To compute the gradient of the objective function with respect to the set of parameters, automatic differentiation tools, such as TensorFlow [3] or PyTorch [79], can be used.

2.2.3 Model-free Reinforcement Learning

In the VI and PI algorithms in Section 2.1.3, the probability transition matrix P is required by the operators T and T_μ. As a result, efforts in system identification have to be made prior to finding the optimal control policy. Model-free Reinforcement Learning (MFRL) aims to find the optimal control policy without the need to explicitly conduct system identification. This is accomplished by using either the value-based algorithms with the Q function or the policy-based algorithms with explicit policies in the form of function approximators that map system states directly to actions.

Meanwhile, MFRL algorithms can be broadly categorized into on-policy algorithms and off-policy algorithms. In on-policy algorithms, the behavior policy (the one used to collect samples) and the target policy (the one being evaluated and improved) are identical to each other, while in off-policy algorithms, these two policies can be different. In the following sections, some common RL algorithms and concepts will be reviewed based on whether they are value-based or policy-based, but comments will be made regarding whether they are on-policy or off-policy.

Value-based Algorithms

Unlike the dynamic programming methods in Section 2.1, which using the operators T and T_μ with the probability transition matrix P, the value and Q function need to be estimated from data in MFRL. Two types of estimation methods, namely the Monte Carlo method and temporal-difference learning (TD), are commonly used to estimate $V^\pi(s_t)$ and $A^\pi(s_t, a_t)$. With the Monte Carlo method, the value function

can be approximated as follows:

$$\begin{aligned}\hat{V}^{\pi}(s_t) &= \sum_{m=1}^{M}\left[\sum_{i=t}^{\infty}\gamma^{i-t}r\left(s_i,a_i\right)\right],\\ \hat{Q}^{\pi}(s_t,a_t) &= \sum_{m=1}^{M}\left[\sum_{i=t}^{\infty}\gamma^{i-t}r\left(s_i,a_i\right)\right],\\ &\text{where } a_i\sim\pi(\cdot|s_t), i=t+1,t+2,\ldots.\end{aligned} \tag{2.23}$$

Here, M is the total number of trajectory collected, and $\hat{V}^{\pi}$ and $\hat{Q}^{\pi}$ are the value and Q function estimated from data, respectively. Although being unbiased, the Monte Carlo estimator is of high variance and requires the entire trajectory to be simulated or executed [100].

Meanwhile, TD estimator is defined as follows:

$$\begin{aligned}\hat{V}^{\pi}(s_t) &= \sum_{m=1}^{M}\left[r\left(s_i,a_i\right)+\gamma\hat{V}^{\pi_\theta}_{\xi_{\text{old}}}(s_{t+1})\right], \\ &\text{where } a_t\sim\pi(\cdot|s_t),\end{aligned} \tag{2.24}$$

$$\hat{Q}^{\pi}(s_t,a_t) = \sum_{m=1}^{M}\left[r\left(s_i,a_i\right)+\gamma\hat{V}^{\pi_\theta}_{\xi_{\text{old}}}(s_{t+1})\right], \tag{2.25}$$

Compared to the Monte Carlo method, it requires only one-step rollout, and it reduces the variance of the estimation by bootstrapping. However, TD estimator is biased due to the approximation error in $V^{\pi}(s_{t+1})$. TD(N) and TD(λ) are more advanced techniques to balance the bias and variance with hyperparameters, and the details are referred to [100]

Suppose the optimal value function V^{π^*} is given, the deterministic optimal policy can be computed by

$$\min_{a_t}\left[r(s_t,a_t)+\gamma V^{\pi^*}(s_{t+1})\right]. \tag{2.26}$$

In this formulation, the transition probability matrix $s_{t+1}\sim P(\cdot|s_t,a_t)$, which encodes the system dynamics with uncertainties, is required to solve the optimization problem.

In the meantime, the policy can be determined by

$$\min_{a_t} Q^{\pi^*}(s_t, a_t). \tag{2.27}$$

Unlike that in Eqn. (2.26), the optimization in Eqn. (2.27) does not require transition matrix anymore. SARSA [100] and Q-learning [114] are two commonly used value-based RL algorithm using Q functions, and they differ by how the optimal Q function Q^{π^*} is learned.

In SARSA, the Q function is learned as

$$Q^{\pi}(s, a) \leftarrow r + Q^{\pi}(s', a'). \tag{2.28}$$

Here, the data tuple (s, a, r, s', a') (where SARSA comes from) are strictly collected under the policy π. Since the Q^{π} is learned by only using the data collected under the policy π, SARSA is an on-policy algorithm. The policy improvement is then conducted by

$$\pi(s) \leftarrow \max_{a} Q^{\pi}(s, a). \tag{2.29}$$

On the other hand, Q-learning aims to learn optimal Q function Q^{π^*} directly:

$$Q(s, a) \leftarrow r + Q(s', \max_{a} Q(s', a)) \tag{2.30}$$

Note the tuple (s, a, r, s') without a' is needed for Q-learning. Unlike SARSA, Q-learning is off-policy as data from any policy can be used for learning the optimal Q function.

When the Q function is approximated as a function parameterized by θ, gradient descent can be used to update the parameters incrementally:

$$y_j = r_j + \gamma Q_\theta \left(s'_j, a'_j\right), \quad \text{(SARSA)} \tag{2.31}$$

$$y_j = r_j + \gamma Q_\theta \left(s'_j, \underset{a}{\operatorname{argmin}} \left(Q_\theta(s_j, a)\right)\right), \quad \text{(Q-learning)} \tag{2.32}$$

$$\theta_{i+1} = \theta_i - \alpha \nabla_\theta \left[\frac{1}{M} \sum_{i=1}^{M} \left(y_j - Q_\theta(s_j, a_j)\right)^2\right], \tag{2.33}$$

where M is the number of data tuple collected, and α is the step size. Note y_j is considered as constant once it is calculated, and it will not contribute to the gradient.

In practice, Q-learning is used more often as it is off-policy. Known issues to Q-learning are overestimation and training instability. The discussion and improvement on these topics are referred to [42, 106].

Policy-based Algorithm

With SARSA or Q-learning, implicit policy can be derived from Eqn. (2.27). The limitation of the value-based algorithms stems from solving the optimization problem. As calculating $\partial Q(s, a)/\partial a$ is difficult, if not intractable, the optimization is typically done by numerically evaluating a grid of actions and selecting the optimal value from the discrete set. This step is computationally limiting, especially when the dimension of the action space is high.

In policy gradient algorithms, a parameterized explicit policy network π_ξ is learned. Policy can be either a stochastic policy $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$ or a deterministic policy $\pi : \mathcal{S} \to \mathcal{A}$.

The learning of the stochastic policy follows the policy gradient theorem [101]:

$$\begin{aligned} \xi_{i+1} &= \xi_i - \alpha \nabla_\xi \eta(\pi_\xi) \\ &= \xi_i - \alpha \mathbb{E}_{s \sim \rho_\pi, a \sim \pi} \left[\nabla_\xi \log \pi_\xi(a|s) Q^{\pi_\xi}(s, a)\right]. \end{aligned} \tag{2.34}$$

The deterministic counterparts [93] follows:

$$\begin{aligned}\xi_{i+1} &= \xi_i - \alpha \nabla_\xi \eta(\pi_\xi) \\ &= \xi_i - \alpha \mathbb{E}_{s\sim\rho_\pi} \left[\nabla_\xi \pi_\xi(s) \, \nabla_a Q^{\pi_\xi}(s,a)|_{a=\pi_\xi(s)} \right].\end{aligned} \tag{2.35}$$

The actor-critic algorithm is a popular extension on top of the policy gradient theorem [9]. The actor is the policy network parameterized by ξ and updated by Eqn. (2.34) or Eqn. (2.35), and the critic is the Q function approximated as Q_θ^π and updated following Eqn. (2.33).

Some popular policy based algorithms include off-policy algorithms with deterministic policies [28, 60], off-policy algorithms with stochastic policies [38], and on-policy algorithms with stochastic policies [86, 88].

2.2.4 Model-based Reinforcement Learning

Despite showing promising results in simulation, MFRL presents several shortcomings. First, MFRL algorithms have low sample efficiency, i.e., a large amount of simulated data and a long training period are required to learn a good policy [112]. Second, MFRL algorithms are better suited for applications where building mathematical models is difficult. However, for many engineering applications, physics-based models of the complete or a part of systems are readily available. Embedding physics-based models can greatly improve the learning efficiency [20]. Last, in model-free methods, the policy maps the observations directly to actions. While it reduces the online computational burden, this explicit policy is more prone to constraint violations.

Model-based Reinforcement Learning (MBRL) uses data either to learn the model dynamics or combines the known model dynamics with the RL framework to improve the performance. When the model needs to be learned, the model bias of the learned

model [22], i.e., how close the learned dynamics is to the ground truth, becomes the limiting factor of the overall policy performance.

The common selections of the function approximator used for learning dynamic system models are Gaussian Processes (GPs) [21], and Ensembles of DNN [19], and Bayesian Neural Networks (BNNs) [23]. These models not only predict the mean of the model output but also track the uncertainties of the model.

In general, the learned model can assist the learning algorithm in three manners [112]. First, the learned model is used to generate synthetic data, which reduces the amount of real-world data needed for the MFRL algorithms with low sample efficiency [56, 66, 99]. Second, the learned model is used to conduct trajectory rollout, and the parameterized policy is updated by optimizing the cumulative rewards in the rollout trajectory. Optimization in long horizon makes the planning and learning more efficient [21, 57, 102]. Finally, like the previous category, the learned model is used to conduct trajectory rollout. The policy is implicitly embedded in the MPC formulation. When coupled with MPC, constraints and feasibilities can be handled directly. In addition, the value function V can be used as the terminal cost function in MPC to improve the controller performance [19, 51, 65, 103, 104]

2.3 Generative Models

Apart from policy networks, value functions and Q functions in reinforcement learning, two types of generative models will be used in Chapter 6 to provide the learning agent a feasibility guarantee and to improve the training performance.

In general, a generative model describes how data is generated via probabilistic models. Unlike its counterpart, the discriminative model that learns the conditional

probability $p(y|x)$, the generative model estimates the joint distribution $p(x, y)$ [72]. In supervised learning, where x and y are the feature inputs and labels respectively, generative models make predictions by using Bayes rules to calculate $p(y|x)$. In unsupervised learning where the labels y are missing, generative models learn the joint distribution of the feature data $p(x)$. In either case, as the name suggested, generative models can be used to generate samples from the joint distribution, and this feature is essential for the algorithm developed in Chapter 6.

In Chapter 6, the safe set, as a mechanism to provide feasibility guarantee, will be learned in an unsupervised learning fashion. The data collected during safe operation will be collected and used to train the parameterized joint distribution $p_\psi(X)$. Guided by the learned distribution, the controller can predictively choose to visit the states that are more likely to be safe. In the meantime, Variational Autoencoder (VAE) is used to alleviate the performance deterioration due to the distributional mismatch between the model-based actor and the off-policy critic. In this case, the VAE model is used to draw samples from the distribution learned to represent the model-based actor.

In this section, the mathematical backgrounds of these generative models are introduced while the detailed motivation, adoption and implementation are discussed in Chapter 6. Note the field of deep generative models is massive and attract lots of research interests in recent years. To keep focus, this section only covers what is necessary to understand the algorithm later developed in Chapter 6.

2.3.1 Kernel Density Estimation

Kernel density estimation (KDE) is a non-parametric approach to estimate the probability density function of a random variable. Compared to histograms, which is another simple non-parametric method for density function estimation, KDE results in a smooth estimation and can be applied to problems with higher dimensions.

Let x_i, $i = 1, 2, ...n$ be n independent and identically distributed (i.i.d.) samples draw from a d-variate distribution. The kernel density estimate is defined as [94]

$$\hat{f}_H(x) = \frac{1}{n} |H|^{-1/2} \sum_{i=1}^{n} K\left(H^{-1/2}(x - x_i)\right), \tag{2.36}$$

where x_i is the i^{th} dimension of the discrete input vector and d is the dimension of the input vector. Therefore,

$$x = \left(x^{(1)}, x^{(2)}, ..., x^{(d)}\right)^T, \tag{2.37}$$

$$x_i = \left(x_i^{(1)}, x_i^{(2)}, ..., x_i^{(d)}\right)^T. \tag{2.38}$$

H is the bandwidth matrix that is positive definite and of size $d \times d$. K is the kernel function, which is a symmetric multivariate density function. The selection of the kernel function can be uniform, triangular, normal or other distributions. As the model performance is not very sensitive to the selection of the kernel function type, multivariate normal distribution is commonly used. The model performance is, however, sensitive to the bandwidth matrix H as it regulates the amount and orientation of smoothing induced by the kernel function. As KDE is not explicitly used in this work, the optimal design of the bandwidth matrix and further details about the method are referred to [94].

2.3.2 Autoregressive Models

Due to the selection of the bandwidth matrix, KDE methods become less accurate under a very high-dimensional setup, commonly seen in deep learning tasks. Here, the autoregressive model is introduced for higher-dimensional problems. For any probability distribution, the joint distribution can be factorized as a product of conditional probabilities as follow:

$$\begin{aligned} p(x) &= \prod_{i=1}^{d} p(x^{(i)}|x^{(1)}, x^{(2)}, ..., x^{(i-1)}), \\ \rightarrow \log p(x) &= \sum_{i=1}^{d} \log p(x^{(i)}|x^{(1)}, x^{(2)}, ..., x^{(i-1)}), \end{aligned} \tag{2.39}$$

A straightforward way to construct a learning framework that reflects the autoregressive structure in Eqn. (2.3.2) is to use Recurrent Neural Network (RNN) [52]. In this case, as shown in Fig. 2.1, each dimension $x^{(i)}$ is fed into RNN sequentially. In case the random variable is continuous, they are discretized and turned into the one-hot representation first. Because of the sequential (autoregressive) structure, the output $y^{(i)}$ is only a function of the inputs $x^{(k)}$, $k = 1, ...i$. As a result, the conditional probability function $p_\psi(\cdot|x^{(1)}, x^{(2)}, ..., x^{(i-1)})$, parameterized by ψ, can be represented by the softmax of the output $y^{(i-1)}$. Here $y^{(i)}$ is a vector with the size of the discretization, thus its softmax represents the conditional probability at each realization.

The model is trained by minimizing the Kullback-Leibler (KL) divergence between the true data distribution and the approximated distribution:

$$\begin{aligned} &\min_{\psi} \; D_{\mathrm{KL}} \left[p^*(x)||p_\psi(x)\right] \\ =&\min_{\psi} \; \mathbb{E}_{x\sim p^*(x)} \left[-\log p_\psi(x)\right] + \text{constant} \end{aligned} \tag{2.40}$$

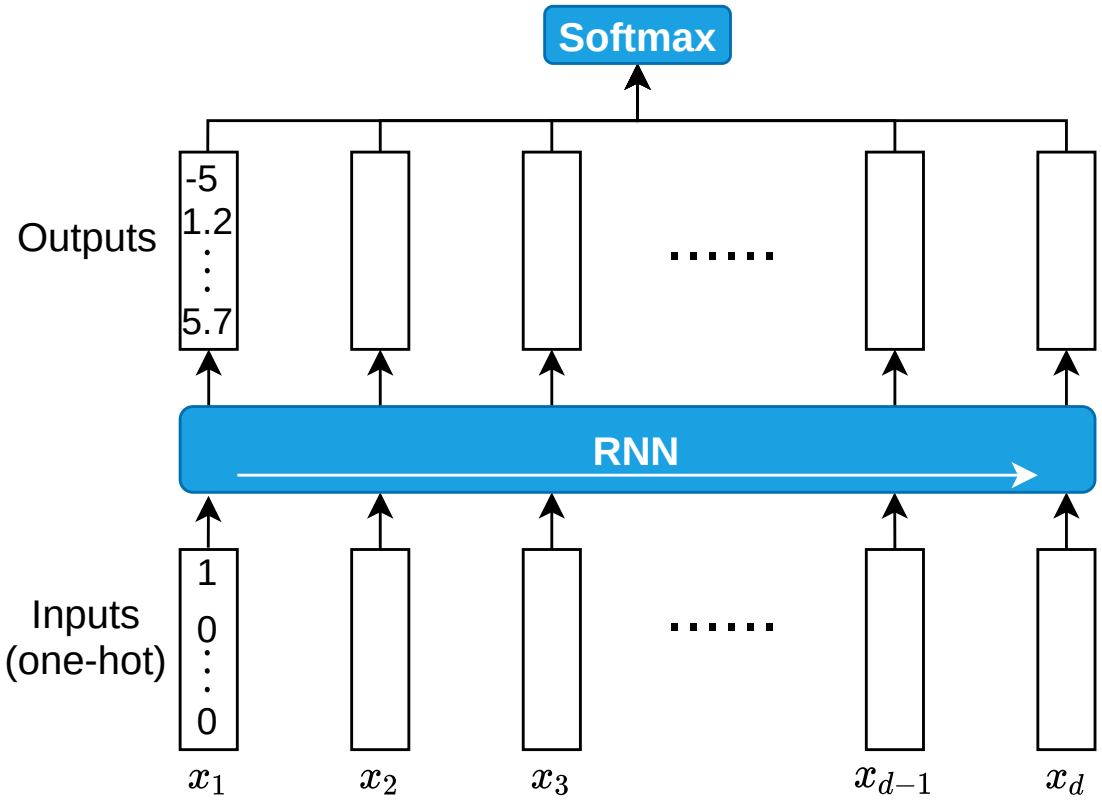


Figure 2.1: The Network Architecture of Recurrent Autoregressive Model.

Since the dimension of the variable is less than 100 ($d < 100$) in this work, the accuracy and computation requirement of the recurrent autoregressive model are both satisfying. Although the autoregressive model with RNN can learn to approximate problems with even higher dimensions, it can be slow as each dimension needs to be fed into the model sequentially for both training and evaluation tasks. To avoid the sequential evaluation while preserving the autoregressive property, full connected multilayer perceptrons (MLP) [30] and convolutional neural network (CNN) [74] with carefully designed masks are designed.

In the case that the discretization step is not desired, normalizing flows, as in [26, 27] and autoregressive flows, as in [53, 77] can be applied to directly approximate the probability density function.

2.3.3 Variational Autoencoders

Another type of generative model is Variational Autoencoder (VAE). As it does not use probability chain rule as in Eqn. (2.3.2), its inference is fast. Different from autoregressive models though, it only provides a mechanism to sample data from the underneath distribution while does not result in an explicit probability density function.

Let $X = \{x_i\}_{i=1}^N$ be some data set and Z represent a set of low-dimensional latent variables, the objective is to maximize the marginal log-likelihood:

$$\begin{aligned} \log p(X) = \sum_{i=1}^{N} \log p(x_i) &= \sum_{i=1}^{N} \int_z \log p(x_i|z) p(z) dz \\ &= \sum_{i=1}^{N} \mathbb{E}_{z\sim p(z)} \log p(x_i|z), \end{aligned} \tag{2.41}$$

As Eqn. (2.41) is in general intractable, its variational lower bound is instead maximized:

$$\begin{aligned} \mathcal{L}(\omega_1, \omega_2, X) = &- D_{\mathrm{KL}} \left(q_{\omega_1}(z|X)||p(z)\right) \\ &+ \mathbb{E}_{q_{\omega_1}(z|X)} \left[\log p_{\omega_2}(X|z)\right] \end{aligned}. \tag{2.42}$$

Here, $p(z)$ is the prior distribution that is typically assumed to be a multivariate normal distribution. $q_{\omega_1}(z|X)$ is the posterior distribution parameterized by ω_1. To analytically evaluate the KL divergence, the posterior is typically constructed as $\mathcal{N}(z|\mu_{\omega_1}(X), \Sigma_{\omega_1}(X))$. From a coding theory perspective, $q_{\omega_1}(z|X)$ and $p_{\omega_2}(X|z)$ can be considered as a probabilistic encoder and a probabilistic decoder, respectively.

As the distribution of which the expectation $\mathbb{E}_{q_{\omega_1}}[\cdot]$ evaluated under is a function of ω_1, policy gradient theorem [117] or reparameterization trick [53, 117] need be used to compute the gradient $\nabla_{\omega_1} L(\omega_1, \omega_2, X)$. The latter is often used in VAE as it typically leads to lower variance.

Chapter 3: Design of Eco-driving Virtual Environment

3.1 Introduction

Unlike supervised or unsupervised learning, where a static data set is provided for training and validation, the use of RL requires an environment to provide reactive data for closed-loop stability. During RL training, the learning agent progressively improves its policy using the data collected from the environment in response of the continuously changing policy. In principle, the RL environment can be based either on a physical system (providing experimental data) or utilize a virtual experiment consisting of a simulation model. Due to the need to explore during RL training, a virtual setting is generally utilized, especially at the initial stage of training. In addition, MFRL methods typically require a large amount of data, which can be expensive to collect from real-world. As a result, the development of an efficient and diverse simulation platform of sufficient fidelity becomes critical to RL training and testing.

In the context of eco-driving, considering the cost and complexity of generating and collecting real-world driving data and the safety concern for human operators, a simulated environment is developed and used in the work. The environment model, named EcoSIM, consists of a Vehicle Dynamics and Powertrain (VD&PT) model

and a microscopic traffic simulator. Fig.3.1 shows the schematics of EcoSIM. The controller commands three control inputs, namely the ICE torque, the BSG torque and the mechanical brake torque. The component-level torques collectively determine the HEV powertrain dynamics, the longitudinal dynamics of the ego vehicle and its location along the trip. While the states of the vehicle and powertrain models such as battery State-of-Charge (SoC), velocity and gear are readily available to the powertrain controller, the availability of the connectivity information depends on the infrastructure and the types of sensors onboard. In this study, it is assumed that a Dedicated Short Range Communication (DSRC) [6] communication devices are available onboard which is able to receive V2I and V2V information within a communication range of 500m. This allows for the vehicle to receive Signal Phasing and Timing (SPaT) information from signalized intersections, which may be utilized to optimize the approach and departure maneuvers. To this end, one of the assumptions made in this work is that the vehicle control system utilizes SPaT information only from the nearest upcoming traffic light, while information from any other traffic light is neglected, regardless of the availability. Specifically, the distance to the upcoming traffic light, its status and SPaT program are fed to the control strategy as observations. Finally, a navigation application with Global Positioning System (GPS) is assumed to be available on the vehicle, such that the locations of the origin and destination, the remaining distance, the speed limits and road grade of each segment are available at every point during the trip.

In the remaining of the chapter, the VD&PT model, the microscopic traffic simulator and the interface connecting the two are described in greater details.

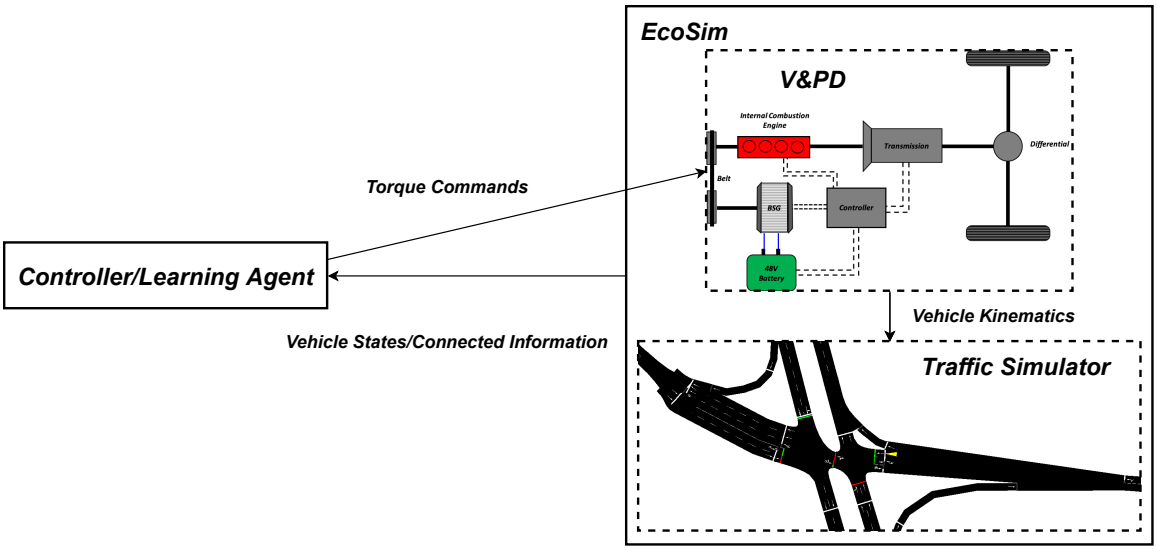


Figure 3.1: The Structure of The Environment Model

3.2 Hybrid Electric Vehicle Model

A forward-looking dynamic powertrain model is developed for fuel economy evaluation and control strategy verification over real-world routes. In this work, a P0 mild-hybrid electric vehicle (mHEV) is considered, equipped with a 48V Belted Starter Generator (BSG) performing torque assist, regenerative braking and start-stop functions. The topology of this powertrain is illustrated in Fig.3.2 and Fig.3.3, respectively. The key components of the low-frequency quasi-static model are described below.

3.2.1 BSG Model

In a P0 configuration, the BSG is connected to the engine via a belt, as shown in Eqn. 3.1. A simplified, quasi-static efficiency map $\eta(\omega_{\text{bsg},t}, T_{\text{bsg},t})$ is used to compute the electrical power output $P_{\text{bsg},t}$ in both regenerative braking and traction operating

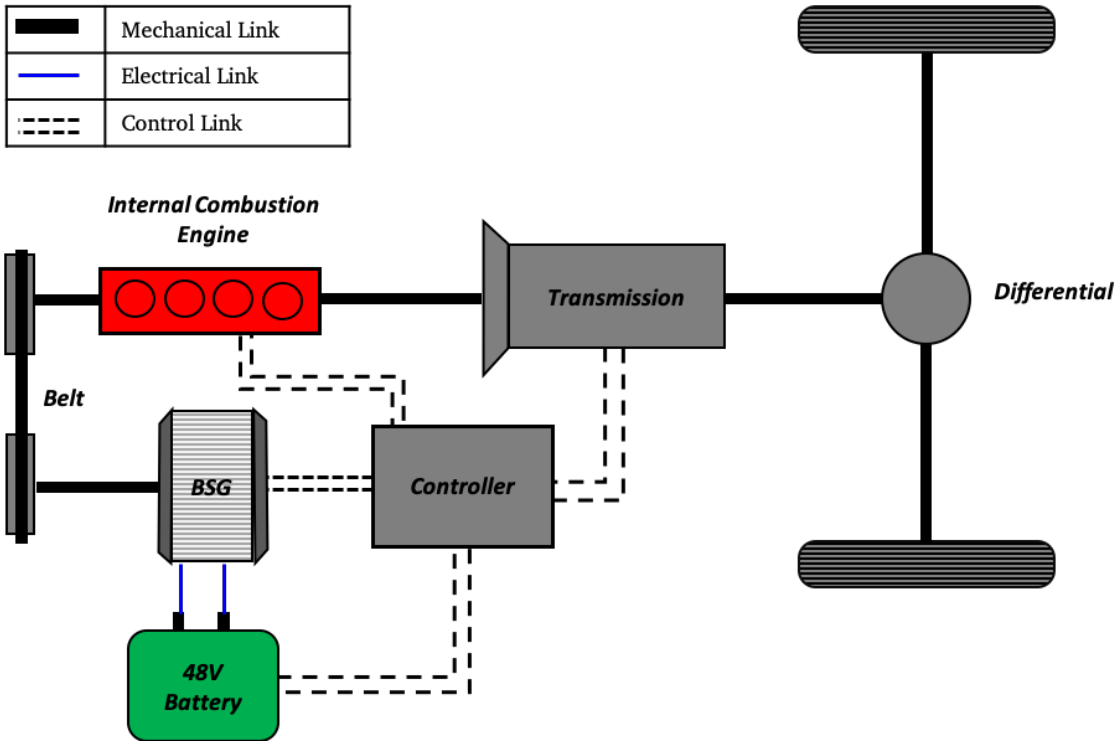


Figure 3.2: Block Diagram of P0 mHEV Topology.

modes:

$$\omega_{\mathrm{bsg},t} = \tau_{\mathrm{belt}}\omega_{\mathrm{eng},t}, \tag{3.1}$$

$$P_{\mathrm{bsg},t} = T_{\mathrm{bsg},t}\omega_{\mathrm{bsg},t}\begin{cases}\eta(\omega_{\mathrm{bsg},t}, T_{\mathrm{bsg},t}), & T_{\mathrm{bsg},t} < 0\\ \frac{1}{\eta(\omega_{\mathrm{bsg},t}, T_{\mathrm{bsg},t})}, & T_{\mathrm{bsg},t} > 0\end{cases} \tag{3.2}$$

where τ_{belt}, $\omega_{\mathrm{bsg},t}$ and $T_{\mathrm{bsg},t}$ refer to the belt ratio, the BSG angular velocity and the BSG torque, respectively.

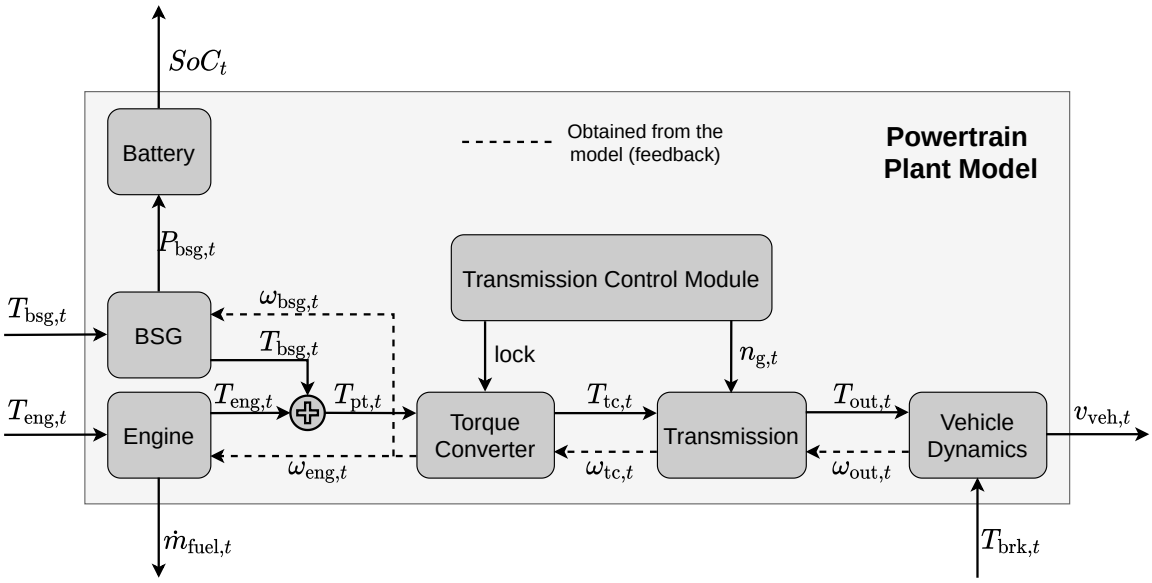


Figure 3.3: Block Diagram of 48V P0 Mild-Hybrid Drivetrain.

3.2.2 Battery Model

A zero-th order equivalent circuit model is used to model the current (I_t) dynamics. Coulomb counting [83] is used to compute the battery SoC:

$$I_t = \frac{V_{\mathrm{OC}}(SoC_t) - \sqrt{V_{\mathrm{OC}}^2(SoC_t) - 4R_0(SoC_t)P_{\mathrm{bsg},t}}}{2R_0(SoC_t)}, \tag{3.3}$$

$$SoC_{t+1} = SoC_t - \frac{\Delta t}{C_{\mathrm{nom}}}(I_t + I_{\mathrm{a}}), \tag{3.4}$$

where Δt is the time discretization, which is set to 1s in this study. The power consumed by the auxiliaries is modeled by a calibrated constant current bias I_{a}. The cell open circuit voltage $V_{\mathrm{OC}}(SoC_t)$ and internal resistance $R_0(SoC_t)$ data are obtained from the pack supplier.

3.2.3 Torque Converter Model

A simplified torque converter model is developed with the purpose of computing the losses during traction and regeneration. Here, the lock-up clutch is assumed to be always actuated, applying a controlled slip ω_{slip} between the turbine and the pump. The assumption might be inaccurate during launches, and this can be compensated by including a fuel consumption penalty in the optimization problem, associated to each vehicle launch event. This model is described as follows [63]:

$$T_{\text{tc},t} = T_{\text{pt},t}, \tag{3.5}$$

$$\omega_{\text{p},t} = \omega_{\text{tc},t} + \omega_{\text{slip}}\left(n_{\text{g},t}, \omega_{\text{eng},t}, T_{\text{eng},t}\right), \tag{3.6}$$

$$\omega_{\text{eng},t} = \begin{cases} \omega_{\text{p},t}, & \omega_{\text{p},t} > \omega_{\text{stall}} \\ \omega_{\text{idle},t}, & 0 \leq \omega_{\text{p},t} < \omega_{\text{stall}} \\ 0, & 0 \leq \omega_{\text{p},t} < \omega_{\text{stall}} \text{ and Stop} = 1 \end{cases} \tag{3.7}$$

where n_{g} is the gear number, $\omega_{\text{p},t}$ is the speed of the torque converter pump, $\omega_{\text{tc},t}$ is the speed of the turbine, ω_{stall} is the engine stall speed, $\omega_{\text{idle},t}$ is the engine idle speed, stop is a flag from the ECU indicating engine shut-off when the vehicle is stationary, $T_{\text{tc},t}$ is the turbine torque, and $T_{\text{pt},t}$ is the combined powertrain torque. The desired slip ω_{slip} is determined based on the powertrain conditions and desired operating mode of the engine (traction or deceleration fuel cut-off).

3.2.4 Transmission Model

The transmission model is based on a static gearbox, whose equations are as follows:

$$\begin{aligned}\omega_{\mathrm{tc},t} &= \tau_{\mathrm{g}}(n_{\mathrm{g},t})\omega_{\mathrm{trans},t} \\ &= \tau_{\mathrm{g}}(n_{\mathrm{g},t})\tau_{\mathrm{fdr}}\omega_{\mathrm{out},t} \\ &= \tau_{\mathrm{g}}(n_{\mathrm{g},t})\tau_{\mathrm{fdr}}\frac{v_{\mathrm{veh},t}}{R_w},\end{aligned} \tag{3.8}$$

$$T_{\mathrm{trans},t} = \tau_{\mathrm{g}}(n_{\mathrm{g},t})T_{\mathrm{tc},t}, \tag{3.9}$$

$$T_{\mathrm{out},t} = \begin{cases} \tau_{\mathrm{fdr}}\eta_{\mathrm{trans}}(n_{\mathrm{g},t}, T_{\mathrm{trans},t}, \omega_{\mathrm{trans},t})T_{\mathrm{trans},t}, & T_{\mathrm{trans},t} \geq 0 \\ \dfrac{\tau_{\mathrm{fdr}}}{\eta_{\mathrm{trans}}(n_{\mathrm{g},t}, T_{\mathrm{trans},t}, \omega_{\mathrm{trans},t})}T_{\mathrm{trans},t}, & T_{\mathrm{trans},t} < 0 \end{cases} \tag{3.10}$$

where τ_{g} and τ_{fdr} are the gear ratio and the final drive ratio, respectively. The transmission efficiency $\eta_{\mathrm{trans}}(n_{\mathrm{g}}, T_{\mathrm{trans}}, \omega_{\mathrm{trans}})$ is scheduled as a nonlinear map expressed as a function of gear number n_{g}, transmission input shaft torque $T_{\mathrm{trans},t}$ and transmission input speed ω_{trans}. $\omega_{\mathrm{out},t}$ refers to the angular velocity of the wheels. R_w and $v_{\mathrm{veh},t}$ are the radius of the vehicle wheel and the longitudinal velocity of the vehicle, respectively.

3.2.5 Vehicle Longitudinal Dynamics Model

The vehicle dynamics model is based on the road-load equation, which accounts for the effects of the tire rolling resistance, road grade, and aerodynamic drag:

$$\begin{aligned} a_{\mathrm{veh},t} &= \frac{F_{\mathrm{tr},t} - F_{\mathrm{road},t}}{M}, \\ F_{\mathrm{tr},t} &= \frac{T_{\mathrm{out},t} - T_{\mathrm{brk},t}}{R_{\mathrm{w}}}, \\ F_{\mathrm{road},t} &= \frac{1}{2}C_{\mathrm{d}}\rho_{\mathrm{a}}A_{\mathrm{f}}v_{\mathrm{veh},t}^2 + MgC_{\mathrm{r}}\cos\alpha_t v_{\mathrm{veh},t} + Mg\sin\alpha_t. \end{aligned} \tag{3.11}$$

Here, $a_{\mathrm{veh},t}$ is the longitudinal acceleration of the vehicle, T_{brk} is the brake torque applied on wheel, M is the equivalent mass of the vehicle, C_{d} is the aerodynamic

drag coefficient, ρ_{a} is the air density, A_{f} is the effective aerodynamic frontal area, C_{r} is rolling resistance coefficient, and α_t is the road grade [37].

3.2.6 Vehicle Model Verification

The forward model was calibrated and verified using experimental data from chassis dynamometer testing [73]. The key variables used for evaluating the model are vehicle velocity, battery SoC, gear number, engine speed, desired engine and BSG torque profiles, and fuel consumption. Fig. 3.4 show sample results from a model verification conducted over the FTP regulatory drive cycle, where the battery SoC and fuel consumption are compared against experimental data.

The mismatches in the battery SoC profiles can be attributed to the simplification made to the electrical system model, in which the electrical accessory loads are modeled using a constant current bias. The fuel consumption over the FTP cycle is well estimated by the model, with error on the final value less than 4% relative to the experimental data.

3.3 Microscopic Traffic Simulator

Traffic simulation models can be categorized as macroscopic, microscopic, and mesoscopic [55]. Macroscopic traffic simulation models focus on representing section-based traffic by predicting the speed and density of the traffic stream, whereas microscopic simulation models focusing on predicting the dynamics of individual vehicles in a traffic network [54]. To integrate the VD&PT model with a traffic simulation environment and study the behavior of the decision-making agent under complex driving scenarios, a large-scale microscopic traffic simulator is developed in the open source software Simulation of Urban Mobility (SUMO) [64].

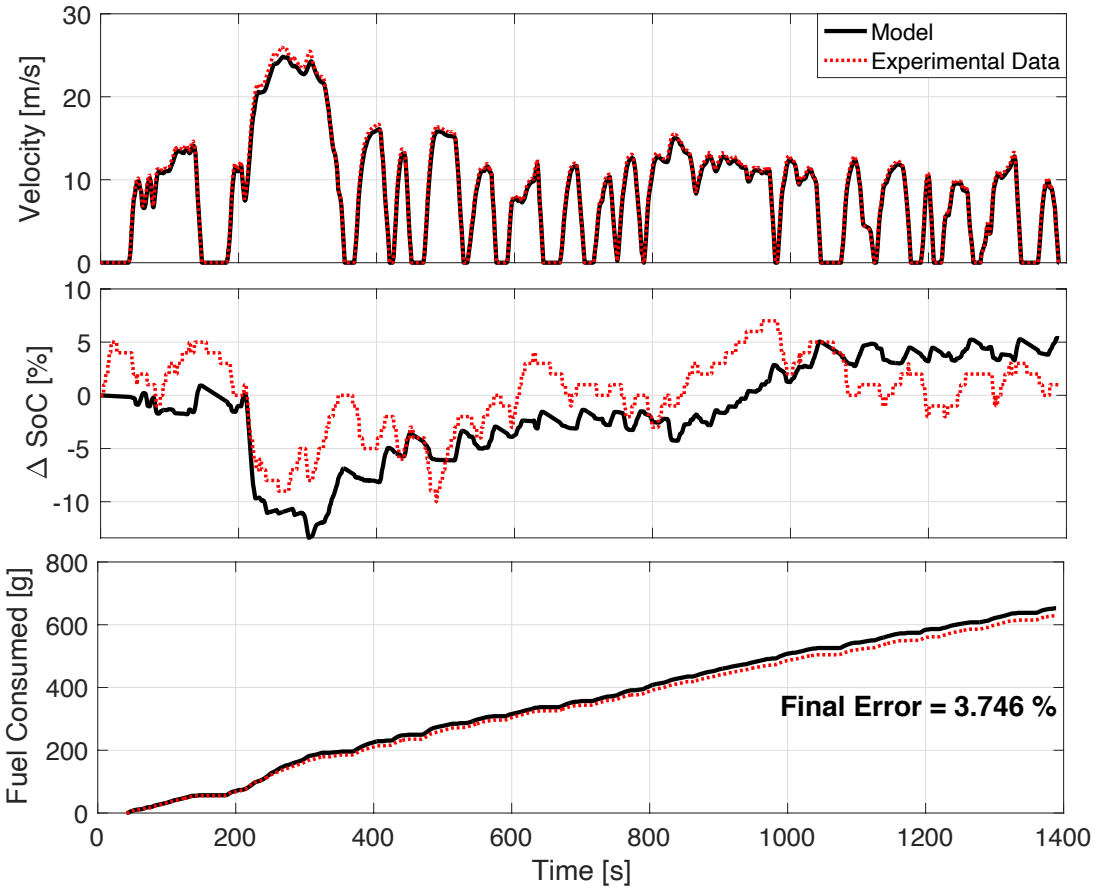


Figure 3.4: Validation of Vehicle Velocity, SoC and Fuel Consumed over FTP Cycle.

3.3.1 Large-scale Traffic Simulation Environment

In order to recreate realistic mixed urban and highway trips for training, the map of the city of Columbus, OH, USA is downloaded from the online database OpenStreetMap [75]. The map contains the length, curvature, grade and speed limit of the road segments and the detailed program of each traffic light at signalized intersections. Fig. 3.5 highlights the area ($\sim 11km$ by $11km$) covered in this study. Within this area, 10,000 random passenger car trips are generated as the training set, and the total distance of each trip is randomly distributed from 5 km to 10

km. Another 100 trips, for which the origins and the destinations are marked in red and blue in Fig. 3.5, respectively, are generated following the same distribution as the testing set. In addition, the inter-departure time of each trip follows a geometric distribution with success rate $p = 0.01$. The variation and the randomness of the trips used for training enhance the richness of the environment, which subsequently leads to a learned policy that is less subject to local minima and more easily generalizable to driving conditions beyond those considered in the training set [41].

Since the surrounding vehicles are not included in this study, the traffic density is set to zero. Evaluating how to formulate the VD&PT optimization problem in presence of lead vehicles or surrounding vehicles, and the consequent impact on safety and fuel economy, is left to future work.

3.3.2 Online Interface

The interface between the traffic simulator and the VD&PT model is established via Traffic Control Interface (TraCI) as part of the SUMO package. While the VD&PT model keeps track of several variables such as vehicle speed, acceleration, fuel consumption and battery state of charge, only the vehicle speed is required by the traffic simulator. At any given time step, the speed of the ego vehicle determined by the VD&PT model is fed to the traffic simulator. Subsequently, SUMO determines the location of the ego vehicle, updates the connectivity information such as the SPaT of the upcoming traffic light and the GPS signal and returns them to the agent as part of the observations.

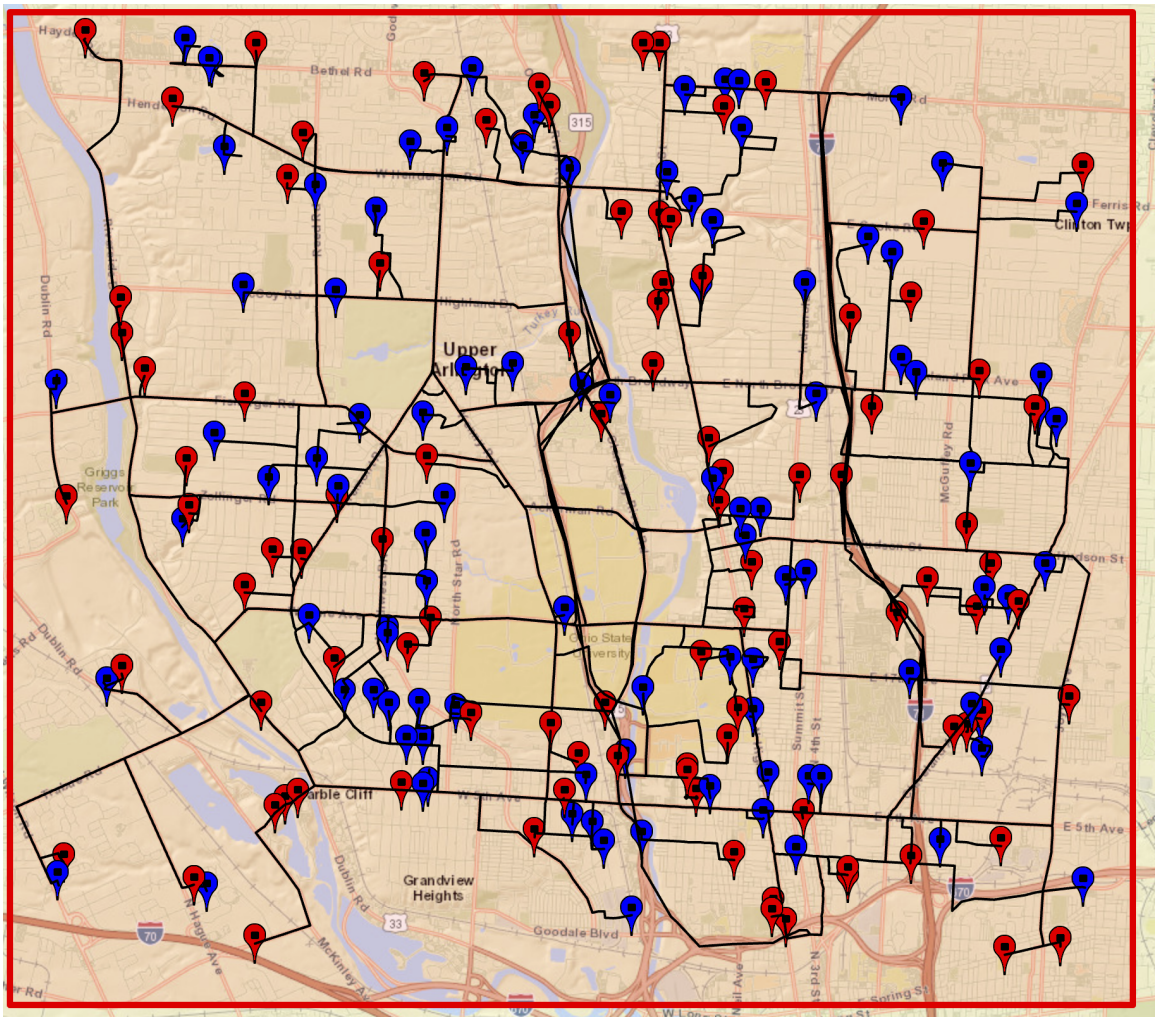


Figure 3.5: Map of Columbus, OH for DRL Training

3.4 Distributed Parallel Simulation

As both the SUMO simulation and the powertrain model are not computationally demanding, they can be executed on a single CPU thread. To accelerate data collection and to maximize hardware resource usage, particularly for the MFDRL algorithm presented in Chapter 5, a distributed parallel simulation framework is developed in

this work with the open source distributed framework Ray [70]. The detailed structure is shown in Fig. 3.6. Here, the Master contains the value function and policy in the form of function approximators, and uses the data collected by Workers to estimate the value function and improve the policy via backpropagation, which will be further explained in Chapter 5. At each learning iteration, the Master passes the most recent policy to a group of Workers and assigns simulation tasks to them. Each Worker utilizes a single CPU thread, and it contains a copy of the current policy and the environment model. Once the Workers complete the simulation tasks, they pass the collected data back to the Master, which subsequently updates the policy with the obtained data.

3.5 Summary

In this chapter, a large-scale simulation environment for the development and verification of RL-based optimal controllers for the eco-driving problem is developed. The environment consists of an experimentally validated PT&VD model of a mHEV and a microscopic traffic simulation model developed in SUMO. Finally, a distributed parallel simulation framework is developed using Ray for fast data collection.

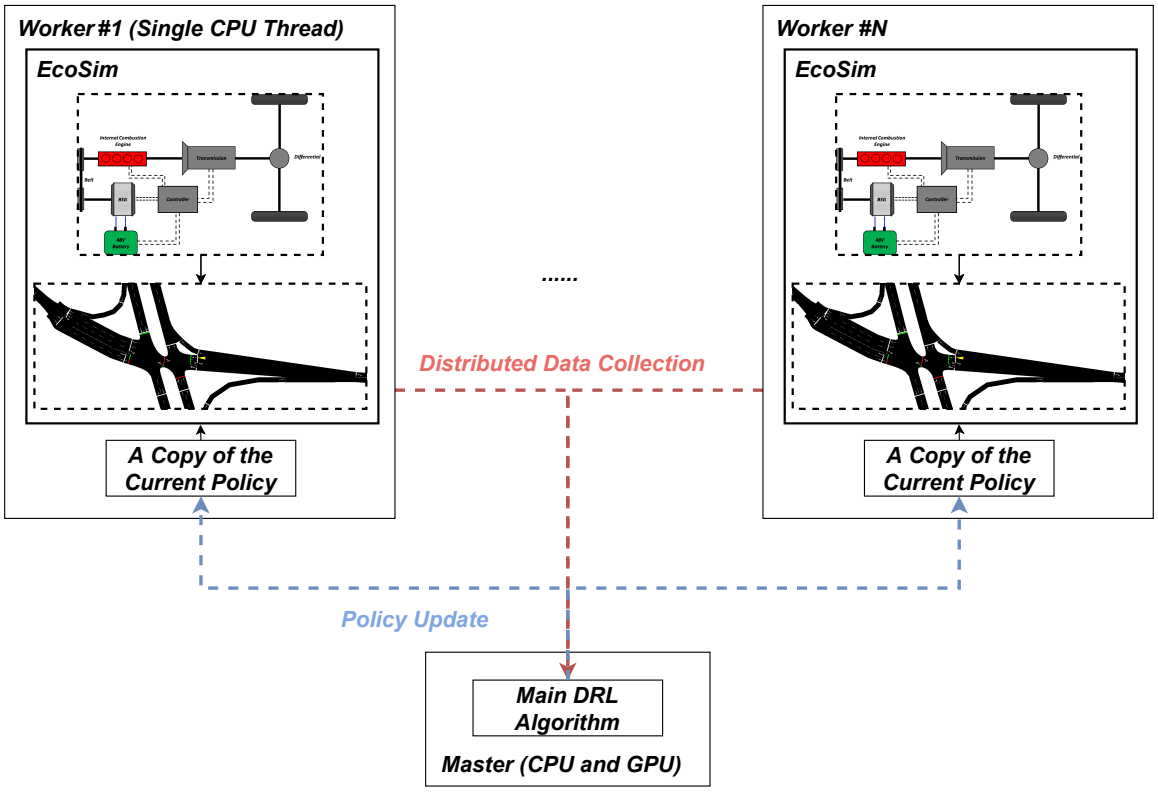


Figure 3.6: Distributed Parallel Simulation Framework

Chapter 4: Problem Formulation and Deterministic Solution

4.1 Introduction

This chapter formally introduces the eco-driving control problem formulation in both the time and spatial domains. The problem in the time domain is formulated as an infinite horizon problem since the time required to finish the trip depends on the control policy, whereas the problem in the spatial domain is formulated as a finite horizon optimization problem as the total distance of the itinerary is fixed.

In Chapter 5 and 6, two RL approaches will be presented to solve the eco-driving problem in the time domain. The advantage of formulating the problem in the time domain is that the resulting controller or policy can be used in fixed-time real-time devices directly. On the other hand, the spatial formulation is advantageous because it has finite-horizon, and the solution can be obtained via backward recursion introduced in 2.1. In general, when previewing the road condition ahead of the vehicle, the time-based information, e.g., SPaTs from signalized intersections, can be queried independently from control policy in the time domain formulation. Similarly, the distance-based information, e.g., speed limits of a route segment, can be queried independently from the control policy in the spatial domain formulation. As a counter example, the speed limit of the road in the next $5s$ depends on which road segment

does the control policy take the vehicle to, thus the information query becomes dynamic.

In this chapter, a solution method to the spatial domain formulation of the eco-driving optimization problem will be presented. This allows one to obtain a non-causal solution over a given route, which could be used as a benchmark (wait-and-see, WS) for comparing other optimization strategies. The solution method will also be used as a trajectory optimizer and integrated with the MBRL approaches proposed later in Chapter 6. Specifically, a deterministic finite horizon dynamic programming [97] applied to the spatial domain formulation of the eco-driving energy optimization problem is presented. The computational speed, from which the method in [97] notoriously suffers, is addressed by developing a novel massively parallel implementation. As a result, the proposed Parallel Deterministic Dynamic Programming (PDDP) solver offers both fast access to the non-causal solution and the possibility for real-time implementation.

4.2 Time Domain Formulation

The objective of the eco-driving optimization problem is to minimize the weighted sum of fuel consumption and travel time over a route defined between two designated

locations. The optimal control problem is formulated as follows:

$$\min_{T_{\text{eng}},T_{\text{bsg}},T_{\text{brk}}} \mathbb{E}\left[\sum_{t=0}^{\infty}\left(\lambda \dot{m}_{\text{fuel},t} + (1-\lambda)\right)\Delta t \cdot \mathbb{I}\left[s_t < s_{\text{total}}\right]\right] \tag{4.1a}$$

$$\text{s.t. } SoC_{t+1} = f_{\text{batt}}(v_{\text{veh},t}, SoC_t, T_{\text{eng},t}, T_{\text{bsg},t}, T_{\text{brk},t}) \tag{4.1b}$$

$$v_{\text{veh},t+1} = f_{\text{veh}}(v_{\text{veh},t}, SoC_t, T_{\text{eng},t}, T_{\text{bsg},t}, T_{\text{brk},t}) \tag{4.1c}$$

$$T_{\text{eng}}^{\min}(\omega_{\text{eng},t}) \leq T_{\text{eng},t} \leq T_{\text{eng}}^{\max}(\omega_{\text{eng},t}) \tag{4.1d}$$

$$T_{\text{bsg}}^{\min}(\omega_{\text{bsg},t}) \leq T_{\text{bsg},t} \leq T_{\text{bsg}}^{\max}(\omega_{\text{bsg},t}) \tag{4.1e}$$

$$I^{\min} \leq I_t \leq I^{\max} \tag{4.1f}$$

$$SoC^{\min} \leq SoC_t \leq SoC^{\max} \tag{4.1g}$$

$$SoC_T \geq SoC_{\text{F}} \tag{4.1h}$$

$$0 \leq v_{\text{veh},t} \leq v_{\text{lim},t} \tag{4.1i}$$

$$(t, s_t) \notin \mathcal{S}_{\text{red}} \tag{4.1j}$$

Here, f_{batt} and f_{veh} represent the battery and vehicle dynamics introduced in Section 3.2. $\dot{m}_{\text{fuel},t}$ is the instantaneous fuel consumption at time t, and λ is the weighing factor between the fuel consumption and the travel time. Unless the destination is reached, i.e., the traveled distance s_t is greater than the total distance of the trip s_{total}, an additional cost $(1-\lambda)$ is assigned. Eqn. (4.1d) to (4.1f) are the constraints imposed by the powertrain components. Eqn. (4.1g) and Eqn. (4.1h) are the constraints on the instantaneous battery SoC and terminal SoC for charge sustaining. Here, the subscript T represents the time at which the vehicle reaches the destination. SoC_{min}, SoC_{max} and SoC_{F} are set to 30%, 80% and 50%. Eqn. (4.1i) and (4.1j) are the constraints imposed by the specific features of the route travelled. Here, $v_{\text{lim},t}$ represents the speed limit of the route segment on which the ego vehicle is at the

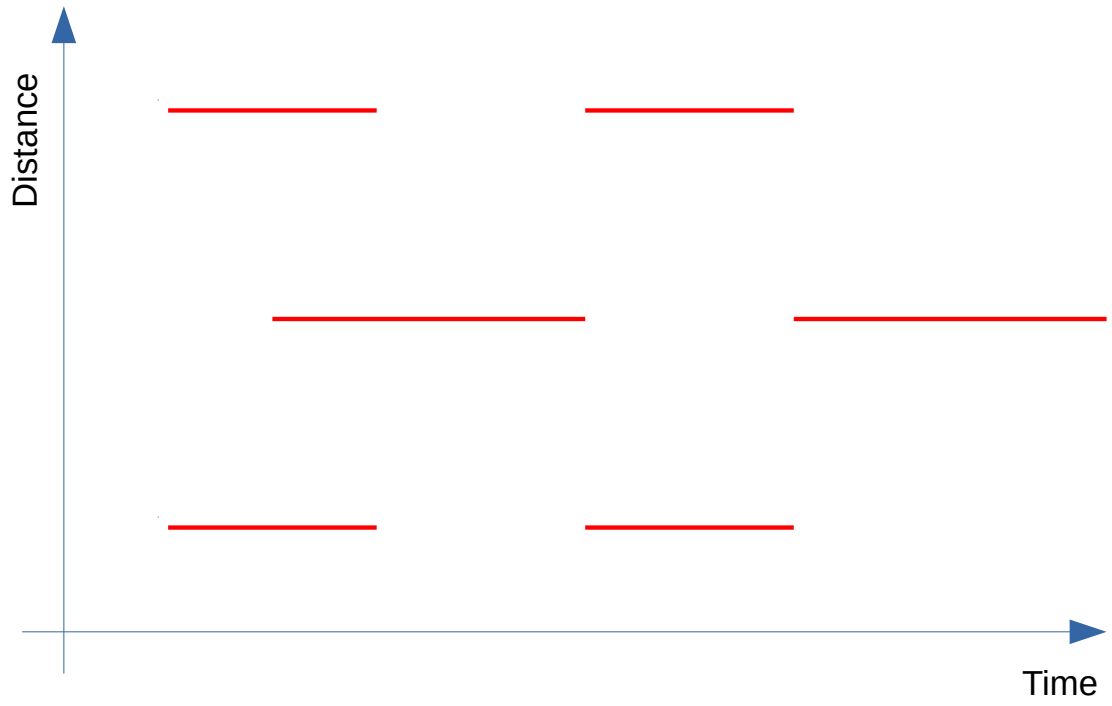


Figure 4.1: Feasible Set Imposed by Traffic Lights

given time t. The set $\mathcal{S}_{\text{red}}$ represents the set in which the traffic light at the certain location is in the red phase, which can be represented as a time/distance constraint as in Fig. 4.1.

As the controller can only accurately predict the future driving condition in a relatively short range due to the limited connectivity range and onboard processing power, the stochastic optimal control formulation is deployed to accommodate the future uncertainties. Specifically, since the surrounding vehicles are not considered in the study, the main source of uncertainty comes from the change of route, the unknown SPaTs and the road conditions, such as the speed limits and the distance between signalized intersections, beyond the connectivity range.

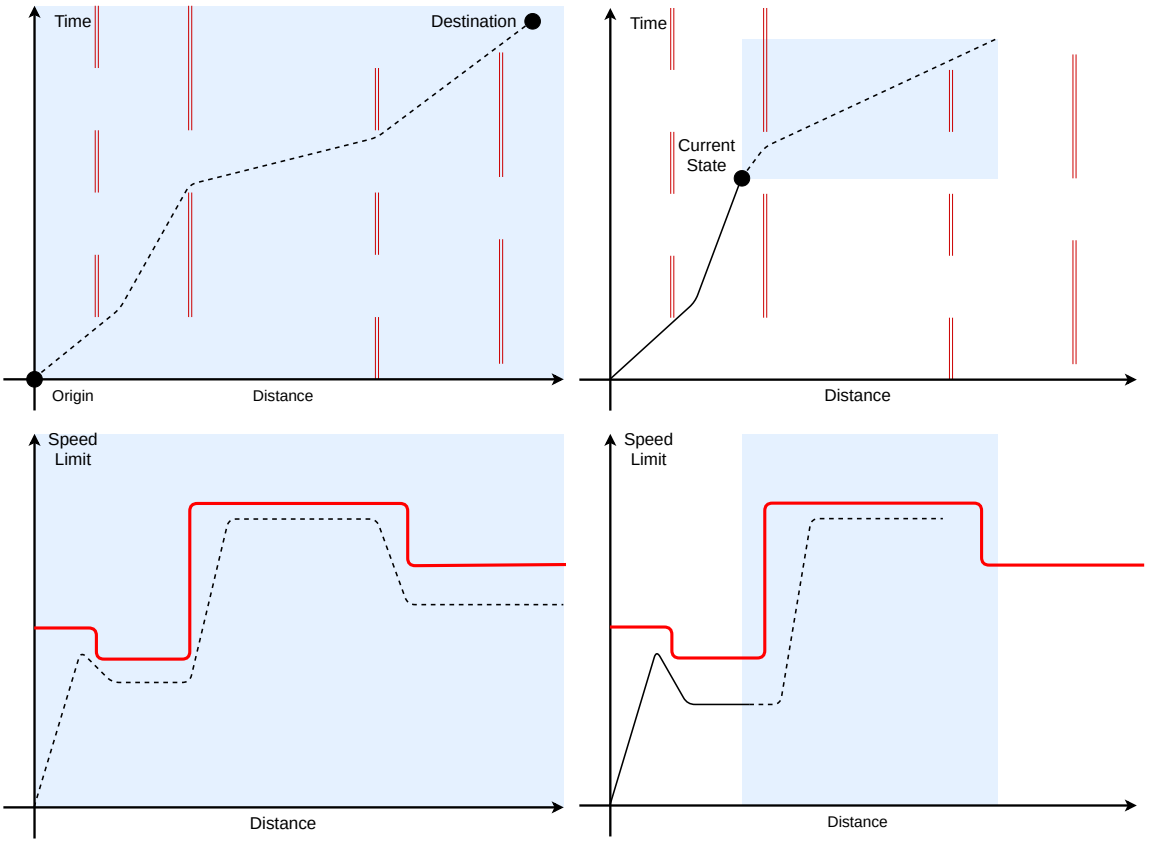
4.3 Spatial Domain Formulation

While the RL methods developed later in this dissertation primarily utilize the time domain formulation, the spatial domain formulation provides a recursive but not iterative mechanism to compute the deterministic solution as in [97].

The deterministic solution requires the route information, including the traffic light status of each signalized intersection and the route segment information, to be known *a priori*. When the full-route information is provided as in Fig. 4.2a, the wait-and-see (WS) [14] solution can be obtained. Here, the blue shaded area indicates that the future time and spatial conditions are assumed to be known perfectly. The solid line indicates the history of the states while the dashed line indicates the predicted trajectory from optimization. While the solution serves as an excellent benchmark for optimality, its usage for real-time control is, however, limited due to the long prediction horizon. In reality, the itinerary is subject to change, and route and SPaT information can be unavailable or inaccurate from distance.

On the other hand, the assumption that the route information can be previewed within a certain range, e.g., 200 m ahead of the current ego-vehicle position, is more realistic as in Fig. 4.2b. In practice, the optimization problem for either the full route or a shorter preview distance remains the same. In this chapter, the full route solution will be presented, and the approach will be later used in Chapter 6 as a solver to the optimization problem in a shorter predictive horizon.

The objective of the eco-driving problem, formulated in the spatial domain, is to minimize the fuel consumed by the vehicle over an entire route consisting of N



(a) Full-route Formulation in the Spatial Domain

(b) Rollout Formulation in the Spatial Domain

Figure 4.2: The Spatial Domain Formulation of Eco-driving Problem

steps [85]:

$$\min_{T_{\text{eng},s},T_{\text{bsg},s}} \sum_{s=1}^{s=N} (\lambda \cdot \dot{m}_{\text{f},s} + (1-\lambda)) \cdot \Delta t_s, \tag{4.2}$$

where Δt_s is the travel time per step computed as follows:

$$\Delta t_s = \frac{\Delta d}{\bar{v}_s}, \tag{4.3}$$

where Δd is the distance step, i.e., $\Delta d = d_{s+1} - d_s$, calculated from d_s the distance traveled along the route, and

$$\bar{v}_s = \frac{v_s + v_{s+1}}{2} \tag{4.4}$$

is the average velocity over a distance step. The state and action space are subject to the constraints as described in Section 4.2.

An additional benefit of the spatial formulation is that it inherently lends itself to the incorporation of distance-based route features, such as speed limits, grade, traffic light and stop sign locations. However, the spatial-domain formulation would make it difficult to incorporate time-based information such as SPaT received from V2I communication. This requires one to augment the state space with time as an additional state, as will be elaborated later in this section.

Consider a case where the ego vehicle is approaching a signalized intersection and receives a traffic light phase (red/green) through V2I communication. To avoid boundary cases and ensure feasibility, the yellow phase is assumed to be a part of the green phase if the distance to the traffic light is close to the critical braking distance as formulated in [35]. The time remaining in the current phase or transitioning to the next phase would change as the vehicle approaches the signalized intersection. In this section, $t_{\text{GR},s}$ and $t_{\text{RG},s}$ will represent time for a green-red and red-green transition respectively at position s, as shown in the Fig. 4.3. In this dissertation, it is assumed

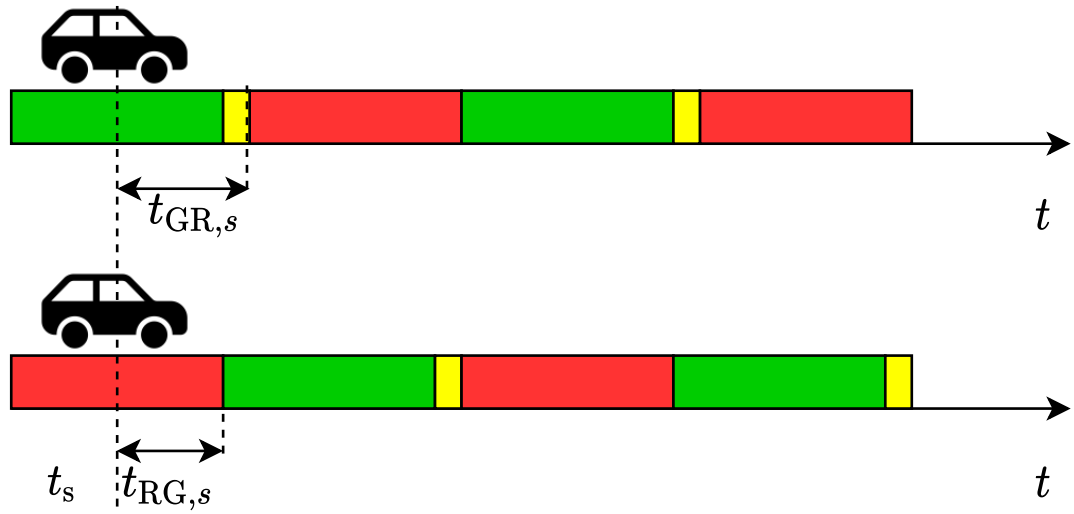


Figure 4.3: Possible Cases of Traffic Light Phase Encountered by the Vehicle.

that the positions of all the traffic lights in the route are known a priori from a navigation system, and contained in the set $\mathcal{D}_{\mathrm{TL}}$.

Assume that the discretized state dynamics for the eco-driving problem has the following form:

$$x_{s+1} = f_s(x_s, u_s), \quad s = 1, \cdots, N-1. \tag{4.5}$$

where $x_s \in \mathcal{X}_s \subseteq \mathbb{R}^p$ and $u_s \in \mathcal{U}_s \subseteq \mathbb{R}^q$ are the state and control action, respectively.

In this dissertation, the state variables are the vehicle velocity v_s, battery SoC_s and travel time t_s. The control actions are the engine torque $T_{\mathrm{eng},s}$, BSG torque $T_{\mathrm{bsg},s}$ and brake torque $T_{\mathrm{brk},s}$. To reduce the control dimension, the brake torque $T_{\mathrm{brk},s}$ is integrated into the engine torque T_{eng}, i.e., a negative engine torque represents a braking torque from the mechanical brake.

$$x_s = [v_s, SoC_s, t_s]^T \tag{4.6a}$$

$$u_s = [T_{\mathrm{eng},s}, T_{\mathrm{bsg},s}]^T \tag{4.6b}$$

The equations describing the state dynamics $f_s(x_s, u_s)$ are:

$$a_s = \frac{F_{\mathrm{tr},s} - F_{\mathrm{road},s}(v_s)}{M} \tag{4.7a}$$

$$v_{s+1} = \sqrt{v_s^2 + 2\Delta d a_s} \tag{4.7b}$$

$$SoC_{s+1} = SoC_s - \frac{\Delta d}{\bar{v}_s} \cdot \frac{\bar{I}_{\mathrm{bat},s}}{C_{\mathrm{nom}}} \tag{4.7c}$$

$$t_{s+1} = \begin{cases} t_s + t_{\mathrm{RG},s} & , s \in \mathcal{D}_{\mathrm{TL}} \text{ and } \bar{v}_s = 0 \\ t_s + \frac{\Delta d}{\bar{v}_s} & , s \notin \mathcal{D}_{\mathrm{TL}} \end{cases} \tag{4.7d}$$

Here, $F_{\mathrm{tr},s}$ and $F_{\mathrm{road},s}$ are the tractive and road load forces, respectively, introduced in Section 3.2. a_s is the acceleration of the ego vehicle. $\bar{I}_{\mathrm{bat},s}$ is the average current evaluated over a distance step. C_{nom} is the nominal battery capacity, t_s is the travel time at a position s. Intuitively, Eqn. (4.7d) shows the time is teleported to the end of the red phase when the vehicle stops at the traffic light.

4.4 Deterministic Solution of the Eco-driving Problem

In this section, the DP algorithm solving the spatial domain formulation for the WS optimal solution is presented. The same solver will be revisited later in Chapter 6 as a fast online trajectory optimization solver. The eco-driving problem is formulated as a finite horizon optimization problem where the full route of N steps is solved:

$$\begin{aligned} \mathcal{J}^*(x_s) &= \min_{\{\mu_s\}_{s=0}^{N}} \sum_{s=0}^{N} c(x_s, \mu_s(x_s)) + c_{\mathrm{T}}(x_N), \\ c(x_s, \mu_s(x_s)) &= (\alpha \cdot \dot{m}_{\mathrm{f},s}(x_s, \mu_s(x_s)) + (1-\alpha)) \cdot \Delta t_s \end{aligned} \tag{4.8}$$

where $\mu_s : \mathcal{X}_s \to \mathcal{U}_s$ is the admissible control policy at the step k in the prediction horizon, $c : \mathcal{X}_s \times \mathcal{U}_s \to \mathbb{R}$ is the stage cost function defined as the weighted average of the fuel consumption and the travel time, $c_{\mathrm{T}} : \mathcal{X}_s \to \mathbb{R}$ is the cost function at the final step.

The state space and action space are subject to following constraints:

$$v_s \in [v_s^{\min}, v_s^{\max}], \tag{4.9a}$$

$$SoC_s \in [SoC_s^{\min}, SoC_s^{\max}], \tag{4.9b}$$

$$t_s \in \mathcal{T}_{\mathrm{G},s}, \tag{4.9c}$$

$$a_s \in [a^{\min}, a^{\max}], \tag{4.9d}$$

$$T_{\mathrm{eng},s} \in [T_{\mathrm{eng}}^{\min}(v_s), T_{\mathrm{eng}}^{\max}(v_s)], \tag{4.9e}$$

$$T_{\mathrm{bsg},s} \in [T_{\mathrm{bsg}}^{\min}(v_s), T_{\mathrm{bsg}}^{\max}(v_s)], \tag{4.9f}$$

where $s = 0, 1, \ldots, N$, $v_s^{\min}, v_s^{\max}$ are the minimum and maximum route speed limits respectively, $SoC_s^{\min}, SoC_s^{\max}$ are the static limits applied on battery SoC, $\mathcal{T}_{\mathrm{G},s}$ represents the feasible set on the travel time at the signalized intersection, $a^{\min}, a^{\max}$ represent the limits imposed on the acceleration for drive comfort, $T_{\mathrm{eng}}^{\min}(v_s), T_{\mathrm{eng}}^{\max}(v_s)$ are the minimum and maximum engine torque limits, and $T_{\mathrm{bsg}}^{\min}(v_s), T_{\mathrm{bsg}}^{\max}(v_s)$ are the minimum and maximum BSG torque limits, respectively. To ensure SoC neutrality over the entire itinerary, a terminal constraint $SoC_0 = SoC_N$ is applied.

Consider a case where the entire itinerary with a N horizon comprises of three signalized intersection located at $k_{\mathrm{TL},n}$, $n = 1, 2, 3$ as shown in Fig. 4.4. The travel time at the beginning of the trip is 0 and a maximum travel time t_{f} is imposed at the end of the horizon. t_{f} is chosen such that the ego vehicle does not impede the traffic flow, which can be computed as an expectation from multiple historical trips along the same route [48]. In this dissertation, t_{f} is assumed to be 800s for the sampled trips ranging from $5km$ to $10km$ to ensure feasibility. The status of the traffic light is then sampled into an indicator vector $\mathbb{I}_{\mathrm{G},s}$ of size $t_{\mathrm{f}}/\Delta t$ with each element representing no traffic light or green phase with 1 and red phase with 0. Note that Δt here is the

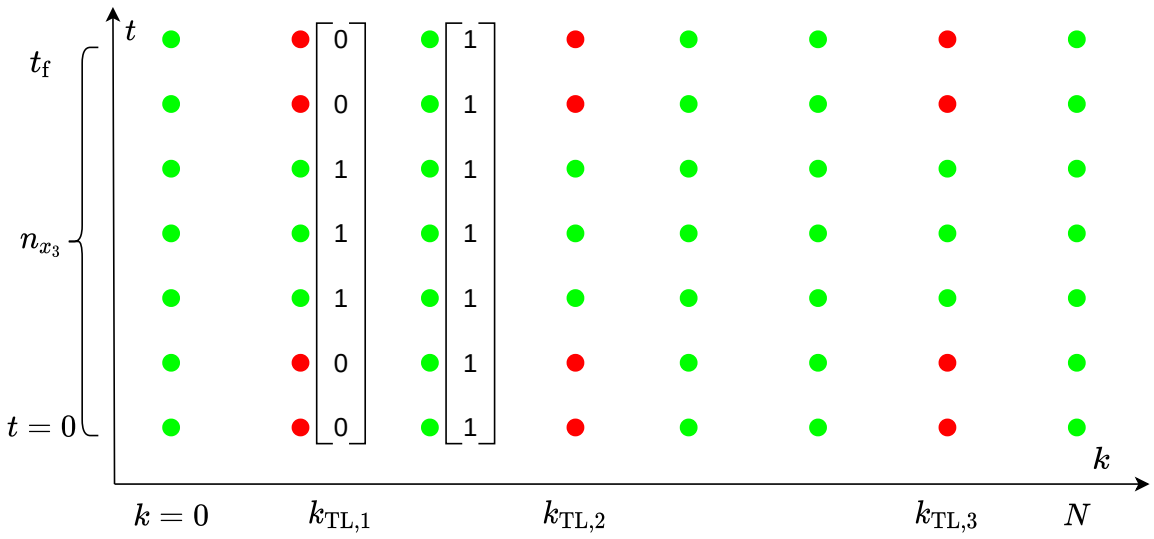


Figure 4.4: Travel Time Constraint at the Signalized Intersection.

constant time discretization for sampling. The feasible set $\mathcal{T}_{\mathrm{G},s}$ is then defined as follows:

$$\mathcal{T}_{\mathrm{G},s} = \left\{ t : \Delta t \cdot z \middle| \left(\mathbb{I}_{\mathrm{G},s}\right)_z = 1, z = 1, ..., t_\mathrm{f}/\Delta t \right\}, \tag{4.10}$$

where $\left(\mathbb{I}_{\mathrm{G},s}\right)_z$ represents the z^{th} element of the vector. The size of the sampled traffic light status vector is chosen to be the same as the grid size of the DP solver in the time dimension, which will be explained in the next section.

Knowing the driving conditions over the entire itinerary, the optimization problem can be solved offline via DP. Following the nomenclature in [98], the optimal policy μ_s^*, along with the optimal cost-to-go function $\mathcal{J}_s : \mathcal{X} \to \mathbb{R}$, $\forall s = 0, 1 \ldots N$ can be

calculated through backward recursion as follows:

$$\mathcal{J}_N(x) = c_{\mathrm{T}}(x) + \phi_{\mathrm{T}}(x), \tag{4.11a}$$

$$\mathcal{F}_s(x,u) = c_s(x,u) + \phi_s(x) + \mathcal{J}_{s+1}(f_s(x,u)), \tag{4.11b}$$

$$\mu_s^* = \underset{\mu_s}{\operatorname{argmin}}\, \mathcal{F}_s(x, \mu_s(x)), \tag{4.11c}$$

$$\mathcal{J}_s(x) = \mathcal{F}_s(x, \mu_s^*(x)), \tag{4.11d}$$

where c_s and c_{T} are the discretized stage and terminal cost function respectively; ϕ_s and ϕ_{T} are penalty functions introduced to ensure that the trajectory stays feasible. The benefit of using the deterministic dynamic programming is that it provides the closed-loop optimal policy that inherently adds robustness against approximated plant dynamics or other modeling errors [39]. Further, DP can solve highly nonlinear and hybrid problems while ensuring constraint satisfaction.

4.5 Parallel Deterministic Dynamic Programming

For the eco-driving optimization problem, a Serial Deterministic DP (SDDP) solver can be constructed, where each state-action combination is evaluated in a 5-layer nested loop, as shown in Fig. 4.5. The powertrain model takes the current vehicle speed, the engine torque and the BSG torque as the inputs and calculates the change in vehicle speed Δv_s, the change in time Δt_s, BSG power demand P_{bat} (input to the battery model) and the stage cost $c(x_s, \mu(x_s))$. The battery model takes P_{bat} and calculates the change in battery state-of-charge ΔSoC for the nodes in the SoC grid. If an infeasibility is encountered for a state-action combination, the cost-to-go is set to a value that is orders of magnitude larger than the stage cost.

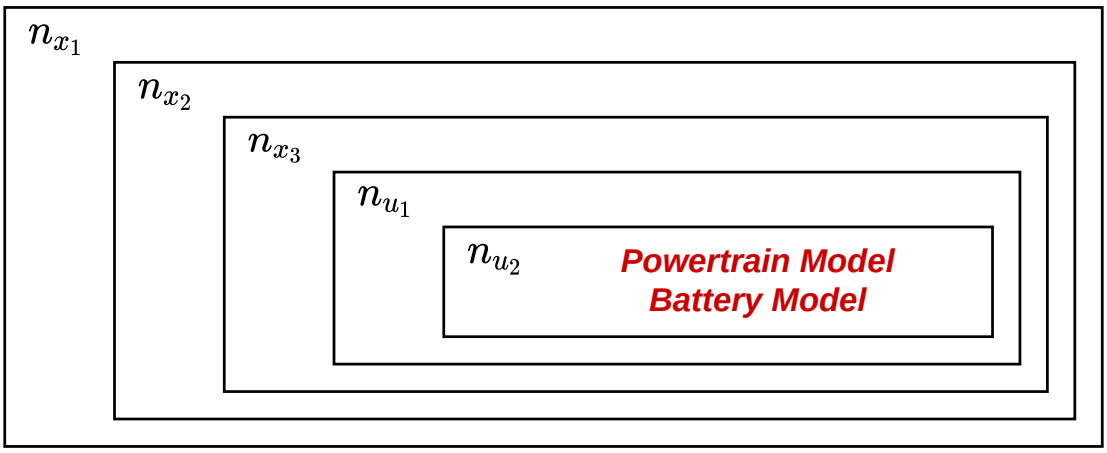


Figure 4.5: Serial DP Architecture

Table 4.1: Discretization Setup for Deterministic Dynamic Programming

Variable	Unit	Discretization	Min	Max
v_s	m/s	1	0	35
SoC_s	N/A	0.02	0.3	0.8
t_s	s	1	0	$s_N/10$
$T_{\text{eng},s}$	N/m	10	-260	210
$T_{\text{bsg},s}$	N/m	2.5	-40	40

To maintain a reasonable accuracy, the grid size of each dimension is shown in Tab. 4.1. As a result, the number of executions of the innermost loop is on the order of 10^7 for each DP backward recursion step. Meanwhile, the full-route solution requires running the backward recursion step for N times. Without any parallelization, hours are required to compute the full-route DP solution. Such computational speed significantly slows down the benchmarking and design process. In addition, as the same solver is used in Chapter 6 for online trajectory optimization, the excessive computational time eliminates the possibility for any online application.

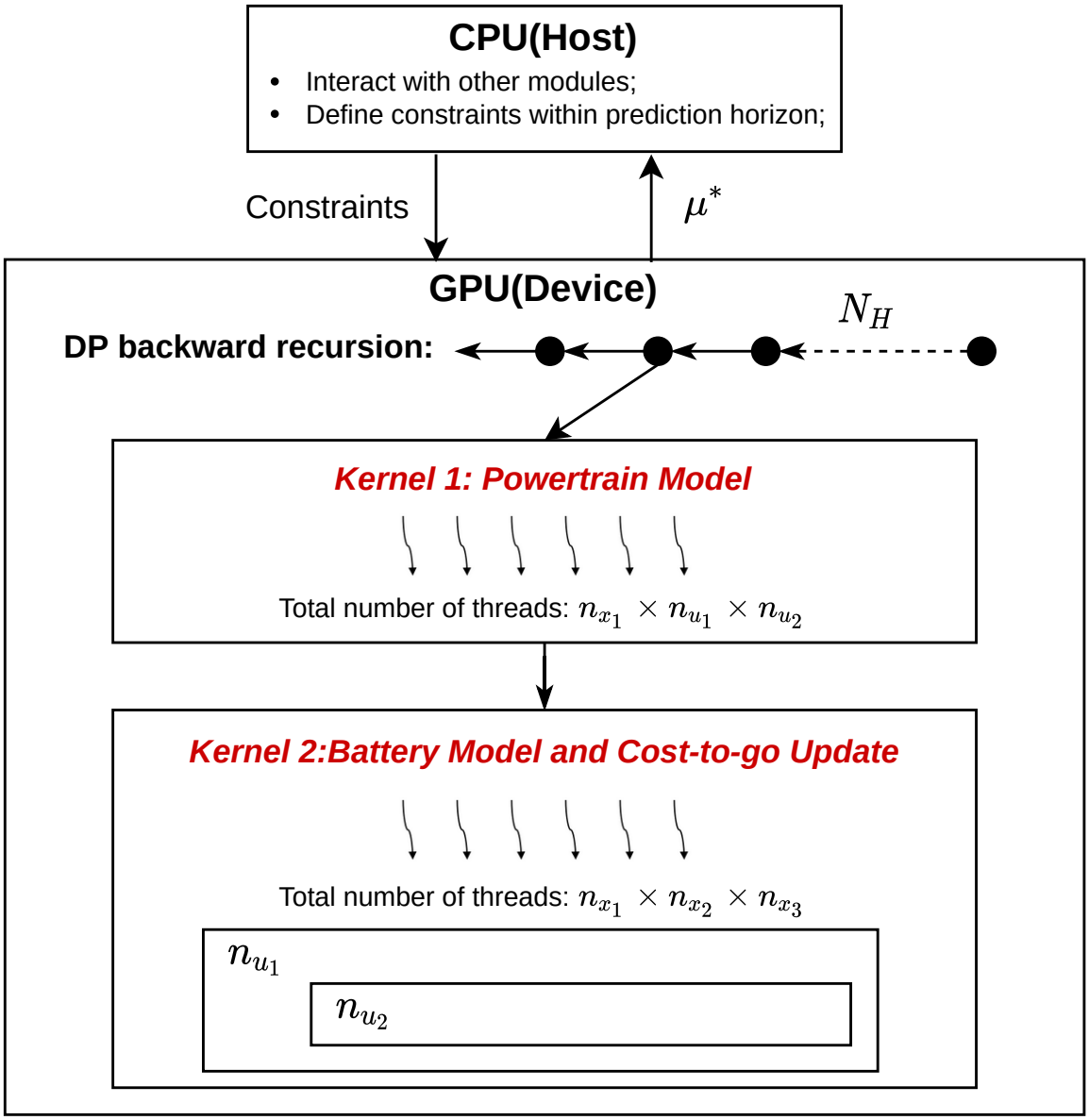


Figure 4.6: Parallel DP Architecture

To address the issue, a parallel architecture is proposed in this dissertation and then implemented on an NVIDIA GPU with CUDA C++ programming language. Fig. 4.6 shows the architecture of the parallel implementation.

With the comparable expense and power consumption, a CPU is faster at performing complex and long computing sequences thanks to the more resources dedicated to memory caching and control flow, whereas a GPU saves more resources on processors that can be run in parallel [1]. In CUDA programming, CPU and GPU are referred to as Host and Device, respectively, while Kernel is the function that is launched by Host and executed in parallel by Device. Each thread represents a sub-process running on GPU with unique inputs. Depending on the hardware specification and computational requirement of the task in each thread, the numbers of available threads vary. In general, the computational advantage shows when thousands of threads are launched in parallel. To have an efficient implementation, the data transfer frequency between host and device should be minimized and, in the current framework, such data transfer occurs only once per DP optimization call.

In the first Kernel, each thread represents a combination of $(v_s, T_{\text{eng},s}, T_{\text{bsg},s})$ on the 3D grid and calculates the corresponding Δv_s, Δt_s, $P_{\text{bat},s}$, $c(x_s, u_s)$ and $\text{fea}(x_s, u_s)$ for the second Kernel. Here, $\text{fea}(x_s, u_s)$ is a binary output representing whether the state-action combination is feasible or not. Each thread launched by the second Kernel represents a combination of the three states (v_s, SoC_s, t_s) and loops through the control inputs, determines ΔSoC based on P_{bsg} from the first Kernel, and finally determines whether to update the cost-to-go matrix and the optimal control matrix following [98]. As Eqn. (4.11c) requires minimization to find the optimal control action, the nested loops for control actions are kept to avoid race conditions [1].

[1]A race condition is a situation when multiple threads can access the same shared data at the same time during multi-threading in coding. [17]

Table 4.2: Computation Time Comparison between Serial and Parallel Implementations for Full-route Solution

Solver	Mixed Condition	Urban Condition
SDDP	3470 s	3601 s
PDDP	9 s	10 s

4.6 Results

The optimality of the WS solution with the aforementioned DP solver is shown in Fig. 4.7, representing mixed driving condition and Fig. 4.8 representing urban driving with high traffic light density.

While the comparison between the non-causal WS solution and other causal controllers is left to the later chapters, the merits of the solution are shown intuitively here. In both cases, the solution intentionally drives below the speed limits and successfully plans through all the signalized intersections without violating any traffic light constraint. In particular, the solution in Fig. 4.7 passes all the signalized intersections without coming to full stop at all. In the second half, when there are no traffic lights for several kilometers of drive, the solution starts to drive at the exact speed limit to reduce travel time. Meanwhile, the solution takes advantage of the final braking events to efficiently charge the battery back to $SoC = 0.5$ to satisfy the terminal SoC constraint.

In Table 4.2, the solver times of the serial and parallel implementations are compared on the two trips shown above. Here, the serial and parallel implementations are executed on a desktop PC with a 2.9 GHz Intel Core i7 CPU and an NVIDIA RTX 2080 GPU, respectively. Performing the same task, PDDP is able to reduce the

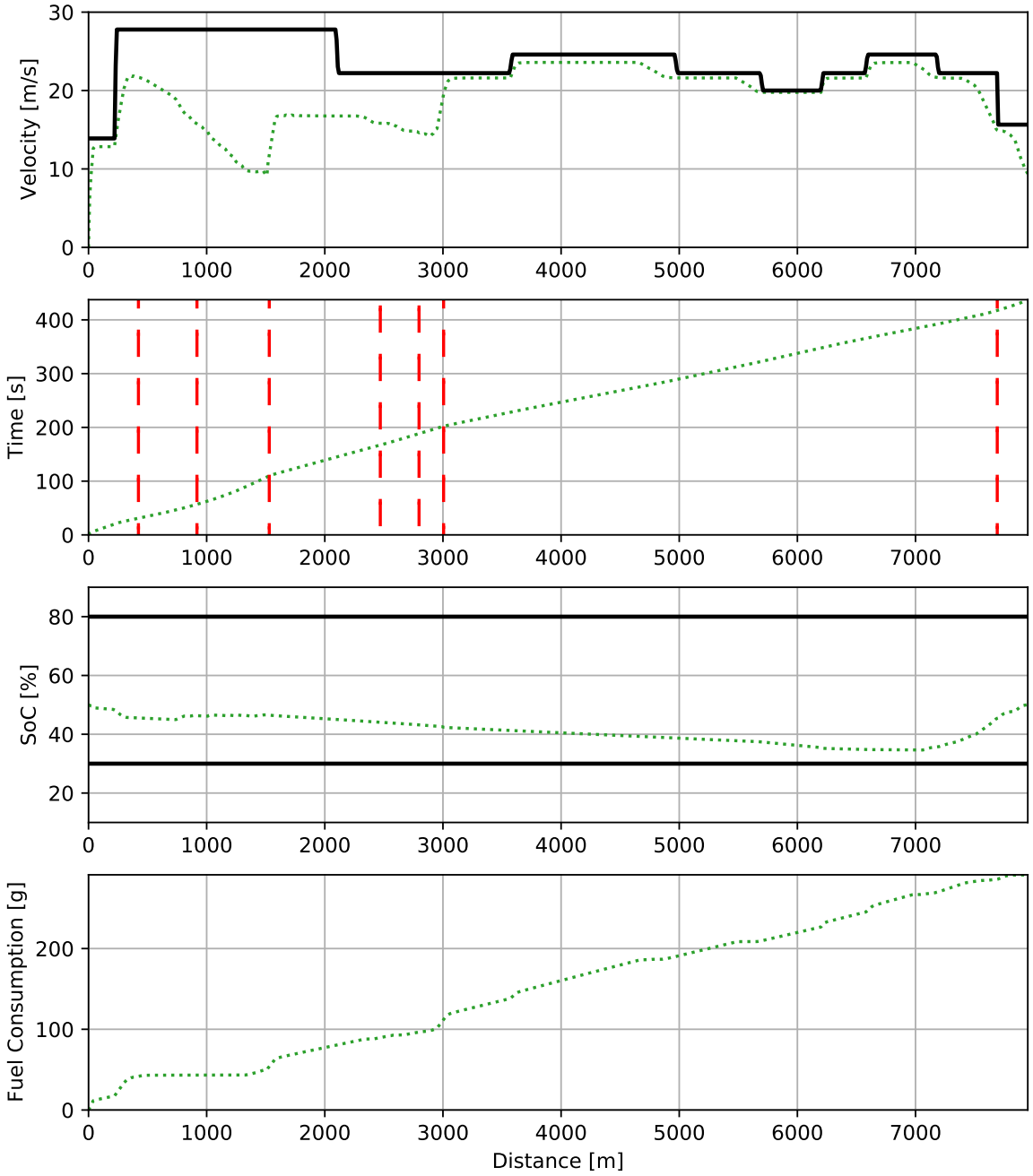


Figure 4.7: Deterministic DP Solution: Mixed Driving Condition

solver time by more than 300 times. The computational advantage makes the WS

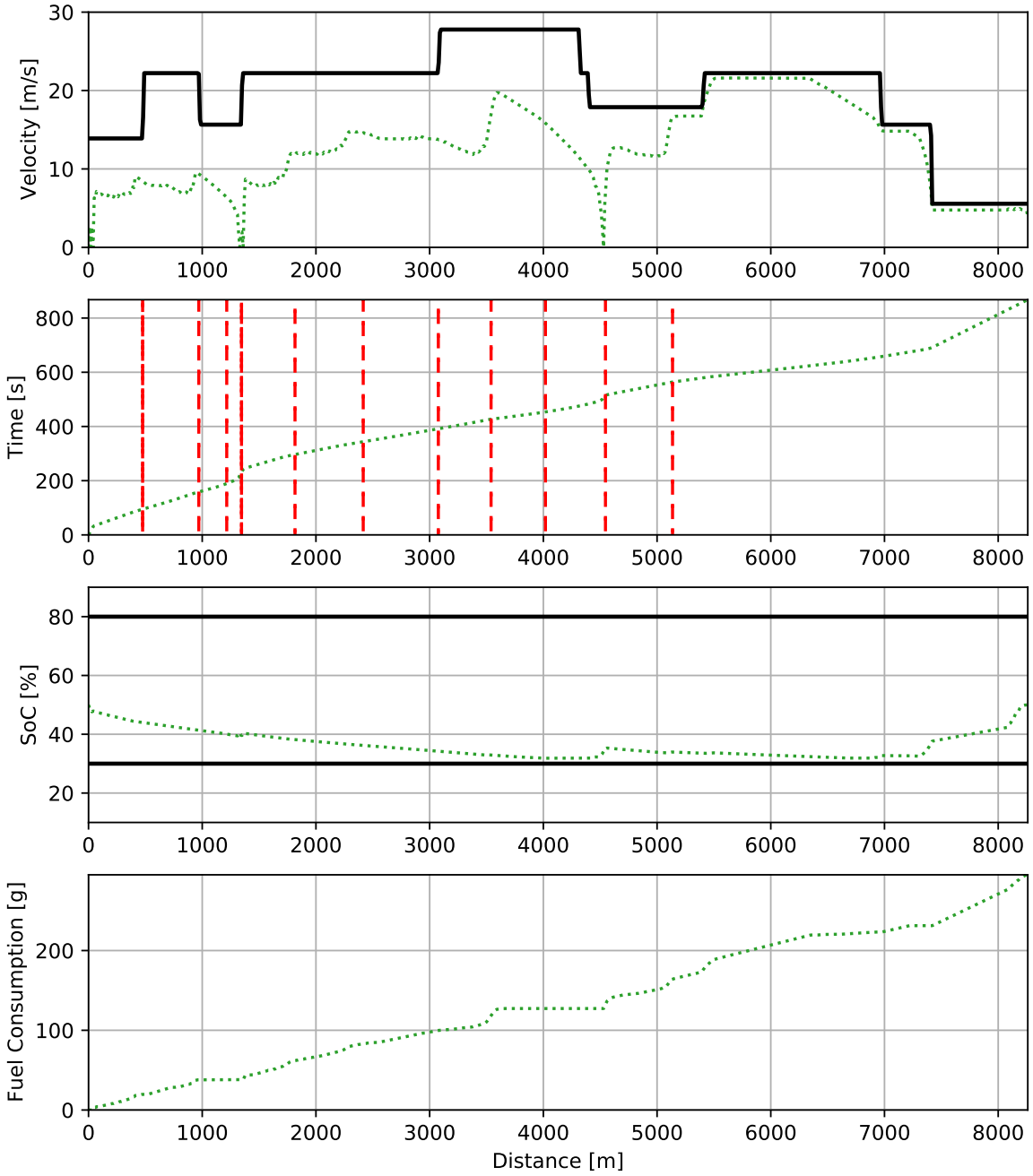


Figure 4.8: Deterministic DP Solution: Urban Driving Condition

solution readily available, and in Chapter 6, it will be demonstrated that the solver can be used for the real-time purpose given an NVIDIA GPU is equipped onboard.

4.7 Summary

In this chapter, the eco-driving problem was formulated in both the time and spatial domains. The time domain formulation will be used for the RL methods introduced in the later chapters, and the problem in spatial domain was solved in this chapter under the deterministic assumption. Finally, a fast DP solver implemented in CUDA C++ to take advantage of the parallel capability from GPU was presented and demonstrated a 300 times computational time reduction given the specific hardware.

Chapter 5: Model-Free Actor-Critic Method for Eco-Driving

5.1 Introduction

In Chapter 4, a deterministic dynamic programming algorithm and a practical fast solver were presented. To reduce the onboard computation and to alleviate the performance deterioration from the deterministic setup, a causal controller can be obtained by solving the trajectory optimization problem with a rollout length that is much smaller than the full route length, as in Model Predictive Control (MPC). However, the optimality and feasibility of the MPC policy are affected by the design of the terminal cost function and terminal set. Generally speaking, a longer rollout trajectory, which means a more computationally demanding onboard task, can be used to reduce the impact of the design of terminal cost function and terminal set. A good practical design should consider the trade-off between the performance and the available onboard computation resources.

Specific to the eco-driving problem studied in this work, a poor design of the terminal cost function and terminal set raises two concerns. First, as the objective only reflects the weighted sum of the fuel and travel time, an MPC controller without a proper terminal cost function tends to drain the battery SoC at the beginning of the trip, and this short-sighted strategy is typically not the most fuel-efficient

one. Intuitively, a controller that considers long-term optimality should strategically save the battery electric energy for the maneuvers that require running the engine in its low efficiency region, e.g., launching events throughout the trip, and use the braking events to recuperate energy into the battery pack [121]. Second, when the rollout length is not sufficiently long, the vehicle might not have enough time/distance to safely brake in front of a signalized intersection, potentially leading to infeasible solutions.

In this and the following chapters, Deep Reinforcement Learning (DRL), where the control agent can iteratively collect data, explore different strategies and progressively improve its policy, will be used to address the aforementioned issues. DRL provides a train-offline, execute-online methodology with which the policy is learned from the historical data or the interaction with simulated environments. With function approximators, the learning agent can then "memorize" either a policy that explicitly maps the state to action under complex and stochastic settings or a value function that represents the quality of each state in the state space. In the latter case, the value function can subsequently help to reduce the receding horizon while still accounting for the long term optimality.

In this chapter, model free DRL (MFDRL) will be considered, and model based DRL (MBDRL) will be studied in the following chapter. Via the interaction with the environment developed in Chapter 3, the MFDRL agent incrementally improves its policy by maximizing the sum of the rewards collected throughout each trip. Each time it violates constraints listed in Eqn. 4.2, a large negative reward (penalty) will be assigned, such that the agent progressively learns to avoid penalties from constraint violation.

In literature, several attempts have been made to synthesize RL for the eco-driving problem. Shi et al. [91] modeled the conventional vehicles with ICE as a simplified model and implemented Q-learning to minimize the CO_2 emission at signalized intersections. Li et al. [59] apply an actor-critic algorithm on the ecological ACC problem in car-following mode. Pozzi et al. [82] designed a velocity planner considering the signalized intersection and hybrid powertrain configuration with Deep Deterministic Policy Gradient (DDPG).

Compared to the existing work, the contributions of this chapter are as follows. First, the eco-driving problem is formulated as a Partially Observable Markov Decision Process (POMDP), which is then solved with a state-of-art MFDRL Actor Critic algorithm, Proximal Policy Optimization (PPO) with Long Short-Term Memory (LSTM) as function approximator. Second, a baseline controller representing human drivers and a deterministic trajectory optimization algorithm, along with the wait-and-see (WS) deterministic optimal solution presented in Chapter 4, are used to benchmark the performance of the MFDRL agent. Finally, simulations are conducted to illustrate that, with a minimal onboard computational requirement and a comparable travel time, the MFDRL agent reduces the fuel consumption by more than 17% compared against the baseline controller by modulating the vehicle velocity over the route and performing an energy-efficient approach and departure at signalized intersections, over-performing the more computationally demanding trajectory optimization method via deterministic MPC.

To stay consistent with most of the MFDRL literature, the common MDP notations will be used throughout this chapter, meaning the state, the action and the reward are represented by s, a and r, respectively.

5.2 Background

5.2.1 Proximal Policy Optimization

Actor Critic is one of the earliest concepts in the field of RL [9, 100]. The actor is typically a control policy, which can be either stochastic or deterministic, and the critic is a value function assisting the actor to improve the policy. For DRL, the actor and critic are both typically in the form of deep neural networks.

In this study, the policy gradient method is used to iteratively improve the policy. According to Policy Gradient Theorem in [67, 117], the gradient of the policy, parameterized by θ, can be determined by the following equation:

$$\nabla_\theta \eta(\pi_{\theta_k}) \propto \sum_s \rho(s) \sum_a Q^{\pi_{\theta_k}}(s,a) \nabla_\theta \pi_{\theta_k}(a|s), \tag{5.1}$$

where $\rho(s)$ is the discounted on-policy state distribution defined as follows:

$$\rho(s) = \sum_{t=0}^{\infty} \gamma^t P(s_t = s). \tag{5.2}$$

As in [101], the gradient can be estimated as follows:

$$\nabla_\theta \eta(\pi_{\theta_k}) = \mathbb{E}_\pi \left[A^{\pi_{\theta_k}}(s_t, a_t) \frac{\nabla_\theta \pi_{\theta_k}(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)} \right]. \tag{5.3}$$

Accordingly, to incrementally increase $\eta(\pi_\theta)$, the gradient ascent rule follows

$$\theta_{k+1} = \theta_k + \alpha_k \nabla_\theta \eta(\pi_{\theta_k}), \tag{5.4}$$

where α_k is the learning rate.

As Eqn. (5.3) is a local estimation of the gradient at the neighborhood of the current policy, updating the policy in such direction with a large step size could potentially lead to a large performance drop. Schulman et al. [86] proposed to constrain the difference between the probability distributions of the old and the new policy

with the trust region method. However, the algorithm requires the analytical Hessian matrix, resulting in a high computational load and a nontrivial implementation. In this work, a first-order method proposed by Schulman et al. [88] is used. Instead of Eqn. (5.3), a clipped surrogate objective function is defined as follows:

$$\begin{gathered} L_t(\theta) = \mathbb{E}_\pi\left[\min\left(r_t(\theta), \operatorname{clip}\left(r_t(\theta), 1-\epsilon, 1+\epsilon\right)\right) A^{\pi_{\theta_k}}(s_t, a_t)\right], \\ \text{where } r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)}, \\ \operatorname{clip}(x, a_{\min}, a_{\max}) = \min\left(\max\left(x, a_{\min}\right), a_{\max}\right). \end{gathered} \tag{5.5}$$

Here, the hyperparameter ϵ is the clipping ratio. Note the first-order derivative of the loss function around θ_k, $\nabla_\theta L_t(\theta)|_{\theta_k}$, is equal to $\nabla_\theta \eta(\pi_\theta)|_{\theta_k}$ which is consistent with Policy Gradient Theorem. Fig 5.1. shows the value of $L_t(\theta)$ as a function of $r_t(\theta)$. When $A^{\pi_{\theta_k}}(s_t, a_t)$ is positive, an increase in $\pi_\theta(a_t|s_t)$ results in an increase in $L_t(\theta)$. In the meantime, due to the clipping effect, any increase in $r_t(\theta)$ beyond $(1+\epsilon)$ no longer improves the surrogate objective. Similarly, when $A^{\pi_{\theta_k}}(s_t, a_t)$ is negative, any decrease beyond $(1-\epsilon)$ would not increase the surrogate objective. The minimization operation guarantees $L_t(\theta) \leq r_t(\theta) A^{\pi_{\theta_k}}(s_t, a_t)\ \forall r_t(\theta)$, which is required by the method of majorization-minimization (MM) [45].

5.2.2 Partially Observable Markov Decision Process

In many practical applications, states are not fully observable. The partial observability can arise from sources such as the need to remember the states in history, the sensor availability or noise and unobserved variations of the plant under control [40]. Such problem can be modeled as a POMDP where observations $o_t \in \mathcal{O}$, instead of states, are available to the agent. The observation at certain time follows the conditional distribution given the current state $o_t \sim P(\cdot|s_t)$.

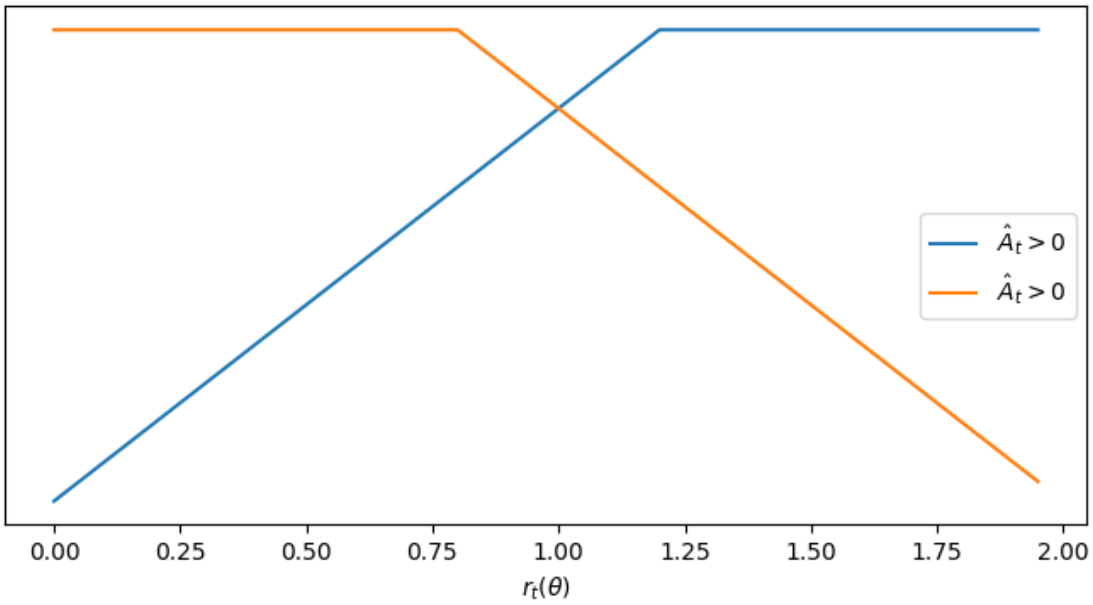


Figure 5.1: Surrogate Objective Function

For POMDP, the optimal policy, in general, depends on the entire history $h_t = (o_1, a_1, o_2, a_2, ...a_{t-1}, o_t)$, i.e., $a_t \sim \pi(\cdot|h_t)$. The optimal policy can be obtained by giving policies the internal memory to access the history h_t. In [116], it is shown that the policy gradient theorem can be used to solve the POMDP problem with Recurrent Neural Network (RNN) as the function approximator, i.e., Recurrent Policy Gradients. Compared to other function approximators such as Multilayer Perceptron (MLP) or Convolutional Neural Network (CNN), RNN exploits the sequential property of inputs and uses internal states for memory. Specifically, Long Short-Term Memory (LSTM) [43], as a special architecture of RNN, is typically used in DRL to avoid gradient explosion and gradient vanishing, which are well-known issues to

RNN [10]. In LSTM, three types of gates are used to keep the memory cells activated for arbitrarily long. The combination of Policy Gradient and LSTM has shown excellent results in many modern DRL applications [11, 109].

In the eco-driving problem, there are trajectory constraints while approaching traffic lights and stop signs. Thus, in these situations, the ego vehicle needs to remember the states visited in the recent past. LSTM based function approximators are therefore chosen to approximate the value function and the advantage function so that the ego vehicle can use information about the past states visited to decide on its torques.

5.3 Deep Reinforcement Learning Controller

5.3.1 POMDP Adoption

In this study, the eco-driving problem described by Eqn. (4.1) is solved as a POMDP. The constraints on the action space, i.e., Eqn. (4.1d), (4.1e) and (4.1f), are handled implicitly by a saturation function in the environment model, whereas the constraints on the state space are handled by imposing numerical penalties during the offline training.

Tab. 5.1 lists the observation and action spaces used to approach the eco-driving stochastic OCP. Here, SoC and v_{veh} are the states measured by the onboard sensors, and v_{lim}, v'_{lim}, s_{tlc}, s'_{lim} and s_{rem} are assumed to be provided by the downloaded map and GPS. t_{s} and t_{e} are the SPaT signals provided by V2I communication. When the upcoming traffic light is in the green phase, t_{s} remains 0, and t_{e} is the remaining time of the current green phase; when the upcoming traffic light is in red phase, t_{s} is the

Table 5.1: Observation and Action Space of the MFDRL in the Eco-driving Problem

	Variable	Description
$\mathcal{O} \in \mathbb{R}^9$	SoC	Battery SoC
	v_{veh}	Vehicle velocity
	v_{lim}	Speed limit at the current road segment
	v'_{lim}	Upcoming speed limit
	t_{s}	Time to the start of the next green light
	t_{e}	Time to the end of the next green light
	s_{tfc}	Distance to the upcoming traffic light
	s'_{lim}	Distance to the road segment where speed limit changes
	s_{rem}	Remaining distance of the trip
$\mathcal{A} \in \mathbb{R}^3$	T_{eng}	Engine torque
	T_{bsg}	Motor torque
	T_{brk}	Equivalent brake torque

remaining time of the current red phase, and t_{e} is the sum of the remaining red phase and the duration of the upcoming green phase.

The reward function $r : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ consists of four terms:

$$r = \text{clip}\left[r_{\text{obj}} + r_{\text{vel}} + r_{\text{batt}} + r_{\text{tfc}}, -1, 1\right]. \tag{5.6}$$

Here, r_{obj} represents the rewards associated with the OCP objective; r_{vel} is the penalty (negative reward) associated with the violation of the speed limit constraint in Eqn. (4.1i); r_{batt} represents the penalties associated with the violation of the battery constraints imposed by Eqn. (4.1g), (4.1h); r_{tfc} is the penalties regarding the violation of the traffic light constraint imposed by Eqn. (4.1j). Specifically, the first three terms

are designed as follows:

$$r_{\text{obj},t} = c_{\text{obj}} \left(\dot{m}_{\text{fuel},t}[g/s] + c_{\text{time}} \right), \tag{5.7}$$

$$r_{\text{vel},t} = c_{\text{vel},1} \left[v_{\text{vel},t} - v_{\text{lim},t} \right]^+ + c_{\text{vel},2} \dot{a}^2_{\text{veh},t}, \tag{5.8}$$

$$r_{\text{batt},t} = \begin{cases} c_{\text{batt},1} \left(\left[SoC_t - SoC^{\max} \right]^+ \right. \\ \qquad \left. + \left[SoC^{\min} - SoC_t \right]^+ \right), & s_t < s_{\text{total}} \\ c_{\text{batt},2} \left[SoC_F - SoC_t \right]^+, & s_t \geq s_{\text{total}} \end{cases} \tag{5.9}$$

where $[\cdot]^+$ is the positive part of the variable defined as

$$[x]^+ = \max(0, x). \tag{5.10}$$

In Eqn. (5.8), a penalty to the longitudinal jerk is assigned to the agent to improve the drive quality and to avoid unnecessary speed fluctuations.

While the design of the first three rewards is straightforward, the reward associated with the traffic light constraints is more convoluted to define. First, a discrete state variable m_{tfc} is defined in Fig. 5.2. $m_{\text{tfc}} = 0$ whenever the distance to the upcoming traffic light is greater than the critical braking distance $s_{\text{critical},t}$, which is defined as follows:

$$s_{\text{critical},t} = \frac{v^2_{\text{veh},t}}{2b_{\max}}, \tag{5.11}$$

where $b_{\max}$ is the maximal deceleration of the vehicle. Intuitively, the agent does not need to make an immediate decision regarding whether to accelerate to pass or to decelerate to stop at the upcoming signalized intersection outside the critical braking distance range. Once the vehicle is within the critical braking distance range, m_{tfc} is determined by the current status of the upcoming traffic light, the distance between the vehicle and the intersection and the maximal distance that the vehicle could drive

given the remaining green phase $s_{\max,t}$ defined as follows:

$$s_{\max,t} = \sum_{i=0}^{t_e} \left[\min\left(v_{\lim,t}, v_{\text{veh},t} + i a_{\max}\right)\right], \tag{5.12}$$

where $a_{\max}$ is the maximal acceleration.

If the upcoming traffic light is in green, i.e., $t_s = 0$, m_{tfc} gets updated to 1 if the distance between the vehicle and the upcoming intersection is less than the $s_{\max,t}$ and 2 otherwise. Intuitively, $m_{\text{tfc}} = 1$ means that the vehicle has enough time to cross the upcoming intersection within the remaining green phase in the current traffic light programming cycle. In case the vehicle following the actor policy was not able to catch the green light, a penalty proportional to s_{miss}, the distance between the vehicle at the last second of the green phase and the intersection, is assigned. On the other hand, $m_{\text{tfc}} = 2$ means the vehicle would not reach the destination even with the highest acceleration in the current cycle. If the upcoming traffic light is in red phase, $m_{\text{tfc},t}$ gets updated to 3. When the vehicle was not able to come to stop and violate the constraints, a penalty proportional to the speed at which the vehicle passes in red is assigned. As a summary, the reward associated with the traffic light constraints is designed as follows:

$$r_{\text{tfc},t} = \begin{cases} c_{\text{tfc},1} + c_{\text{tfc},2} s_{\text{miss},t}, & m_{\text{tfc},t} = 1 \\ c_{\text{tfc},1} + c_{\text{tfc},3} v_{\text{veh},t}, & \text{otherwise} \end{cases} \tag{5.13}$$

In Appendix A, a thorough guideline to the design and the tuning of the reward mechanism is provided, and the numerical values of all the constants in the reward function are listed in Table 5.2. In general, the reward r_t at each step should be between $[-1, 1]$ for better training stability [107]. To ensure the agent prioritizes constraint satisfaction over performance, the scale of the penalty to the violation of traffic rules and battery constraints should be magnitudes higher than the control

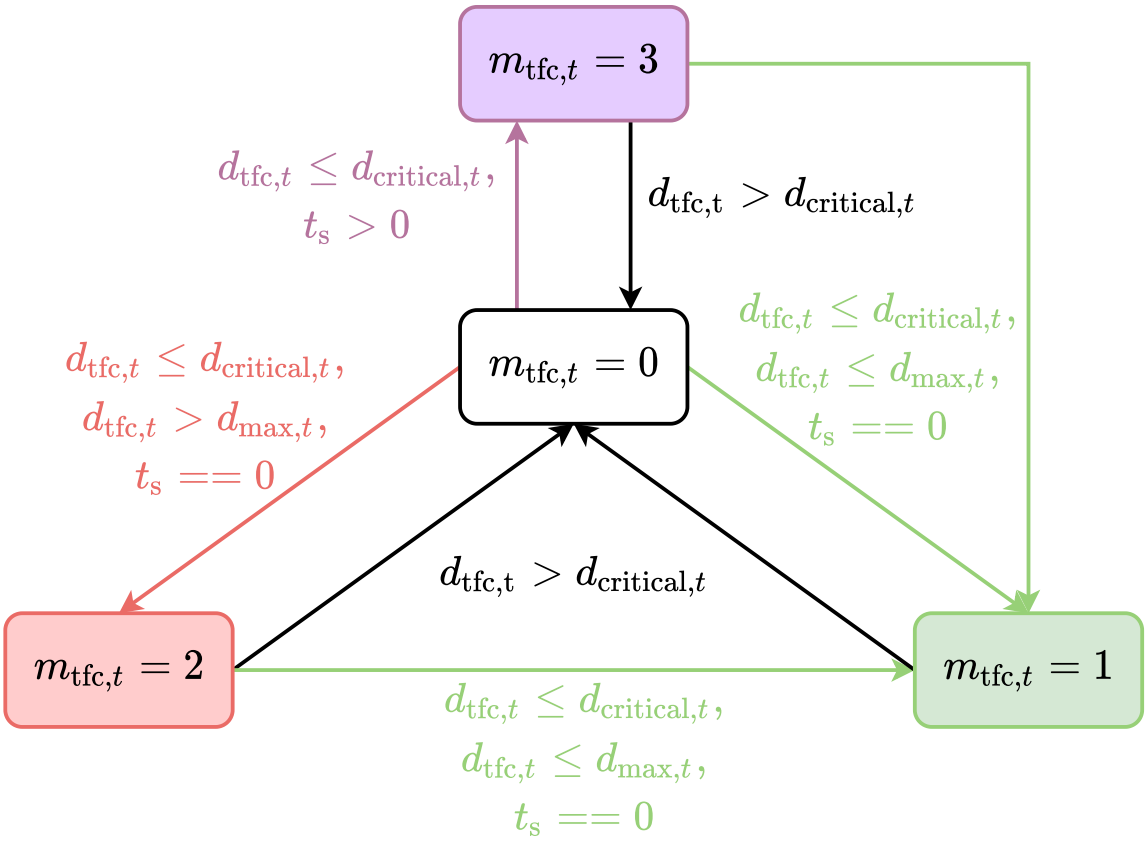


Figure 5.2: State Machine for the Indicator m_{tfc}

objective, i.e., the weighted sum of the fuel consumption and travel time. In the meantime, the performance of the controller deteriorates when the rewards associated with the objective become insignificant to penalties. In this dissertation, the rewards associated with the traffic rules and battery constraints are dense, meaning frequent rewards are received, and they provide feedback reflecting the quality of the strategy progressively. Sparse rewards, on the other hand, are only assigned at the end of the episodes. For example, it can be a binary signal of whether the policy was able to finish the trip without any violation. Despite the policies learned from sparse rewards

Table 5.2: Numerical Values of the Constants in Reward Function

Constants	c_{obj}	c_{time}	c_{vel}	c_{batt}	c_{tfc}
Values	-0.001	5	$\begin{bmatrix}-0.002\\-0.015\end{bmatrix}$	$\begin{bmatrix}-10\\-0.25\end{bmatrix}$	$\begin{bmatrix}-0.25\\-0.01\\-0.02\end{bmatrix}$

are more general, sparse rewards are significantly more demanding for the learning algorithm and often require a more sophisticated exploration scheme.

In order to determine the reward at any given time, the environment model requires states that are not directly available as observations such as $\dot{a}_t$, m_{tfc} and s_{miss}. Instead of making these states available to the control agent, the POMDP formulation is intentionally selected for two reasons. Firstly, m_{tfc} is heuristically determined and ignores the acceleration limit imposed by the powertrain components. Since it poses a significant impact on the reward at the intersection, revealing m_{tfc} to the controller results in a strong dependency of the strategy to it, and occasionally such dependency misleads the agent to violate the constraints. Secondly, s_{miss} is only relevant when the vehicle is catching a green light within the critical braking distance. Its numerical value in other situations could potentially mislead the policy that is in the form of a neural network.

Studies have suggested that clipping the rewards between $[-1, 1]$ results in a better overall training process [69, 107]. With the coefficients listed in Table 5.2, the negative reward saturates at -1 when $s_{\text{miss},t} > 75m$ and $m_{\text{tfc},t} = 1$ or $v_{\text{veh},t} > 7.5m/s$ and $m_{\text{tfc},1} = 2$, which means that the rewarding mechanism would no longer differentiate the quality of the state-action pairs beyond these thresholds at the signalized intersection. Such design, on one hand, reduces the strong impact of the heuristically designed

m_{tfc}. On the other hand, it also significantly slows down, or in some cases, prevents any learning as the rewards carry little directional information. Heess et al. [41] propose to use a more diversified and richer environment to overcome such issues. In this study, the diversity of the environment is ensured by the size of the SUMO map and the 10,000 randomly generated trips. In addition, the vehicle speed v_{veh} and battery SoC are randomly assigned following $\mathcal{U}(0, v_{\text{lim}})$ and $\mathcal{U}(SoC^{\min}, SoC^{\max})$, respectively, at every $T = 100$ time steps. This domain randomization mechanism, used in many other DRL applications [11, 121], forces the agent to explore the state space more efficiently and learn a more robust policy.

5.3.2 Algorithm Details

With the Monte Carlo method, the value function can be approximated as follows:

$$\hat{V}_{\xi}^{\pi_\theta}(s_t) \leftarrow \hat{\mathbb{E}}_{\pi_\theta}\left[\sum_{i=t}^{\infty} \gamma^{i-t} r\left(s_i, a_i\right) | s_t\right], \tag{5.14}$$

where the superscript π_θ indicates the value function is associated with the policy π parameterized by θ, and the subscript ξ indicates the value function itself is parameterized by ξ. Although being unbiased, the Monte Carlo estimator is of high variance and requires the entire trajectory to be simulated. On the other hand, TD(N) estimator is defined as follows:

$$\hat{V}_{\xi}^{\pi_\theta}(s_t) \leftarrow \hat{\mathbb{E}}_{\pi_\theta}\left[\sum_{i=t}^{t+N} \gamma^{i-t} r\left(s_i, a_i\right) + \hat{V}_{\xi_{\text{old}}}^{\pi_\theta}(s_{t+N}) | s_t\right]. \tag{5.15}$$

Compared to the Monte Carlo method, it reduces the required rollout length and the variance of the estimation by bootstrapping. However, TD(N) estimator is biased due to the approximation error in $V^{\pi}(s_{t+N})$. TD(λ) included in [100] takes the geometric sum of the terms from TD(N), leading to an adjustable balance between bias and

variance. In this study, since LSTM is used as the function approximator, the data tuple $(o_t, h_{o,t}, a_t, h_{a,t}, r_t, o_{t+1}, h_{o,t+1})$, instead of (s_t, a_t, r_t, s_{t+1}), are logged in simulation, where h_o and h_a are the hidden states of the policy and value function networks, respectively. Since the state space is randomized at every N steps, truncated TD(λ) is used for value function approximation. Specifically, after having collected the a sequence of tuples $(o_t, h_{o,t}, a_t, h_{a,t}, r_t, o_{t+1}, h_{o,t+1})_{t_0:t_0+N}$, the following equations are used for updating the value function $\forall t \in [t_0, t_0 + N - 1]$:

$$\hat{V}_\xi^{\pi_\theta}(o_t, h_{o,t}) \leftarrow V_\xi^{\pi_\theta}(o_t, h_{o,t}) + \sum_{i=t}^{t_0+N-1} (\gamma\lambda_V)^{i-t}\delta_i, \tag{5.16}$$

$$\text{where } \delta_i = r_i + \gamma V_\xi^{\pi_\theta}(o_{i+1}, h_{o,i+1}) - V_\xi^{\pi_\theta}(o_i, h_{o,i}). \tag{5.17}$$

Similarly, to balance the variance and the bias, the advantage function estimation is estimated with truncated GAE(λ) as proposed in [87]:

$$\hat{A}^{\pi_\theta}(o_t, h_{o,t}, a_t) = \sum_{i=t}^{t_0+N-1} (\gamma\lambda_A)^{i-t}\delta_i. \tag{5.18}$$

Fig. 5.3 shows the architectures of the neural network function approximator for the value function and policy. Here, multivariate Gaussian distribution with diagonal covariance matrix is used as the stochastic policy. With the estimated advantage function and the policy update rule in Eqn. (5.5), the policy tends to converge to a suboptimal deterministic policy prematurely since a sequence of actions is required in order to change the intersection-crossing behavior due to its hierarchical nature. Studies [68, 118] show adding the entropy of the stochastic policy to the surrogate objective function effectively prevents such premature convergence. As a result, the policy is updated by maximizing the following objective function:

$$L_{\text{mod},t}(\theta) = L_t(\theta) + \beta h(\mathcal{N}(\mu, \pm)). \tag{5.19}$$

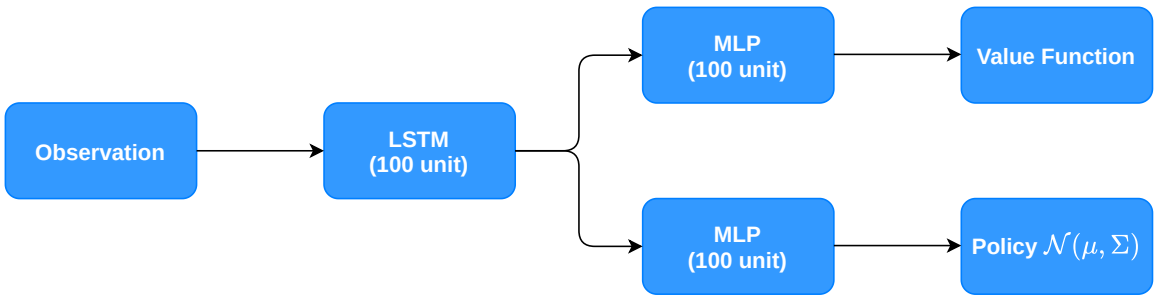


Figure 5.3: Network Architecture.

Here, $L_t(\theta)$ is the surrogate objective function defined in Eqn. (5.5). β and $h(\mathcal{N}(\mu, \pm))$ are the entropy coefficient and the entropy of the multivariate Gaussian policy.

Since PPO is on-policy, its sample efficiency is low. To accelerate the learning progress, multiple actors are distributed over different CPU processors for sample collection as shown in Fig. 3.6. Algorithm 1 lists the detailed steps of the algorithm, and Tab. 5.3 lists all the hyperparameters used for the training.

Algorithm 1: PPO for Eco-driving

Initialize EcoSim with the randomly generated routes;
Initialize the policy and value networks $\mu_\theta^{\pi_0}$, $V_\xi^{\pi_0}$;
while *NOT converged* **do**
  for *actor* $= 1, 2, ...N$ **do**
    Reset $v_{\text{vel},0} \sim \mathcal{U}(0, v_{\text{lim},0})$ and $SoC_0 \sim \mathcal{U}(0.3, 0.8)$;
    Execute policy $\pi_{\theta_{\text{old}}}$ in EcoSim for T time steps;
    Record $(o_t, h_{o,t}, a_t, h_{a,t}, r_t, o_{t+1}, h_{o,t+1})$ for each timestep;
    Compute advantage estimates $\hat{A}^{\pi_{\theta_{\text{old}}}}(o_t, a_t)$ following Eqn. (5.18);
  end
  Compute the gradient of the surrogate objective function $L_{\text{mod}}(\theta)$ in Eqn. (5.19) and apply gradient ascent w.r.t θ;
  Update the value function and its parameter set ξ following Eqn. (5.17);
end

Table 5.3: List of Hyperparameters in MFDRL

Parameter	Value
LSTM Layer Size	100
MLP Layer Size	100
Samples per Episode	50
Number of Episodes per Update	400
Number of CPUs	40
Number of GPUs	1
PPO Clipping	0.2
GAE(λ)	0.8
Truncated TD(λ)	0.4
Entropy Coefficient	0.01 (linear annealing)
Learning Rate α	1e-6
Discount Factor γ	0.999

5.4 Benchmarking Strategies

To benchmark the fuel economy benefits of CAV technologies, it is crucial to establish a baseline representative of real-world driving. In this work, the performance of the developed algorithm is benchmarked against another two causal strategies ready for real-time implementation, and the WS solution presented in the previous section. Compared to the WS solution, the causal strategies are clearly suboptimal. The suboptimality comes from the lack of full route information due to causality and the design of the controller itself. Nevertheless, these strategies serve as great benchmarks as the first strategy represents the combination of a heuristic energy management strategy commonly used in industry and the average human driving behavior, and the second one is a recently published optimal-control-based strategy.

5.4.1 Baseline

The baseline controller consists of the Enhanced Driver Model (EDM), a deterministic reference velocity predictor that utilizes route features to generate velocity profiles representing varying driving styles [35, 36], and a rule-based HEV energy management controller. The baseline strategy passes signalized intersections based on line-of-sight (LoS), a dynamic human-vision based distance parameter used to preview the upcoming route feature as devised by the Intersection Sight Distance (ISD) specified by the American Association of State Highway and Transportation Officials (AASHTO) and US DoT FHA [2].

5.4.2 Optimal Controller

In the previous work [24], a hierarchical MPC is formulated to co-optimize the velocity and powertrain controls with an aim to minimize energy in a 48V mild-HEV using Approximate Dynamic Programming (ADP). The controller solves a long-term optimization at the beginning of the trip that evaluates a base policy using limited full-route information such as speed limits, the position of route markers such as stop signs, traffic lights. To account for variability in route conditions and/or uncertainty in route information, a short-term optimization is solved periodically over a receding horizon using the approximated terminal cost from the long-term optimization. Time-varying SPaT information is accounted for by developing a heuristic controller that uses distance to the traffic light, current vehicle velocity to kinematically reshape the speed limit such that the vehicle can pass at a green light. The formulation and details of the algorithm are referred to [24]. A detailed analysis is done to demonstrate the real-time implementation of the developed controller in [25] at a test

track in Columbus, OH. For brevity, this optimal-control-based controller using ADP will be referred to as ADP controller in the remaining dissertation.

5.5 Results

The training takes place on a node in Ohio Supercomputer Center (OSC) [16] with 40 Dual Intel Xeon 6148s processors and an NVIDIA Volta V100 GPU. The results shown below requires running the training continuously for 24 hours.

As domain randomization is activated during training, the performance of the agent needs to be evaluated separately from the training to show the learning progress. Here, 40 trips in the training set with domain randomization deactivated are executed for every 10 policy updates, i.e., every 4,000 rollout episodes. In Fig. 5.4, the blue curves show the evolution of the mean policy entropy, the average cumulative rewards, the average fuel economy, the average speed and the complete ratio over the 40 randomly selected trips, and the red curves indicate the running average over 10 evaluation points. Here, a trip is considered successfully completed if the agent did not violate any constraint during the itinerary. At the beginning of the training, the policy entropy increases to encourage exploration, and as the training evolves, the policy entropy eventually decreases as the entropy coefficient β is linearly annealing. On average, the agent reaches a performance with an average fuel economy of 41.0 mpg (miles per gallon) and an average speed of ~ 12.7 m/s. In addition, the agent was able to learn to obey the traffic rules at signalized intersections thanks to the properly designed rewarding mechanism.

The performance of the MFDRL controller is then compared against the causal baseline and ADP controllers and the non-causal WS deterministic optimal solution

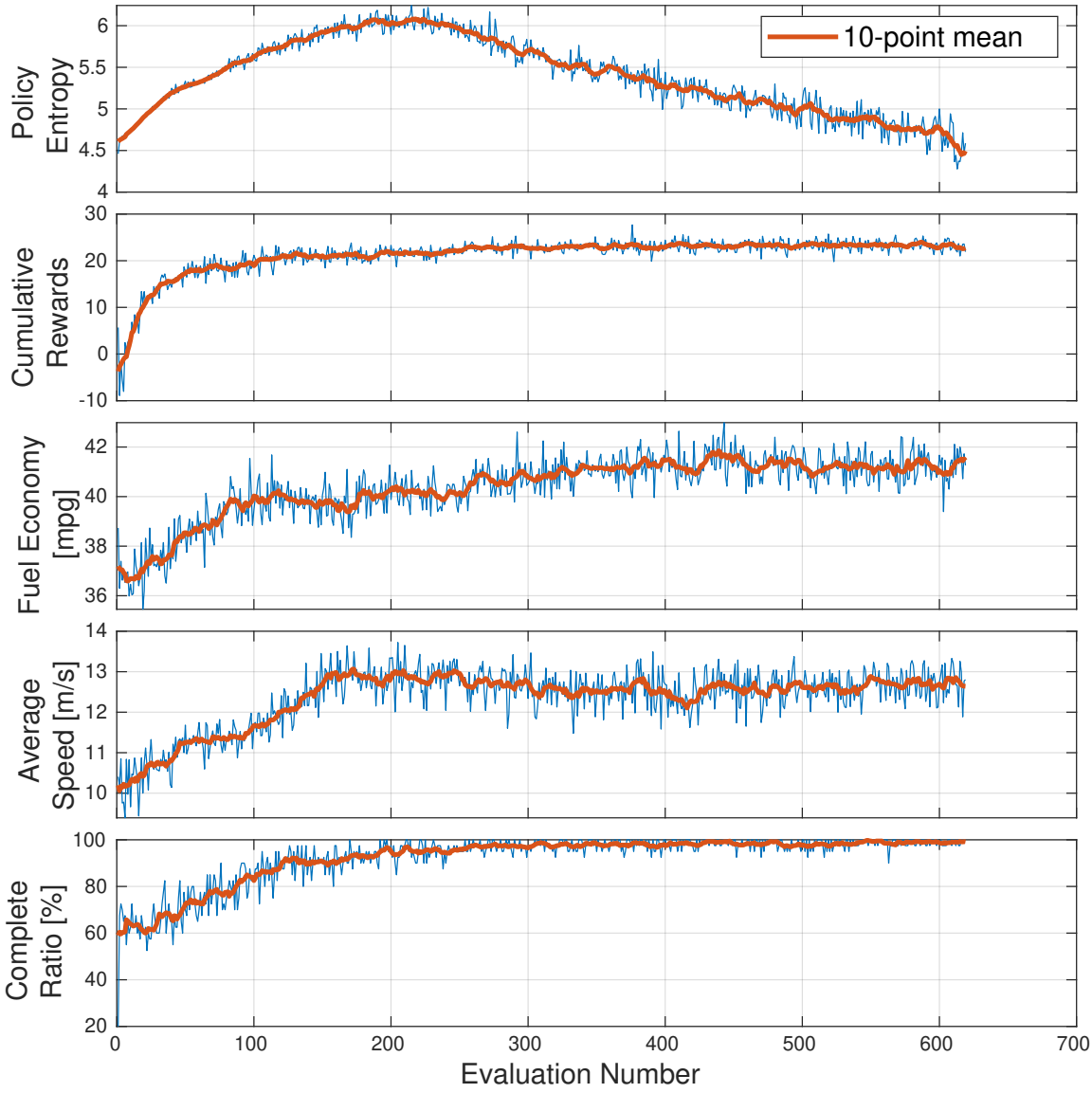


Figure 5.4: Evolution of Policy Entropy, Cumulative Rewards, Fuel Economy, Average Speed and Complete Ratio during Training

among the 100 testing trips shown in Fig. 3.5. Fig. 5.5 and Tab. 5.4 show the statistical comparison among the four strategies. Here, the black line in each box represents the median of the data, and the lower and upper boundaries represent the 25^{th} and 75^{th} percentiles, respectively. The extremes of the lines extending outside

the box represent the minimum and maximum limit of the data and the "+" symbol represents the outliers. With the comparable travel time, the MFDRL agent consumes 17.5% less fuel over all trips. Compared to the ADP controller, the MFDRL agent consumes 3.7% less fuel, while being $\sim 1m/s$ slower. Meanwhile, considering the ADP controller requires solving the full-route optimization via DDP before departure and a trajectory optimization at every timestep in real-time, the MFDRL strategy, which requires only the forward evaluation of the policy network, is more computationally tractable for online implementation.

The WS solution provides a dominant performance over the causal controllers. The additional benefits of the WS solution stem from the fact that the WS solution has the information of all the traffic lights over the entire trip, whereas the causal controllers only use the SPaT of the upcoming traffic light within 500m. Such advantage is also reflected in Fig. 5.6. Here, the average speed and the fuel consumption of each trip are plotted against the traffic light density, i.e., the number of traffic lights divided by the total distance in kilometers. Intuitively, the average speed of the three controllers decreases as the traffic light density increases. Compared to the causal controllers, the fuel economy of the WS solution is not affected by the increase in traffic light density. This is because the additional SPaT information from the subsequent traffic lights allows the WS solution to plan through the intersections more efficiently. In the meantime, as suggested by the regression curves, the MFDRL agent is less sensitive to the increase in traffic light density than the baseline and ADP controllers.

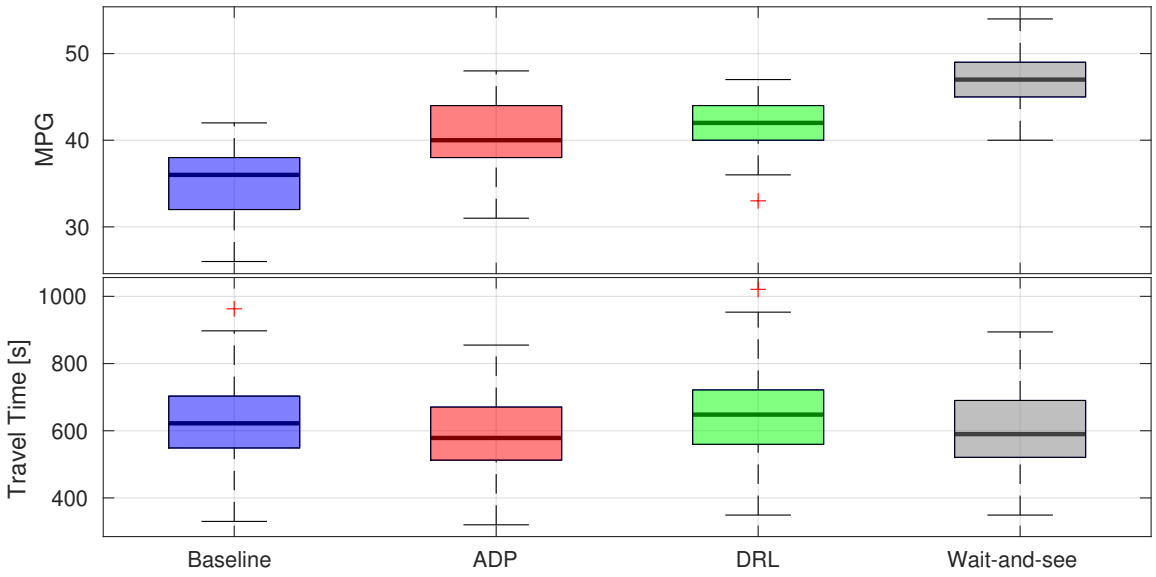


Figure 5.5: Fuel Economy, Travel Time Comparison and Charge Sustenance behavior for Baseline, MFDRL and WS Solutions

Table 5.4: Fuel Economy, Average Speed and SoC Variance for Baseline, ADP, MFDRL and WS Solutions

	Baseline	ADP	MFDRL	WS
Fuel Economy mpg	33.8	39.5	41.0	47.5
Speed Mean m/s	13.0	13.9	12.7	14.5
SoC Variance $\%^2$	13.2	21.6	18.2	22.6

Fig. 5.7 and 5.8 show the trajectories of the three controllers in urban and mixed (urban and highway) driving conditions, respectively. Driving under the same condition, the MFDRL agent was able to come to a full stop at signalized intersections less frequently compared to the baseline. In addition, the MFDRL agent utilizes more

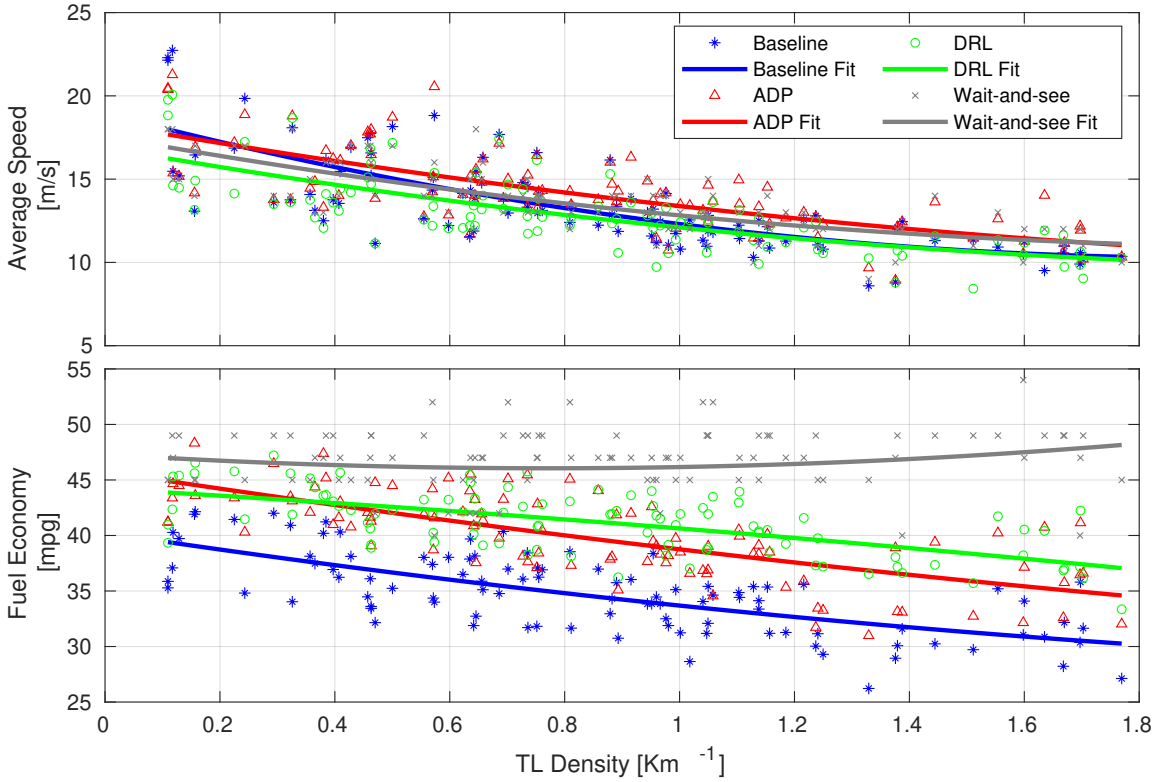


Figure 5.6: Variation of Average Speed and Fuel Economy against Traffic Light (TL) Density for Baseline, MFDRL and WS Solutions

of the battery capacity, i.e., a SoC profile with higher variation, compared to the baseline. MFDRL's efficient maneuvers while approaching the intersection coupled with better utilization of SoC results in up to 27% reduction in fuel consumption for both urban and mixed driving when compared against the baseline.

5.6 Summary

In this chapter, the eco-driving problem for HEVs with the capability of autonomously passing signalized intersections is formulated as a POMDP. To accommodate the complexity and high computational requirement of solving this problem,

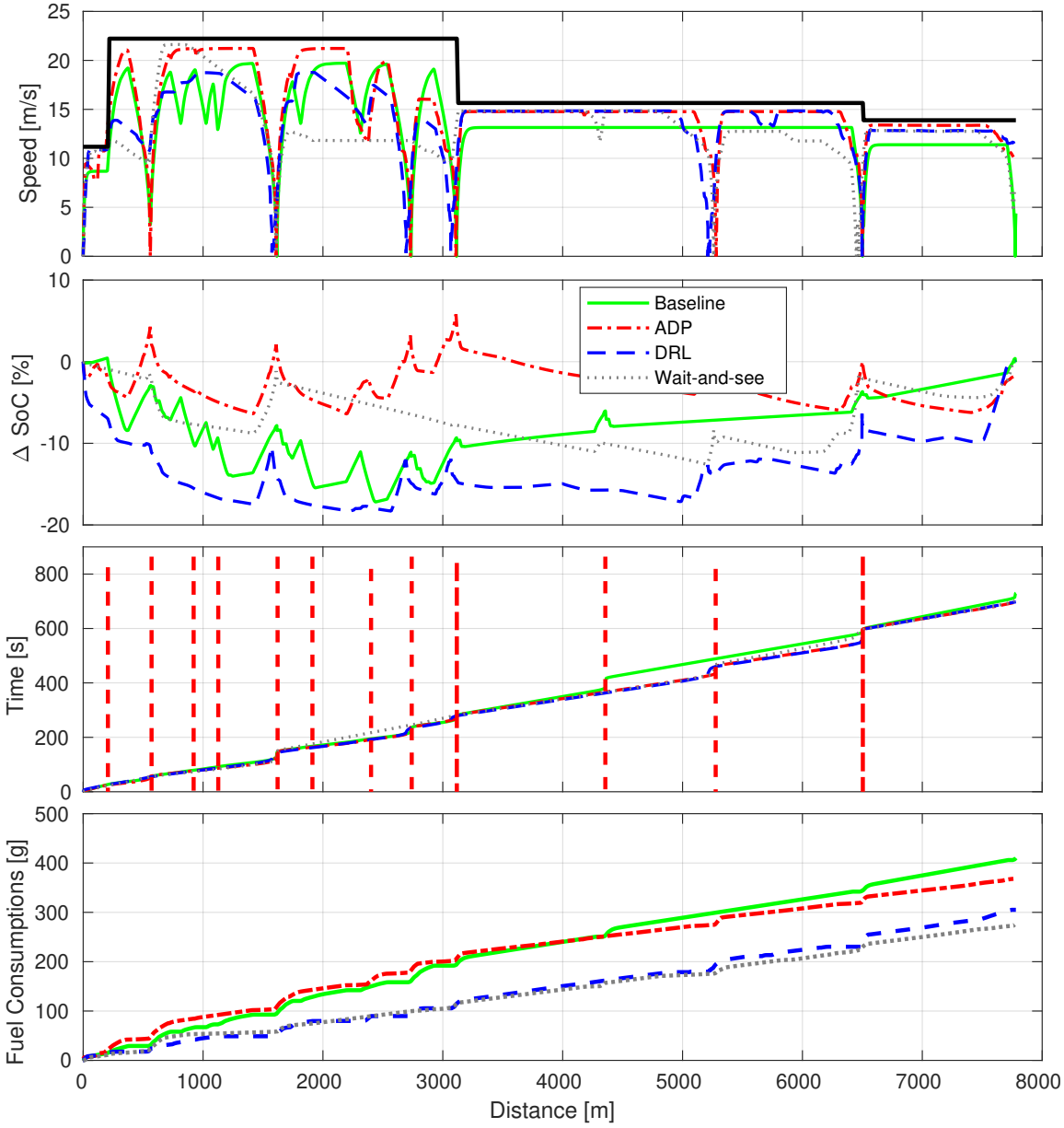


Figure 5.7: Comparison of Velocity, ΔSoC, Time-Space and Fuel Consumption for Baseline, MFDRL and WS Solutions [Urban Route]

a learn-offline, execute-online strategy is proposed. The MFDRL agent is trained via PPO with LSTM as the function approximators. The training and evaluation take place on the previously developed simulation environment EcoSim. To benchmark the

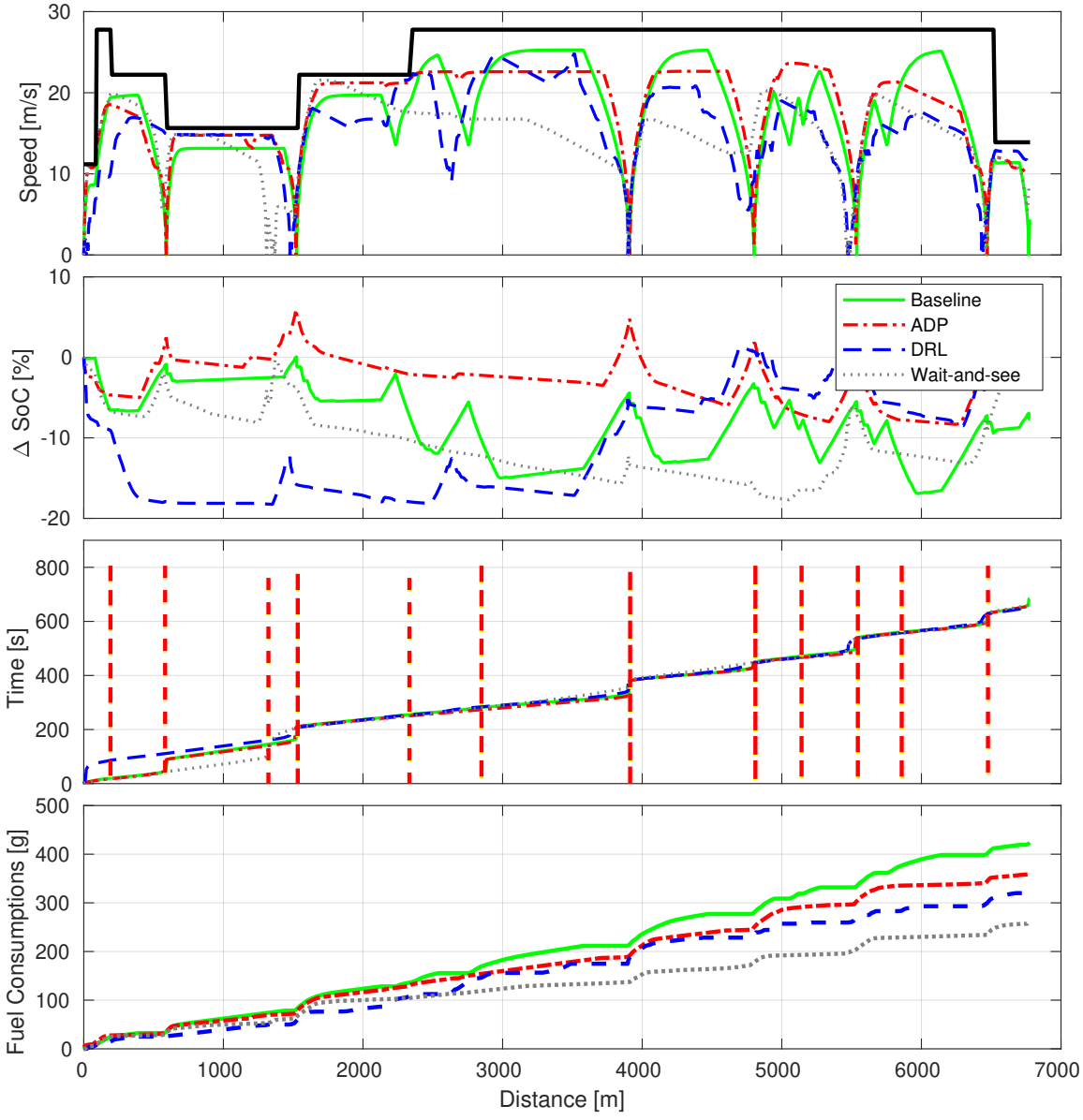


Figure 5.8: Comparison of Velocity, ΔSoC, Time-Space and Fuel Consumption for Baseline, MFDRL and WS Solutions [Mixed Route]

performance of the MFDRL agent, a baseline strategy, an MPC strategy and the WS optimal solution are presented. With the properly designed rewarding mechanism,

the agent learned to obey the constraints in the optimal control problem formulation. Furthermore, the learned explicit policy reduces the average fuel consumption by 17.5% over the 100 randomly generated trips in urban, mixed-urban and highway conditions compared to the baseline while keeping the travel time comparable.

Chapter 6: Safe Model-based Off-policy Reinforcement Learning for Eco-driving

6.1 Introduction

Chapter 5 formulates the eco-driving problem as a POMDP, synthesizes the previously developed virtual simulation environment and trains an agent using the state-of-the-art MFDRL algorithm PPO. While the MFDRL strategy shows improvements in average fuel economy and reductions in onboard computation, the methodology has a fundamental drawback. To teach the agent to drive under complex driving scenarios while satisfying all the constraints from powertrain and traffic rules, a complex and often cumbersome rewarding/penalizing mechanism needs to be designed. Furthermore, under such setup, the agent learns to satisfy constraints by minimizing the expected cost. For scenarios that are rare yet catastrophic, the scale of the cost penalizing constraint violation needs to be significantly larger than the learning objective itself [90]. As a result, such extrinsic rewarding mechanism increases the design period and deteriorates the final performance.

This chapter proposes a Safe Off-policy Model-Based Reinforcement Learning (SMORL) algorithm for the eco-driving problem. The advantages over the previously developed MFDRL approach and other existing works related to the eco-driving

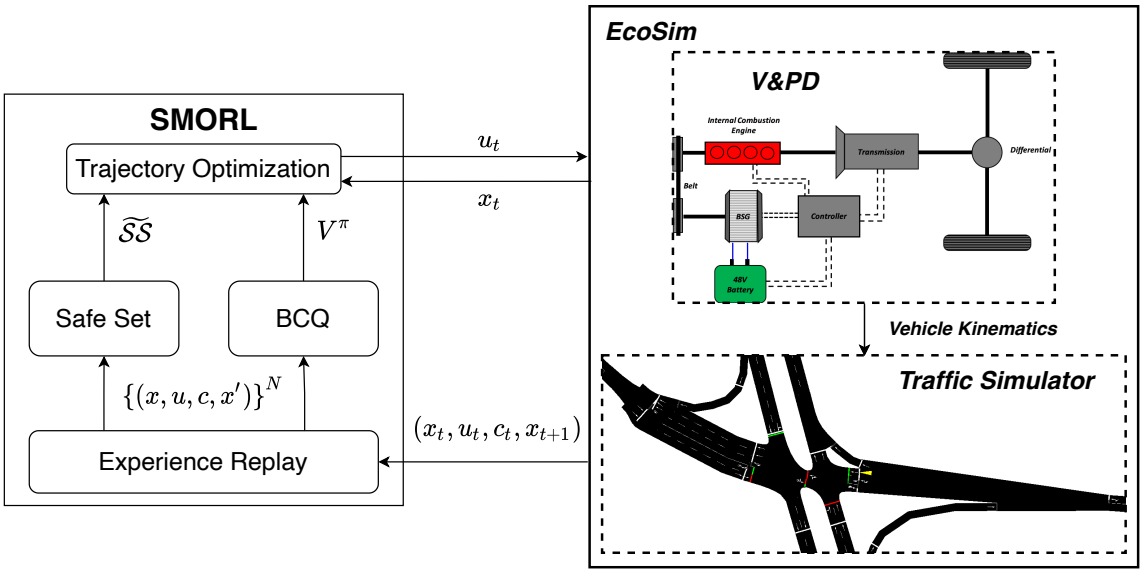


Figure 6.1: The Model-based Reinforcement Learning Framework.

problem are three-fold. First, the combination of off-policy learning and the use of a physics-based model improves the sample efficiency [112]. Second, the training does not require any extrinsic rewarding mechanism for constraint satisfaction. Third, the feasibility of trajectory is guaranteed by a safe set approximated by deep generative models.

In simulation, the proposed algorithm leads to a policy with a higher average speed and a better fuel economy compared to the MFDRL agent. Compared to the baseline controller, the learned strategy reduces the fuel consumption by more than 21% while keeping the average speed comparable.

Specifically, the proposed algorithm, of which the framework and components are shown in Fig. 6.1, synthesizes the structure of MPC to provide a mechanism to impose constraints and long term feasibility. The performance of the agent is improved iteratively with a learned terminal cost function (also known as Approximate Dynamic

Programming [12]). To be consistent with other works in literature, the technique is referred to as Model-based Reinforcement Learning (MBRL) in this chapter.

In [51, 65], on-policy MBRL algorithms with known dynamics are proposed for robotic applications. In [19], the system dynamics is learned as ensembles of neural network models, and the online optimization is solved via Cross Entropy Method (CEM). Thananjeyan et.al extended the MBRL algorithm from [19] and the concept of the safe set from [84] and proposed the safe on-policy MBRL algorithm for iterative tasks in [104] and multi-start, multi-goal tasks in [103]. In [103, 104], safe sets were approximated by kernel density function, and this dissertation extends the idea to a deep generative model for the high-dimensional safe set approximation.

While the use of the model increases the sample efficiency in the aforementioned studies, each transition collection becomes more computationally expensive due to the online trajectory optimization. To synthesize the historical data and reduce the overall training wall-time, Safe Model-based Off-policy Reinforcement Learning (SMORL), a safety-critical model-based off-policy Q-learning algorithm for systems with known dynamics, is proposed. To obtain the value function from Q function, an actor is explicitly trained as in DDPG [60] and Twin Delayed Deep Deterministic policy gradient algorithm (TD3) [28]. As the behavior policy is directly obtained from trajectory optimization and impacted by the learned Q function implicitly, there is a mismatch between the state-action visitation of the target policy and the state-action distribution in the experience buffer collected by the behavior policy. The mismatch, known in offline reinforcement learning [58], causes accumulating bootstrapping error and deteriorates the performance. In this section, Batch Constrained Q-learning (BCQ) [29] is used to address this distributional mismatch.

As SMORL is a model-based off-policy algorithm, it is able to take advantage of the experience replay [62] and model-based planning at the same time to reduce sample complexity. Subsequently, the proposed algorithm is successfully applied to the eco-driving problem. Without any extrinsic reward design, the agent trained with SMORL shows dominating performance compared to the one previously trained with MFRL algorithm in Chapter 5.

6.2 Problem Formulation and Preliminaries

The nonlinear, stochastic, time-invariant system is considered in this work:

$$\begin{aligned} x_{t+1} &= f(x_t, u_t, w_t) \\ x_t &\in \mathcal{X} \subseteq \mathbb{R}^n,\ t \in \mathbb{N}_+ \\ u_t &\in \mathcal{U}(x_t) \subseteq \mathbb{R}^m,\ t \in \mathbb{N}_+ \\ w_t &\in \mathcal{W} \subseteq \mathbb{R}^p,\ t \in \mathbb{N}_+. \end{aligned} \tag{6.1}$$

Here, x_t, u_t and w_t are the state, control, and uncertainty at time t. $\mathcal{X}$ and $\mathcal{U}$ are the feasible sets for states and inputs, respectively. The uncertainties are assumed to be independent and identically distributed (i.i.d.).

Let $\pi : \mathcal{X} \to \mathcal{U}$ be a feasible deterministic policy and Π be the set of all feasible deterministic policies. The objective of the OCP is to reach the goal set $\mathcal{G} \subseteq \mathbb{R}^n$ while finding the optimal policy π^* that minimizes the expectation of the discounted sum of the costs defined as follows:

$$\begin{aligned} \pi^* &= \underset{\pi \in \Pi}{\operatorname{argmin}}\, \eta(\pi), \text{ where} \\ \eta(\pi) &= \mathbb{E}_{w_t}\left[\sum_{t=0}^{\infty} \gamma^t c\left(x_t, u_t\right)\right], \\ &\text{where } u_t = \pi(x_t). \end{aligned} \tag{6.2}$$

Here, γ is the discount factor that prioritizes the immediate rewards and ensures the sum over the infinite horizon remains finite.

As in [103], the following assumption is made regarding the cost function.

Assumption 6.2.1 (Costs) *The cost is zero for the states inside the goal set $\mathcal{G}$ and positive for the states outside, i.e., $\exists \epsilon > 0$ such that $c(x,u) > \epsilon \mathbb{I}_{\mathcal{G}^C(x)}$ where $\mathbb{I}$ is the indicator function and $\mathcal{G}^C$ is the complement of the goal set $\mathcal{G}$.*

As in [15, 84, 103], the following definitions are given.

Definition 6.2.1 (Robust Control Invariant Set) *A set $\mathcal{C} \subseteq \mathcal{X}$ is said to be a robust control invariant set for the system Eqn. (6.1) if for all $x(t) \in \mathcal{C}$, there exists a $u(t) \in \mathcal{U}$ such that $f(x(t), u(t), w(t)) \in \mathcal{C}$, for all $w(t) \in \mathcal{W}$ and $t \in \mathbb{N}_+$.*

Definition 6.2.2 (Robust Successor Set $\boldsymbol{Suc}(\mathcal{S})$) *For a given set $\mathcal{S}$, its robust successor set $Suc(\mathcal{S})$ is defined as*

$$\begin{aligned} Suc(\mathcal{S}) = \{x' \in \mathbb{R}^n : \exists x \in \mathcal{S}, \exists w \in \mathcal{W} \\ \text{such that } x' = f(x, \pi(x), w)\}. \end{aligned} \tag{6.3}$$

Definition 6.2.3 (Robust Reachable Set $\mathcal{R}_N(x_0^j)$) *For a given initial state x_0^j, the N-step robust reachable set $\mathcal{R}_N(x_0^j)$ of the system defined in Eqn. (6.1) in a closed loop policy π at iteration j is defined recursively as*

$$\begin{aligned} \mathcal{R}_{i+1}^{\pi}(x_0^j) &= Suc(\mathcal{R}_i^{\pi}(x_0^j)) \cap \mathcal{X}, \\ \mathcal{R}_0^{\pi}(x_0^j) &= x_0^j, \end{aligned} \tag{6.4}$$

where $i = 0, 1, \ldots, N-1$.

Definition 6.2.4 (Safe Set) *The safe set $\mathcal{SS}^j$ contains the full evolution of the system at iteration j,*

$$\mathcal{SS}^j = \left\{ \bigcup_{k=0}^{\infty} \mathcal{R}_k^{\pi}(x_0^j) \bigcup \mathcal{G} \right\}. \tag{6.5}$$

As shown in [84], the exact form of the safe set in 6.5 is a robust control invariant set. As calculating its exact form is intractable, especially for high dimensional nonlinear system, it is, in practice, approximated as

$$\widetilde{\mathcal{SS}}^j = \bigcup_{k \in \mathcal{M}^j} x^k, \tag{6.6}$$

where $x^k = \{x_t^k : t \in \mathbb{N}_+\}$ is the trajectory at iteration k, and $\mathcal{M}^j = \{k \in [0, j) : \lim_{t\to\infty} x_t^k \in \mathcal{G}\}$ is the set of indices of which the trajectories were successfully driven to the goal. As the safe set in this work is constantly evolving during training, the iteration index j will be neglected in the remaining work.

For any policy π, the value function $V^\pi : \mathcal{X} \to \mathbb{R}$, the Q function $Q^\pi : \mathcal{X} \times \mathcal{U} \to \mathbb{R}$ and the advantage function $A^\pi : \mathcal{X} \times \mathcal{U} \to \mathbb{R}$ are defined as follows:

$$V^\pi(x_t) = \begin{cases} \mathbb{E}_\pi \left[\sum_{i=t}^{\infty} \gamma^{i-t} c\left(x_i, u_i\right) | x_t \right], & x_t \in \mathcal{SS} \\ \infty, & \text{otherwise.} \end{cases} \tag{6.7}$$

$$Q^\pi(x_t, u_t) = \begin{cases} \mathbb{E}_\pi \left[\sum_{i=t}^{\infty} \gamma^{i-t} c\left(x_i, u_i\right) | x_t, u_t \right], & \begin{array}{l} x_t \in \mathcal{SS} \\ u_t \in \mathcal{U} \end{array} \\ \infty & \text{otherwise.} \end{cases} \tag{6.8}$$

$$A^\pi(x_t, u_t) = Q^\pi(x_t, u_t) - V^\pi(x_t). \tag{6.9}$$

6.3 Proposed Method

In this chapter, an off-policy model-based deep reinforcement learning algorithm with approximated safe set is proposed. At any given time t during policy execution, the following trajectory optimization problem with a receding horizon of H steps is

solved:

$$\begin{aligned}
\min_{\{\tilde{u}_k\}_{k=t}^{t+H-1}} \mathbb{E}\left[\sum_{k=t}^{t+H-1} \gamma^{k-t} c(\tilde{x}_k, \tilde{u}_k) + \gamma^H V^\pi(\tilde{x}_{k+H})\right] \\
\text{s.t. } \tilde{x}_{k+1} = f(\tilde{x}_k, \tilde{u}_k, w_k) \\
\tilde{x}_t = x_t \\
\tilde{x}_k \in \mathcal{X},\ k = t, \ldots, t+H-1 \\
\tilde{x}_{t+H} \in \widetilde{\mathcal{SS}} \\
\tilde{u}_k \in \mathcal{U},\ k = t, \ldots, t+H-1,
\end{aligned} \tag{6.10}$$

where $\tilde{x}$ and $\tilde{u}$ are the variables for states and control actions in the predicted trajectory. Compared to the formulation in [65], the state $\tilde{x}_k$ and action $\tilde{u}_k$ are explicitly constrained to be within the feasible region in the receding horizon and the terminal state $\tilde{x}_{t+H}$ to be within the safe set $\widetilde{\mathcal{SS}}$. With the presence of uncertainties in the dynamic system, solving the exact form of the above stochastic optimization problem can be challenging. In [19, 103, 104], Cross Entropy Method (CEM) is used to solve the problem with unknown dynamics as a chance constraint problem. In Section 4.2, techniques will be discussed to simplify and solve the optimization in the eco-driving problem.

As most of the model-based deep reinforcement learning methods with trajectory optimization in literature learn the value function as the terminal cost for the MPC [19,51,65,103], the learning algorithm becomes on-policy. While the trajectory optimization increases the sample efficiency and helps exploration [65], solving the trajectory optimization problem makes each data sample more computationally expensive. As a result, the training wall time is not necessarily reduced. In this work, the off-policy Q-learning [115] is instead proposed. To use the learned Q function in

trajectory optimization, the following equation needs to be solved:

$$\min_{\{\tilde{u}_k\}_{k=t}^{t+H}} \mathbb{E}\left[\sum_{k=t}^{t+H-1} \gamma^{k-t} c(\tilde{x}_k, \tilde{u}_k) + \gamma^H Q_\theta^\pi(\tilde{x}_{t+H}, \tilde{u}_{t+H})\right], \tag{6.11}$$

where Q_θ^π is the approximated Q function parameterized by θ. Compared to solving Eqn.(6.10), solving Eqn. (6.11) requires one extra computational step:

$$V^\pi(\tilde{x}_{t+H}) = \min_{\tilde{u}_{t+H}} Q_\theta^\pi(\tilde{x}_{t+H}, \tilde{u}_{t+H}). \tag{6.12}$$

Depending on the dimension of the problem, solving Eqn. (6.12) can be computationally intractable, especially for online control. Several algorithms, e.g., DDPG, TD3 and dueling network [113] are proposed to obtain the value function from the Q function. In this work, the off-policy actor-critic algorithm TD3 is used since it reduces the overestimation and is shown to be more stable than DDPG. Specifically, with the sample (x_j, u_j, c_j, x'_j) from the experience replay buffer $\mathcal{D}$ [62], the target for the Q function during training is constructed as follows,

$$y_j = c_j + \gamma \max_{i=1,2} Q_{\theta'_i}(x'_j, u'_j), \tag{6.13}$$

$$u'_j = \pi_{\phi'}(x'_j). \tag{6.14}$$

Here, Q_{θ_1} and Q_{θ_2} are two independently trained critic networks. $Q_{\theta'_1}$ and $Q_{\theta'_2}$ are the corresponding target networks. π_ϕ and $\pi_{\phi'}$ are the actor network and its target network, respectively. The critics are then updated following

$$\begin{gathered}\theta_i \leftarrow \theta_i - \alpha \nabla_{\theta_i}\left[\frac{1}{N}\sum_{j=1}^{N}\left(y_j - Q_{\theta_i}(x_j, u_j)\right)^2\right],\ i = 1, 2, \\ \left\{(x_j, u_j, c_j, x'_j) \sim \mathcal{D}\right\}_{j=1}^{N}.\end{gathered} \tag{6.15}$$

where α is the learning rate, and N is the batch size.

In the off-policy learning algorithm used here, the behavior policy is the trajectory optimization where state and action constraints within the receding horizon are satisfied thanks to the constrained optimization formulation. However, the trained actor π_ϕ makes decisions solely based on the Q function. The resulting mismatch between the distribution of state-action pairs induced by the actor π_ϕ and that collected by the behavior policy results in extrapolation error leading to unstable training [58]. As an example in the eco-driving problem, the trajectory optimization ensures the power is solely generated from ICE when the battery SoC is at the lower limit $SoC^{\min}$. Accordingly, no state-action pair resembling low SoC and high motor torque can be collected, which leads to extrapolation in the Q function near the region. The error can eventually cause unstable training or inferior performance.

To address the extrapolation error induced by the mismatch in distributions, Batch Constrained Q-learning (BCQ) [29] originally proposed for offline reinforcement learning is used. Here, a generative model, specifically a Variational Autoencoder (VAE) [53], $G_\omega(x)$ is trained to resemble the state-action distribution in the experience replay buffer.

While the background on VAE and the training objective are covered in Section 2.3.3, the training and sampling procedure specific to the problem formulation here are shown again. Here, the samples generated from VAE are the actions $a' \sim G_\omega(x')$, and they should ideally match the distribution collected by the behavior policy. As the distribution $G_\omega(x')$ is a function of x', the actions a' is sampled conditioning on the states of the system. Defining $A = \{a'_i\}_{i=1}^N$ and $X = \{x'_i\}_{i=1}^N$, the variational lower bound $L(\omega_1, \omega_2, A, X)$ becomes

$$\mathcal{L}(\omega_1, \omega_2, A, X) = -D_{\mathrm{KL}}\left(q_{\omega_1}(z|A, X)||p(z)\right) + \mathbb{E}_{q_{\omega_1}(z|A,X)}\left[\log p_{\omega_2}(A|z, X)\right] \quad (6.16)$$

When sampling the actions a' given x', samples from the multivariate normal distribution are first drawn as

$$z \sim \mathcal{N}(0, 1). \tag{6.17}$$

Subsequently, sampled z are mapped to the action space via the decoder $p_{\omega_2}(\cdot|x', z)$. More details about implementation are presented in Section 6.4.2.

With the VAE to resemble to expert behavior, instead of selecting action following Eqn. (6.14), the action is now selected as

$$\begin{aligned} u'_j = \underset{u_{j,k}+\xi_\phi(x_j,u_{j,k},\Phi)}{\operatorname{argmin}} \left[\max_{i=1,2} Q_{\theta'_i}(x'_j, u_{j,k} + \xi_\phi(x_j, u_{j,k}, \Phi)) \right], \\ \left\{ u_{j,k} \sim G_\omega(x'_j) \right\}_{k=1}^{n}. \end{aligned} \tag{6.18}$$

Here n is the hyperparameter that is the number of actions sampled from the generative model. The action u' used for the target value for the Q function is selected as the best among the n sampled ones. Note that there is no longer an actor network mapping from state to action. Instead, to ensure the agent can learn on top of the actions sampled from the generative model imitating the behavior policy from the experience buffer, a perturbation network ξ_ϕ whose output is clipped between $[-\Phi, \Phi]$ is trained. The perturbation network ξ_ϕ is updated by deterministic policy gradient theorem from [93] as

$$\phi \leftarrow \phi - \alpha \nabla_\phi \left[\frac{1}{N} \sum_{j=1}^{N} Q_{\theta_1}(x_j, u_j + \xi_\phi(x_j, u_j, \Phi)) \right]. \tag{6.19}$$

To reduce the accumulating error from bootstrapping, all the target networks are updated with slower rates as

$$\theta'_i \leftarrow \tau\theta_i + (1 - \tau)\theta'_i, \quad i = 1, 2, \tag{6.20}$$

$$\phi' \leftarrow \tau\phi + (1 - \tau)\phi', \tag{6.21}$$

where τ is a constant on the order of 10^{-3} to 10^{-1}.

In [103, 104], the safe set is approximated by kernel density estimation, which typically works well only for problems in low dimensions as explained in Section 2.3.1. Here, the approximation is extended to high-dimensional setting by using deep generative models. Following the notion in [104], the safe set is approximated as

$$\widetilde{\mathcal{SS}} = \{x : p_\psi(x) \geq \delta\}, \tag{6.22}$$

where $p_\psi : \mathcal{X} \to [0, 1]$ is the probability that a state is inside the safe set parameterized by ψ, and the constant δ regulates how exploratory the controller is. Note that the generative model used for safe set approximation needs to model the probability explicitly and can be slow in sampling, whereas the generative model resembling the distribution of state-action pairs in the experience replay needs to be fast in sampling while the explicit probability is not required.

Due to the aforementioned consideration, the autoregressive model with Long Short-term Memory (LSTM) [52] is used. The description of the model, as well as the training objective, is included in Section 2.3.2. In Sec.6.4, the use of the autoregressive model in the application of eco-driving is motivated as the dimension of the problem can get large once the future conditions are sampled discretely.

In summary, Safe Model-based Off-policy Reinforcement Learning (SMORL) is proposed. The algorithm builds on SAVED [104] and extends it to be an off-policy algorithm with the methods proposed in BCQ. The detailed step-by-step algorithm is included in Algorithm 2.

6.4 Implementation Details

Algorithm 2: Safe Model-based Off-policy Reinforcement Learning (SMORL)

Initialize Q-networks Q_{θ_1}, Q_{θ_2} independently, and duplicate target networks $Q_{\theta'_1}$, $Q_{\theta'_2}$.
Initialize the perturbation network ξ_ϕ, its target network ξ'_ϕ and VAE $G_\omega = \{E_{\omega_1}, D_{\omega_2}\}$.
Initialize the experience replay buffer $\mathcal{D}$.
Collect N_0 successfully executed trajectories with a baseline controller and initialize the safe set $\widetilde{\mathcal{SS}}$.
for $n_{iter} \in 1, \ldots, N_{iter}$ **do**
  while j^{th} *trajectory NOT finished* **do**
    Select control action u_t by solving trajectory optimization in Eqn. (6.10).
    Sample mini-batch of N transitions (x, u, c, x') from $\mathcal{D}$.
    For each transition, sample n actions u'_j from $G_\omega(x')$ and n perturbations from $\xi_\phi(x', u', \Phi)$.
    Update the critic networks Q_{θ_1}, the target networks Q_{θ_2} following Eqn. (6.15) and $Q_{\theta'_1}$, $Q_{\theta'_2}$ following Eqn. (6.20).
    Update perturbation network ξ_ϕ following Eqn. (6.19).
    Update VAE G_ω by maximizing Eqn. (2.42).
  end
  if $x_T \in \mathcal{G}$ **then**
    Push the trajectory $\{(x_t, u_t, c_t, x_{t+1})\}^T$ to $\mathcal{D}$.
    Update the safe set $\widetilde{\mathcal{SS}}$ with minibatchs sampled from $\mathcal{D}$ following Eqn. (2.40).
  end
end

6.4.1 Trajectory Optimization

Specific to the eco-driving problem, the state vector x_t is defined as a vector with 88 states. A description of the states are listed in Tab. 6.1. Here, the first seven elements of the state vector are the battery SoC, the vehicle speed v_{veh}, the current speed limit v_{lim}, the next speed limit v'_{lim}, the distance to the next speed limit s_{lim}, the distance to the upcoming traffic light s_{tls} and the total remaining distance s_{rem}. The remaining 81 elements are the sampled upcoming traffic light status in the next 80 seconds x_{tfc}. For example, if the upcoming traffic light has 20 seconds remaining for the current red phase and will remain in green for the rest of the 80 seconds, the first 21 elements of the sampled upcoming traffic light status are 0, and the rest are set to 1. Compared to the manually extracted feature representation in [120], the sampled representation reduces the discontinuity and results in a better performance.

As the vehicle considered in this study is assumed equipped with connected features, e.g., advanced mapping and V2I connectivity, and surrounding vehicles are not included in the study, it is assumed that the ego vehicle can deterministically predict the uncertainties from driving conditions within the receding horizon in this study. Sun et al. [96] suggests by formulating the problem as a chance constraint or a distributionally robust optimization problem, uncertainties in SPaT can be considered without additional computational load.

In Eqn. (4.1), the receding horizon H is in the time domain. While it is easier to incorporate the time-based information such as SPaT received from V2I communication in the time domain, an iterative dynamic look-ahead process is required to process any distance-based route feature, such as speed limits, grade, traffic light and

Table 6.1: State and Action Spaces of SMORL in the Eco-driving Problem

	Variable	Description
$\mathcal{X}$	$SoC \in \mathbb{R}$	Battery SoC
	$v_{\text{veh}} \in \mathbb{R}$	Vehicle velocity
	$v_{\text{lim}} \in \mathbb{R}$	Speed limit at the current road segment
	$v'_{\text{lim}} \in \mathbb{R}$	Upcoming speed limit
	$d_{\text{tfc}} \in \mathbb{R}$	Distance to the upcoming traffic light
	$d'_{\text{lim}} \in \mathbb{R}$	Distance to the road segment of which the speed limit changes
	$d_{\text{rem}} \in \mathbb{R}$	Remaining distance of the trip
	$x_{\text{tfc}} \in \{0,1\}^{81}$	Sampled status of the upcoming traffic light
$\mathcal{U}$	$T_{\text{eng}} \in \mathbb{R}$	Engine torque
	$T_{\text{bsg}} \in \mathbb{R}$	Motor torque
	$T_{\text{brk}} \in \mathbb{R}$	Equivalent brake torque

stop sign locations. For example, the controller requires the speed limits as the constraints to generate speed trajectory while the speed limits can change based on the distance traveled by the speed trajectory. In this study, the value and Q functions are learned in the time domain for the ease of integration with the time-based traffic simulator, while the trajectory optimization is conducted in the spatial domain.

As SPaTs and speed limits do not depend on the decision made by the ego vehicle in the spatial domain, they are incorporated into the optimization problem as constraints, and only the vehicle speed, battery SoC and the time at which the vehicle reaches the given distance are considered as the state in the trajectory optimization.

Define the optimization state $z \in \mathcal{Z} \subseteq \mathbb{R}^3$ as

$$z_s = \left[v_{\text{veh},s}, SoC_s, t_s\right]^T. \tag{6.23}$$

Here, s is the index in the discretized the spatial domain with $\Delta s = 10m$, and the dynamics of z in the time and spatial domains are converted following

$$\frac{\Delta z}{\Delta s} = \frac{\Delta z}{\Delta t}\frac{\Delta t}{\Delta s} = \frac{\Delta z}{\Delta t}\frac{1}{v_{\text{veh}}}. \tag{6.24}$$

As a result, the trajectory optimization is formulated as

$$\min_{\{\tilde{u}\}_{k=s_t}^{s_t+H_s-1}} \sum_{k=s_t}^{s_t+H_s-1} \gamma^{t_k} c(\tilde{z}_k, \tilde{u}_k) + \gamma^{t_{H_s}} V^{\pi}(\mathcal{G}(x_t, z_{H_s})) \tag{6.25a}$$

where:

$$c(\tilde{x}_k, \tilde{u}_k) = \left(\lambda \dot{m}_{\text{fuel},k} + (1-\lambda)\right) \frac{\Delta s}{v_{\text{veh},k}} \cdot \mathbb{I}\left[s_k < s_{\text{total}}\right] \tag{6.25b}$$

$$\text{s.t. } SoC_{k+1} = f_{\text{batt},s}\left(\tilde{z}_k, \tilde{u}_k\right) \tag{6.25c}$$

$$v_{\text{veh},k+1} = f_{\text{veh},s}\left(\tilde{z}_k, \tilde{u}_k\right) \tag{6.25d}$$

$$T_{\text{eng}}^{\min}(\omega_{\text{eng},k}) \leq T_{\text{eng},k} \leq T_{\text{eng}}^{\max}(\omega_{\text{eng},k}) \tag{6.25e}$$

$$T_{\text{bsg}}^{\min}(\omega_{\text{bsg},k}) \leq T_{\text{bsg},k} \leq T_{\text{bsg}}^{\max}(\omega_{\text{bsg},k}) \tag{6.25f}$$

$$I^{\min} \leq I_k \leq I^{\max} \tag{6.25g}$$

$$SoC^{\min} \leq SoC_k \leq SoC^{\max} \tag{6.25h}$$

$$0 \leq v_{\text{veh},k} \leq v_{\text{lim},k} \tag{6.25i}$$

$$(t_k, s_k) \notin \mathcal{S}_{\text{red}} \tag{6.25j}$$

$$\mathcal{G}(x_t, z_{H_s}) \in \widetilde{\mathcal{SS}}. \tag{6.25k}$$

Here, s_t is the spatial index corresponding to the distance the ego vehicle has traveled at the time t. $H_s = 20$ is the prediction step in the spatial domain, making the total prediction horizon 200 m. $\mathcal{G} : \mathcal{X} \times \mathcal{Z} \to \mathcal{X}$ is the function that takes the full state

x_t and the terminal optimization state z_{H_s} and determines the predicted terminal full state $\tilde{x}_{t+t_{H_s}}$. For example, suppose there are 15 seconds left in the current green phase and t_{H_s} in the optimization state is 10 seconds, i.e., it takes 10 seconds for the ego vehicle to travel the future 200 m, there will be 5 seconds left in the current green phase at the end of the prediction horizon. The trajectory optimization problem is solved by the Parallel Deterministic Dynamic Programming (PDDP) solver developed in Chapter 4.

To further demonstrate the computational efficiency of PDDP as a solver to the onboard rollout optimization problem, the serial and parallel solvers are used to solve Eqn. 6.25 for two trips sampled from the environment. Table 6.2 shows the mean, variance and the maximum values of the solver time. To perform a comparison of the computation throughput, both implementations were compiled in C++ code. The serial and the parallel implementations were then executed on a desktop PC with a 2.9 GHz Intel Core i7 CPU and an NVIDIA RTX 2080 GPU, respectively. Besides the more than 90% reduction in averaged solver time, the parallel implementation has lower variance and maximum of the solver time as well. This is because the serial implementation has multiple break conditions in the nested loop and these conditions are dependent on the specific driving scenario and SPaT sequence encountered during the route. In the meantime, the parallel implementation goes through all the possible combinations by parallel threads, and thus is less subject to the driving conditions. Although both the CPU and GPU implementations benefit of the improved computational capabilities of typical rapid prototyping units, the benefits from utilizing GPU-based parallel computing are very appealing, especially considering that

Table 6.2: Computation Time Comparison between Serial and Parallel Implementations for Real-time Control

	Trip #1		Trip #2	
	Serial	Parallel	Serial	Parallel
Mean (ms)	1600	98	1638	101
Variance (ms^2)	112	11	287	13
Maximum (ms)	2210	118	2312	123

onboard GPUs are an indispensable development tool for the control of self-driving features in autonomous vehicles [81].

6.4.2 Q Learning

Fig. 6.2 shows the architecture of the neural network associated with the batch Q-learning. Upon receiving the state vector, the sampled traffic light status w_{tfc} are fed to a pre-trained autoencoder with Multilayer Perceptron (MLP) of size [81, 100, 5, 100, 81] for dimensionality reduction. The remaining states along with the actions are concatenated with the latent states from the encoder, and subsequently fed into another MLP of size [200, 100, 50] to output the Q function for critic and perturbation for the actor. The critic and the perturbation do not share parameters in this work.

The actor, originally implemented as a VAE in `https://github.com/sfujim/BCQ`, follows the architecture in Fig. 6.3. Here, the training and the sampling procedures are highlighted by the green and red shaded areas, respectively. In this work, the latent space dimension is selected to be 5, and the encoder and the decoder are both MLPs with 2 layers of 200 hidden units.

During training, the encoder takes the actions and states and outputs the mean and variance of the posterior distribution $q_{\omega_1}(z|A, X)$. Samples generated from the

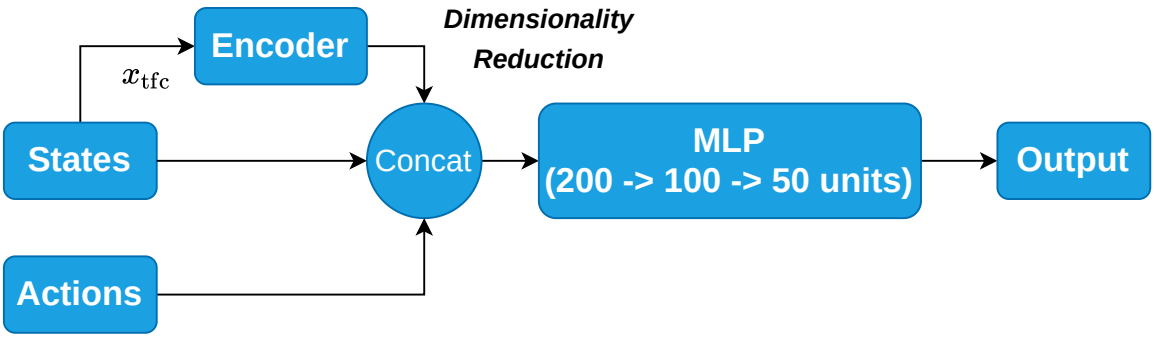


Figure 6.2: The Network Architecture of the Value and Q Functions (Critic).

posterior distribution, concatenated with the states, are subsequently fed to the decoder to generate sampled actions. The sampled actions, along with the mean and variance of the posterior distribution, are then used to minimize the training objective defined in Eqn. (6.16) using reparameterization trick [53]. A detailed explanation of the reparameterization trick is shown in Appendix B.

To accelerate the training and improve generalizability, the state of the vehicle is randomized for every 50 steps in simulation. When the domain randomization occurs, the battery SoC and the vehicle velocity $v_{\text{veh},t}$ are sampled from uniform distributions Uniform$(SoC^{\min}, SoC^{\max})$ and Uniform$(0, v_{\text{lim},t})$, respectively. To guarantee feasibility during the trip, the domain randomization is disabled 200 m within signalized intersections or 1000 m within the destination.

6.4.3 Safe Set Approximation

In the eco-driving problem, two types of constraints can induce feasibility issues, namely, the battery terminal SoC constraint and the constraint imposed by traffic rules at signalized intersections. For the first case, the goal set is considered not

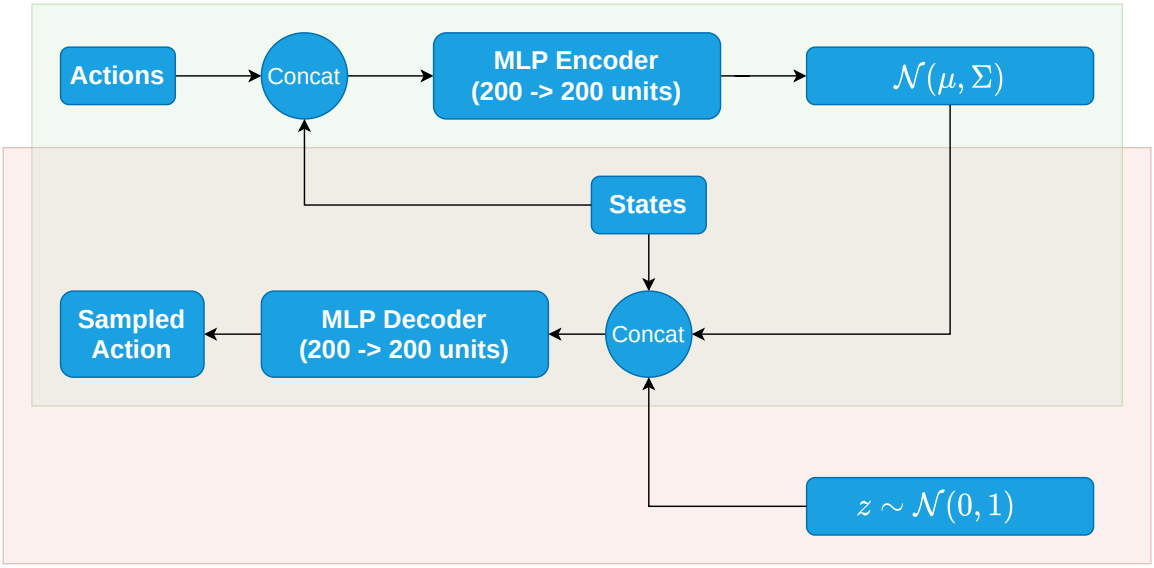


Figure 6.3: The Network Architecture of the VAE.

reached when the vehicle is near the destination, and it cannot sufficiently charge the battery back to SoC^{T} in the remaining distance. For the second case, the trip is considered failed when the vehicle breaks traffic rules at signalized intersections, and infeasibility occurs when the vehicle speed is too high and there is not enough distance to brake to stop in front of a traffic light in red or stop sign. A conservative low speed controller that only uses ICE is used to collect the initial data for the experience replay buffer. During training, only samples from the trips that reach the goal set without violating any constraints are added to the experience replay buffer.

In the eco-driving problem, the sampled traffic light x_{tfc} is binary, while the other variables are continuous. As the PDDP solver also discretizes the continuous state space, the loss of accuracy with the same discretization is considered acceptable. The discretized states as in one-hot form in each dimension are fed into an LSTM network with 50 units sequentially as shown in Fig. 6.4. The outputs from LSTM are then

masked according to the number of categorical classes in each dimension. Finally, the softmax operator ensures the outputs to be a proper conditional probability distribution. As an alternative to LSTM, Causal 1D Convolutional Neural Networks (Causal Conv1D) [74] was also implemented as the network for the autoregressive model. The key difference is that the states in all dimensions can be fed into Causal Conv1D in parallel, whereas each dimension needs to be fed into LSTM sequentially. For applications with long sequences, Causal Conv1D can be more efficient and accurate [8]. For the specific problem, Causal Conv1D shows no noticeable advantage over LSTM in either accuracy or inference speed. As a result, LSTM is chosen as it has fewer hyperparameters.

When the receding horizon ($200m$) in this study is longer than the critical braking distance [120], the vehicle will never violate any constraints imposed by signalized intersections. Nevertheless, using the safe set to constrain the terminal state is still essential for the following reason. At the last step of the receding horizon, the value function $V^{\pi}(\tilde{x}_{t+\Delta t_{H_s}})$ needs to be evaluated numerically for trajectory optimization. Since only the data from the safely executed trips is added into the buffer and there is no penalty mechanism for the constraint violation, the estimation of the critic network is valid only within the safe set and is subject to extrapolation error outside. Although the long receding horizon ensures the feasibility of the actual trajectory regardless of the use of the safe set, the training is subject to instability and the learned performance can significantly deteriorate without the constraint from the safe set.

This effect is shown in Fig. 6.5. Here, the two subplots on top show the optimized trajectories with and without the use of the safe set, respectively. The three curves

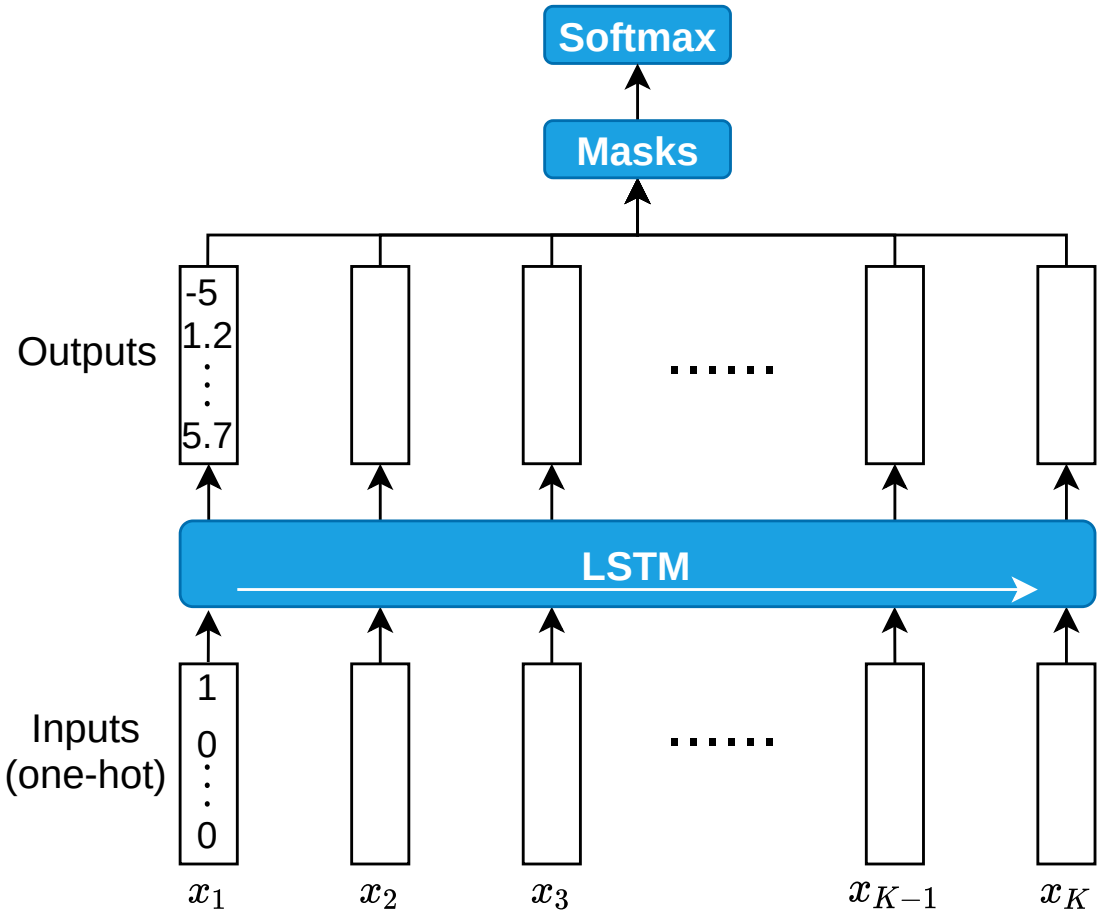


Figure 6.4: The Network Architecture of Recurrent Autoregressive Model.

in each plot are the trajectories from the optimizer at three consecutive seconds. During the first two seconds, the vehicle is more than $200m$ away from the traffic light, and thus the constraint from Eqn. (4.1j) is not considered in the trajectory optimization. The subplots on the bottom show the safety status of the terminal state in the dimension of the vehicle velocity and the time at which it reaches the end of the receding horizon before the signalized intersection appears in the receding horizon. Here, green means the state is considered safe, i.e., $p_\psi(x) \geq \delta$, and red otherwise.

Table 6.3: List of Hyperparameters in SMORL

Parameter	Value
Weighting factor between fuel and time, λ	0.45
Discount factor, γ	0.995
Optimizer	Adam
Learning rate, α	1e-4
Experience buffer size	2e5
Batch size, N	256
Target network update rate, τ	1e-3
Exploration rate, ϵ	0.2
Perturbation range in physical unit, Φ	30 Nm
Sampled actions from the VAE decoder, n	10
Steps per domain randomization	50
LSTM size for the safe set	50

Although the actual trajectories, with or without the safe set, are able to slow down in time to avoid trespassing the red light thanks to the sufficiently long receding horizon, the terminal state without the safe set constraint has a speed of $20m/s$ with $20m$ left before a red light, which is clearly unsafe. Meanwhile, comparing the bottom two subplots, the terminal velocity constrained by the safe set progressively reduces as the vehicle approaches the intersection in the red phase. In addition, given speed limit here is set to $v_{\text{lim}} = 22m/s$, any state with a velocity higher than $22m/s$ is considered unsafe. It can be noticed that the red region on the top right corner is incorrectly considered unsafe (false positive). This is because, by optimality, the agent rarely crosses an intersection in the green phase with low speed, therefore, these false positive regions do not affect the performance.

As a summary for all the implementation details, the hyperparameters are listed in Tab. 6.3.

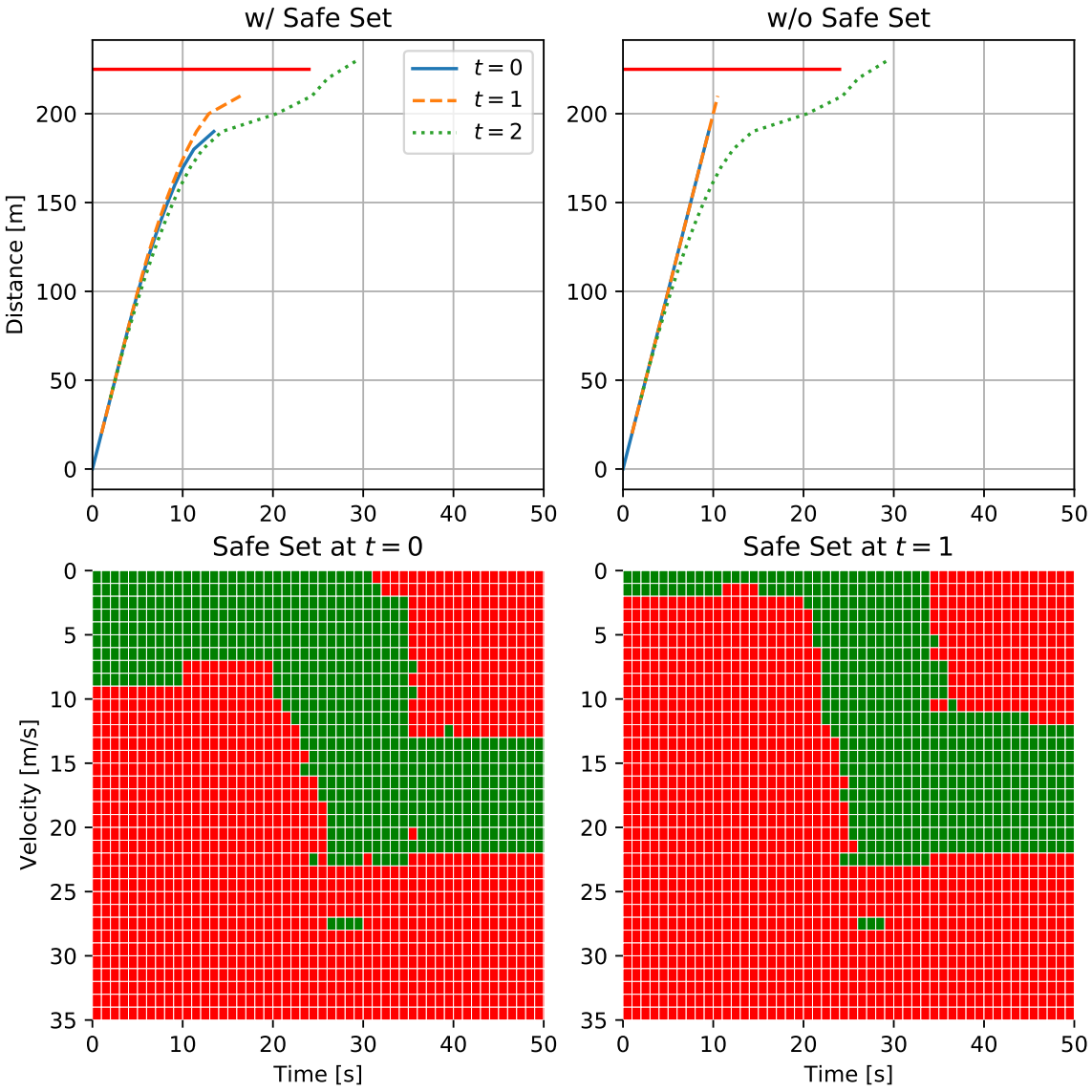


Figure 6.5: The Effect Of the Safe Set on Trajectory Optimization.

6.4.4 Practical Implementation

Fig. 6.6 shows how the three essential components of SMORL interact with each other from a programming perspective. Here, the function approximators in the form of deep neural networks are constructed and trained with PyTorch [79]. The PDDP

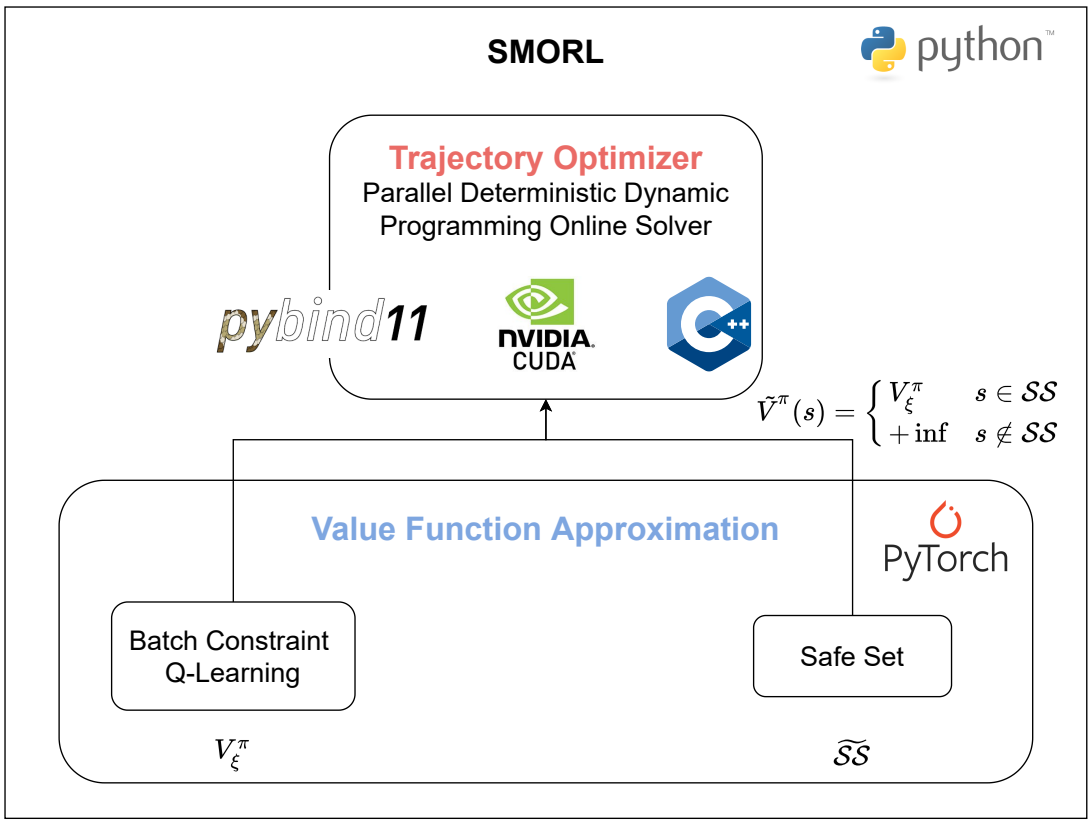


Figure 6.6: The Details of the SMORL Implementation

solver is written with CUDA C++ for real-time performance, and it is bound to Python using *pybind11* [47].

6.5 Results

Both the PDDP optimizer and the neural network training require GPU. To get the results to be shown, the training took 24 hours on a node with an NVIDIA Volta V100 GPU and a 2.9GHz Intel CPU from Ohio Supercomputer Center [16]. As domain randomization is used during training, 5 trips out of 1000 randomly generated trips are repeatedly selected for every 25 training episodes to evaluate the performance

Table 6.4: Fuel Economy, Average Speed and SoC Variance for Baseline, ADP, MFDRL, SMORL and WS Solutions

	Baseline	ADP	MFDRL	SMORL	WS
Fuel Economy mpg	32.4	39.5	41.0	41.6	47.5
Speed Mean m/s	14.1	13.9	12.5	14.0	14.5
SoC Variance $\%^2$	12.1	21.6	18.2	52.6	22.6

of the controller and to quantify the progress of training. During the evaluation, domain randomization and epsilon greedy are both deactivated. Fig. 6.7 shows the evolution of the total costs, fuel economy and average speed of the 5 evaluation trips. Compared to the model-free on-policy method in [120], which takes 80,000 episodes to converge, the sample efficiency of the off-policy model-based method is significantly improved. In the meantime, with the constrained optimization formulation and the safe set, the training quickly learns to respect the constraints imposed by the terminal SoC and signalized intersections. Furthermore, the fact that the agent does not need any extrinsic penalty from constraint violation and is still capable of learning to operate within the safe region significantly simplifies the design and tuning process as deployed in the previous reinforcement learning attempts on eco-driving [59, 82, 120].

Statistically, the performance of the agent trained with SMORL is compared against the four other strategies, namely, the baseline, the ADP and the MFDRL controllers presented in Chapter 5 and the WS solution presented in Chapter 4. The five strategies are evaluated on the 100 random trips as shown in Fig. 3.5, and the fuel economy, average speed and variance of the battery SoC are listed in Tab. 6.4.

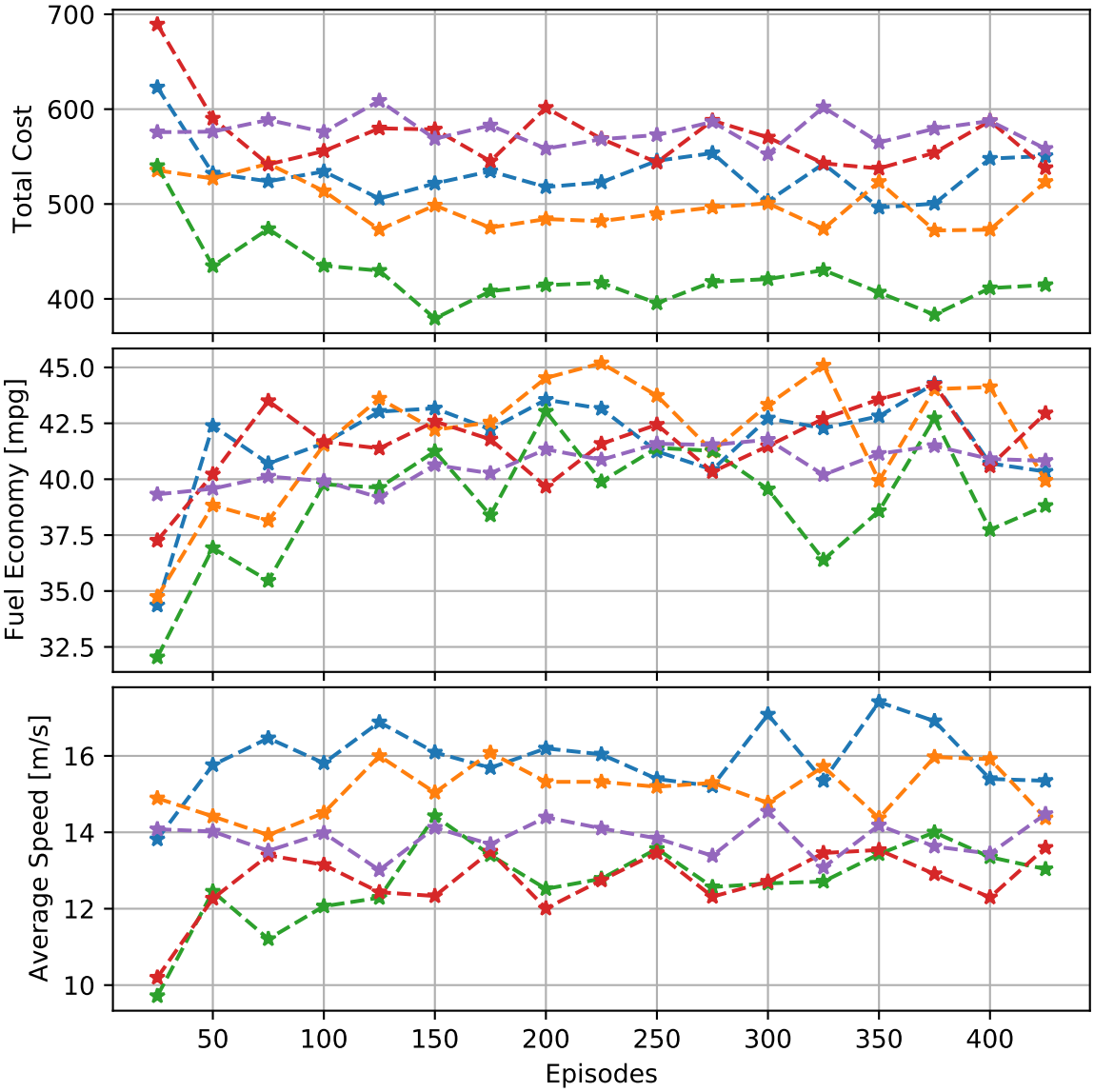


Figure 6.7: The Evolution of the Total Costs, Fuel Economy, and Average Speed of the 5 Evaluation Trips.

Here, the proposed model-based method has a dominant performance in average speed and fuel economy compared to the previously trained MFDRL strategy, which is primarily due to two aspects. First, the online trajectory optimization solved by PDDP guarantees the global optimality within the receding horizon. Compared to

the actions generated from the one-step stochastic policy from a neural network, the solution from trajectory optimization is more accurate and reliable. Second, the fact that there is no extrinsic penalty to assist constraint satisfaction ensures that the agent focuses on learning only the objective of the OCP formulation, i.e., the weighted sum of the trip time and fuel consumption, instead of a carefully designed yet delicate surrogate learning objective. Compared to the baseline EDM driver, SMORL agent consumes 21.8% less total fuels while maintaining a comparable average speed. The benefit in fuel economy is achieved by avoiding unnecessary acceleration events and by taking advantage of a wider range of battery capacity as indicated by the higher SoC variance. Despite the fact that both SMORL and ADP strategies solve a rollout optimization problem, the SMORL strategy demonstrates a dominant performance again thanks to the learned value function. Compared to the value function obtained from solving a deterministic full-route problem prior to departure, the value function learned in SMORL is able to adapt to different stochastic realizations in real-time and thus utilize the connected information more intelligently.

In Fig. 6.8, the average vehicle speed and the fuel economy of each trip are plotted against the traffic light density. As the WS solution calculates the global optimal solution with the knowledge of the full trip, it is able to navigate among the traffic lights accordingly, as indicated by the surprisingly increasing fuel economy. This can be due to the fact that when there are more traffic lights, the vehicle is forced to operate with a lower speed and lower fuel consumption condition. On the other hand, as the baseline driver has limited line-of-sight [35] and the ADP, MFDRL and SMORL controllers have limited DSRC sensing range, the fuel economy decreases as the traffic light density increases. Nevertheless, as indicated by the slope of the fitted

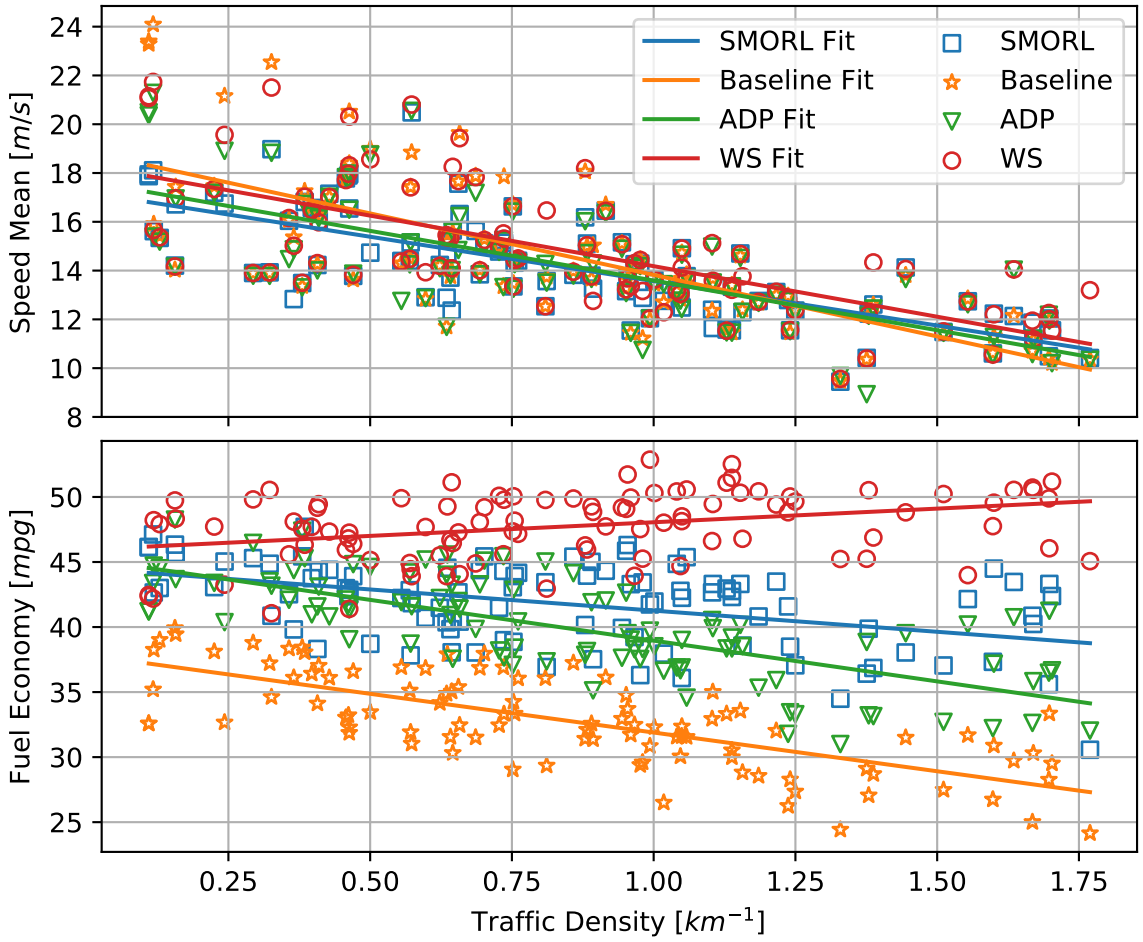


Figure 6.8: The Variation of the Average Speed and the Fuel Economy against Traffic Light Density for Baseline, SMORL and WS Solution

curve, the fuel economy of SMORL is less affected by the increase of the traffic light density compared to the baseline and ADP controller.

Fig. 6.9 shows the comparison among the baseline, ADP, SMORL and the WS solution on a specific testing trip. For this specific trip, while the difference in trip time is within $3s$, SMORL consumes 24.7% and 11.0% less fuel compared to the baseline and the ADP strategies, respectively.

While SMORL demonstrates some merits similar to the WS solution, its inferiority to the WS solution is primarily due to the fact that only the SPaTs from the upcoming intersection are available to the controller. In this trip specifically, the WS solution deployed a constant low-speed strategy from $\sim 1,700m$ all the way till the end of the trip. Although it was behind the other two at the intersection at $4,300m$ by one red phase, it was able to eventually catch up by not having to waste any time at stops. While knowing the SPaTs from the entire trip is unrealistic and subject to uncertainties, the SMORL was able to save a significant amount of fuel compared to the baseline and the ADP strategy by 1. decelerating ahead of time, 2. decelerating with more energy recuperation using the electric motor, 3. avoiding unnecessary high speed within the connectivity range. Additional comparisons among the four strategies are shown in Appendix C.

In addition, an ablation study is included in Appendix D to show the necessities and benefits of each component in the proposed SMORL framework.

6.6 Summary

In this chapter, a safe-critical model-based off-policy reinforcement learning algorithm SMORL is proposed. The algorithm is applied to the eco-driving problem for CAHEVs. Compared to the previous model-free attempts in Chapter 5 and in the literature, the method does not require any extrinsic rewarding mechanism, and thus, greatly simplifies the design process and improves the final performance. With the online constrained optimization formulation and the approximate safe set, the learned strategy is capable of satisfying the constraints in the prediction horizon and restricting the state within the approximate safe set, which is an approximation to the

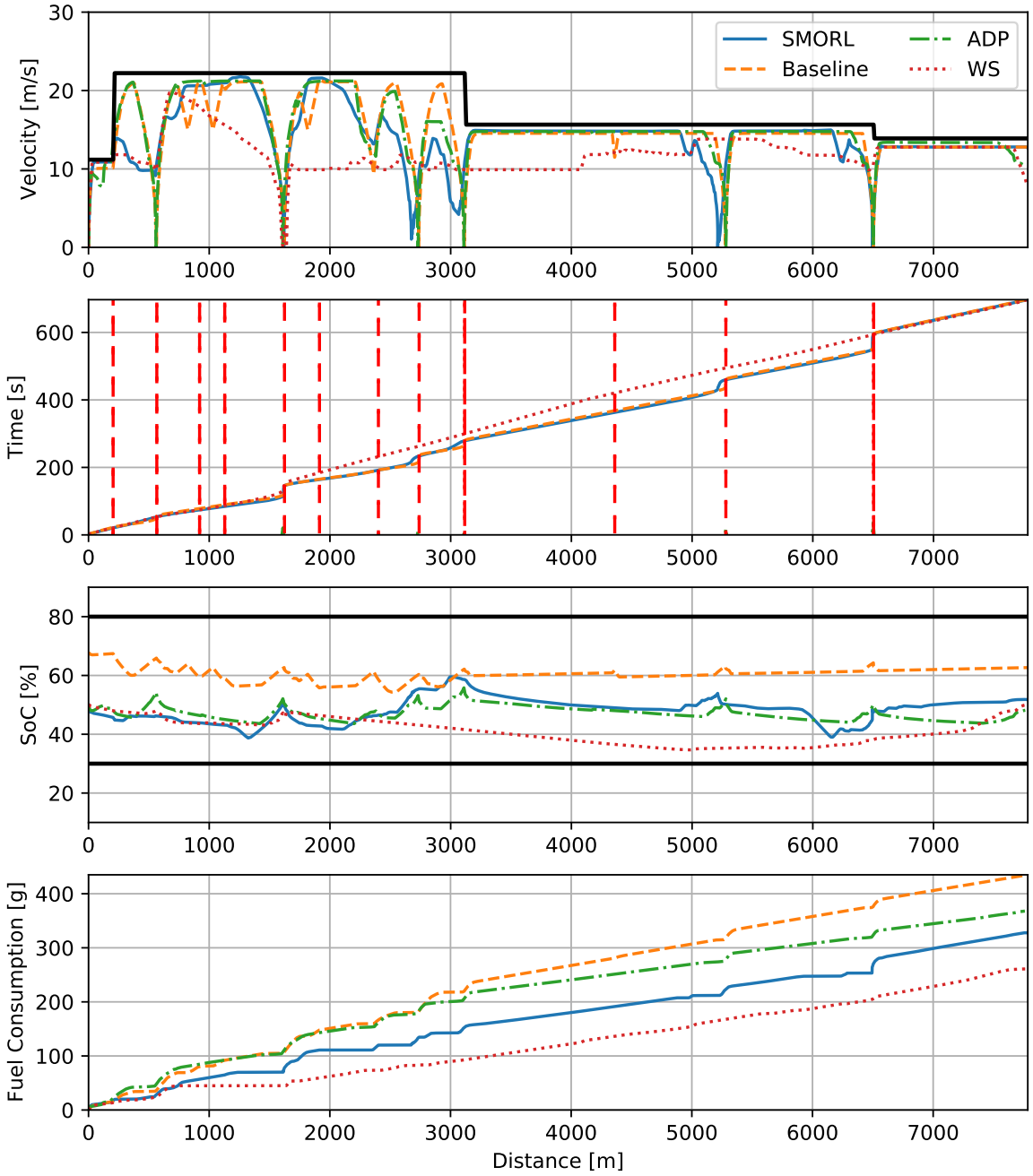


Figure 6.9: The Trajectory Comparison among Baseline, SMORL and WS.

robust control invariant set. The performance of the strategy trained with SMORL is

compared to a baseline strategy representing human drivers' behavior over 100 randomly generated trips in Columbus, OH, US. With a comparable average speed, the strategy from SMORL consumes approximately 22% less fuel.

Chapter 7: Conclusion

This work studies the eco-driving problem for Connected and Automated Hybrid Electric Vehicles (CAHEVs) with the capability to autonomously and intelligently pass signalized intersections. To improve the performance of the control strategy and to reduce the onboard computational requirement, this work developed a complete Reinforcement Learning (RL) framework for the eco-driving problem.

To enable robust training and comprehensive validation, EcoSIM, a distributed RL environment that consists of a large-scale traffic simulator and an experimentally validated powertrain and vehicle dynamics model is developed. Subsequently, a model free deep RL (MFDRL) algorithm and a model based deep RL (MBDRL) algorithm are developed.

The MFDRL algorithm models the problem as a Partially Observable Markov Decision Process, and an agent is trained via an actor critic algorithm Proximal Policy Optimization (PPO) with Long Short-term Memory as the function approximator. The resulting explicit policy is computationally appealing as it does not require solving any onboard optimization. Evaluated on 100 trips generated from EcoSIM, the trained policy demonstrates a 17.5% fuel consumption reduction while keeping the travel time comparable compared to the baseline strategy.

To further overcome the limitations exhibited by the MFDRL approach, namely, complex reward design, low sample efficiency and lack of safety guarantee, a novel MBDRL algorithm Safe Model-based Off-policy Reinforcement Learning (SMORL) is proposed in this work. The approach leverages the Model Predictive Control formulation to impose constraint satisfaction and solve the optimization problem with the fast massive parallel dynamic programming solver developed in this work. The optimal terminal cost function is learned via the interaction with EcoSIM. The learning algorithm is off-policy to improve sample efficiency. To further improve the performance and to provide the feasibility guarantee, the safe set, as an approximation to the robust control invariant set, is learned as a generative model. Evaluated on the same 100 trips, the SMORL agent is able to achieve a 21% fuel saving compared to the baseline strategy, outperforming the MFDRL agent and the non-learning-based optimal controller from literature.

While the demonstration of the algorithm focuses on the eco-driving problem, I believe it can be applied to many other real-world problems, in particular to those with known system dynamics, such as robotics and autonomous driving. Future works include 1. incorporating SPaT information broadcasted from multiple upcoming signalized intersections, 2. incorporating surrounding vehicles into the model, and accordingly update the optimal control problem, 3. releasing the deterministic assumption within the prediction horizon.

Appendix A: Reward Function Design for MFDRL

In general, the process of designing the reward function is an iterative process and requires tuning. Some key takeaways are listed below.

1. Normalize the scale of the reward function such that the numerical value is between $[-1, 1]$ [107].

2. The reward items r_{vel}, r_{batt} and r_{tfc} are associated with the constraints, and they should be at least one order of magnitude higher than r_{obj}.

3. Rewards from the environment should reflect incremental incentives/penalties. For example, rewards associated with traffic lights are sparse, meaning that the agent receives these rewards periodically and needs a sequence of actions to avoid the large penalty. Eqn.(5.13) ensures the penalty for violating the traffic condition is proportional to how bad was the violation.

4. Penalties related to constraints should be orders larger than those related to performance.

Appendix B: Reparameterization Trick

Many machine learning tasks require finding the gradient of a function approximator w.r.t. the parameters under certain fixed distribution:

$$\begin{aligned}\nabla_\theta \mathbb{E}_{p(z)}[f_\theta(z)] &= \nabla_\theta \left[\int_z p(z) f_\theta(z) dz\right] \\ &= \int_z p(z) \left[\nabla_\theta f_\theta(z)\right] dz \\ &= \mathbb{E}_{p_z}\left[\nabla_\theta f_\theta(z)\right]\end{aligned} \tag{B.1}$$

In this case, the procedure for calculating the gradient of the expectation is straightforward as one can first sample data from the underlying distribution, then calculate the gradient for each sample, and finally take the expectation of the gradient terms.

However, when the distribution that the data is sampled from is parameterized by the parameter θ, the previous derivation does not hold since

$$\begin{aligned}\nabla_\theta \mathbb{E}_{p_\theta(z)}[f(z)] &= \nabla_\theta \left[\int_z p_\theta(z) f(z) dz\right] \\ &= \int_z f(z) \nabla_\theta p_\theta(z) dz.\end{aligned} \tag{B.2}$$

As this can no longer be turned into an expectation, the evaluation of the term becomes intractable. This issue rises under two scenarios in this work.

1. In policy gradient method in RL, the gradient of the average cumulative return of a policy w.r.t. the parameters of the policy,

2. In the training of VAE, the gradient of the variational lower bound w.r.t. the parameters of the encoder.

Under the first scenario, Policy Gradient Theorem [117] is typically used. For the second scenario, the reparameterization trick, which is also originally proposed in [117] and recently made more well-known in [53], is typically used.

First, a random variable $\epsilon \sim q \sim \mathcal{N}(0,1)$ is first defined. Assuming the distribution p_θ is a Gaussian distribution, we have

$$z \sim p_\theta \sim \mathcal{N}(\mu_\theta, \sigma_\theta) \sim \mu_\theta + \sigma_\theta^2 \mathcal{N}(0,1). \tag{B.3}$$

Subsequently, drawing samples from $\epsilon \sim \mathcal{N}(0,1)$ is equivalent to drawing samples from $z \sim p_\theta$ as we can define z to be

$$z = \mu_\theta + \sigma_\theta \epsilon \tag{B.4}$$

Accordingly, we have

$$\begin{aligned}
\nabla_\theta \mathbb{E}_{p_\theta(z)}[f(z)] &= \nabla_\theta \left[\int_z p_\theta(z) f(z) dz \right] \\
&= \nabla_\theta \left[\int_\epsilon q(\epsilon) f(\mu_\theta + \sigma_\theta \epsilon) d\epsilon \right] \\
&= \left[\int_\epsilon q(\epsilon) \nabla_\theta f(\mu_\theta + \sigma_\theta \epsilon) d\epsilon \right] \\
&= \mathbb{E}_{\epsilon \sim q} \left[\nabla_\theta f(\mu_\theta + \sigma_\theta \epsilon) \right]
\end{aligned} \tag{B.5}$$

As a result, we can now simply evaluate the gradient of each sample $\nabla_\theta f(\mu_\theta + \sigma_\theta \epsilon)$, and then calculate the expectation.

Note that in the derivation here, it is assumed that $p_\theta \sim \mathcal{N}(\mu_t heta, \sigma)$ for demonstration purpose. While this is consistent with the implementation in this work, another two types of assumption, tractable inverse CDF and composition, are also listed in [53].

Appendix C: Additional Comparison among Strategies

Here, the comparison of two additional trips is shown. The trip shown in Fig. C.1 contains a large number of signalized intersections. As indicated by Fig. 6.8, the gap between SMORL and the baseline and the gap between the wait-and-see solution and SMORL are both amplified by the high traffic density. The trip shown in Fig. C.2 has a very low traffic density and the speed limits higher. In such case, the difference between SMORL and the wait-and-see solution becomes less noticeable. Meanwhile, SMORL was still able to consume less fuel by using the capacity of the battery more efficiently.

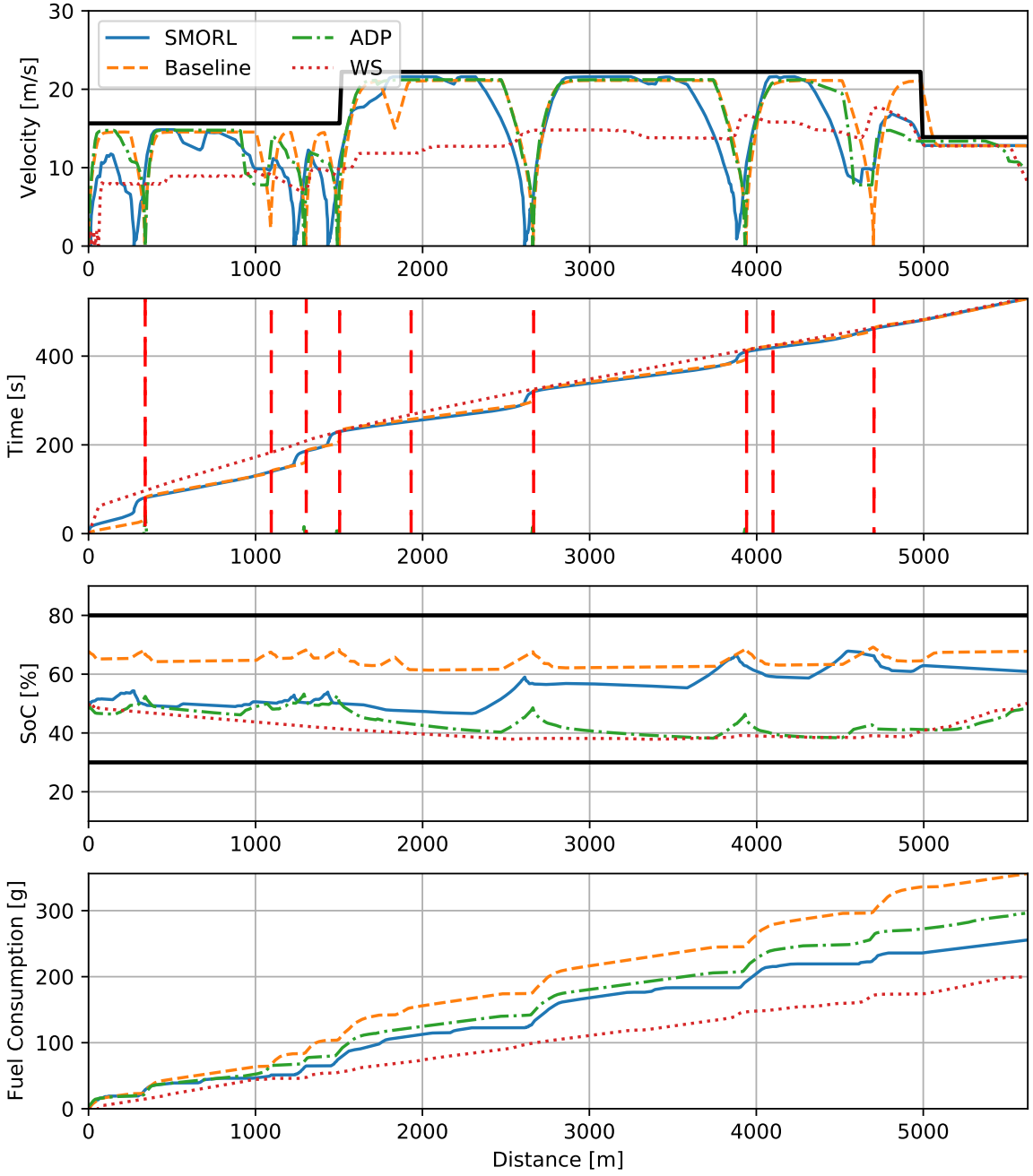


Figure C.1: Comparison for High-density Low-speed Scenario.

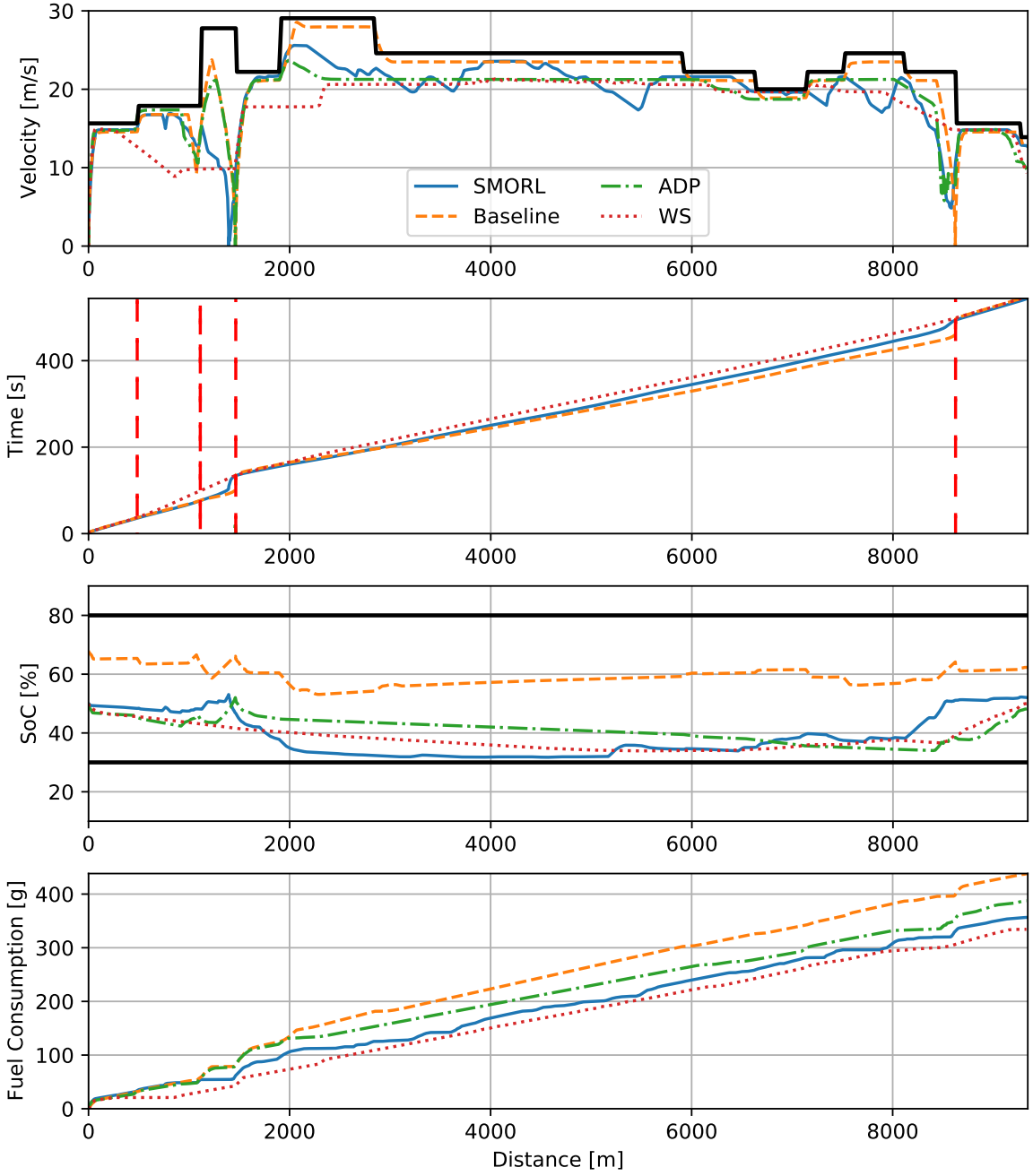


Figure C.2: Comparison for Low-density High-speed Scenario.

Table D.1: Ablation Study for SMORL

	Safe Set	Q-learning	Fuel Economy mpg	Average Speed m/s	Normalized Cost
1		None	43.1	11.2	100
2		TD3	39.1	13.9	88.0
3	✔	TD3	39.9	13.3	90.5
4		BCQ	38.5	14.3	87.2
5	✔	BCQ	41.6	14.0	86.5

Appendix D: Ablation Study for SMORL

In this part, the full SMORL algorithm is compared with four intermediate algorithms. All the algorithms presented below use trajectory optimization solved via PDDP solver. Tab. D.1 shows the difference in configuration and compares the trained final performance over the 100 trips used for testing. Here, safe set and BCQ both have a positive impact on the trained performance. In fact, the native combination of TD3 and trajectory optimization (Config. 2) does not provide any show any significant improvement over trajectory optimization only (Config. 1). In addition, without the use of the safe set, the controller will deplete the battery SoC to $SoC^{\min}$ at the end of the trip as the terminal state constraint cannot be considered unless with the help of extrinsic penalties.

Bibliography

[1] CUDA C++ programming guide. `https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html`, August 2020.

[2] AASHTO AASHTO. Policy on geometric design of highways and streets. *American Association of State Highway and Transportation Officials, Washington, DC*, 1(990):158, 2001.

[3] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[4] Osman D Altan, Guoyuan Wu, Matthew J Barth, Kanok Boriboonsomsin, and John A Stark. Glidepath: Eco-friendly automated approach and departure at signalized intersections. *IEEE Transactions on Intelligent Vehicles*, 2(4):266–277, 2017.

[5] B. Asadi and A. Vahidi. Predictive cruise control: Utilizing upcoming traffic signal information for improving fuel economy and reducing trip time. *IEEE Transactions on Control Systems Technology*, 19(3):707–714, 2011.

[6] Behrang Asadi and Ardalan Vahidi. Predictive cruise control: Utilizing upcoming traffic signal information for improving fuel economy and reducing trip time. *IEEE transactions on control systems technology*, 19(3):707–714, 2010.

[7] Sangjae Bae, Yongkeun Choi, Yeojun Kim, Jacopo Guanetti, Francesco Borrelli, and Scott Moura. Real-time ecological velocity planning for plug-in hybrid vehicles with partial communication to traffic lights. *arXiv preprint arXiv:1903.08784*, 2019.

[8] Shaojie Bai, J Zico Kolter, and Vladlen Koltun. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *arXiv preprint arXiv:1803.01271*, 2018.

[9] Andrew G Barto, Richard S Sutton, and Charles W Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE transactions on systems, man, and cybernetics*, (5):834–846, 1983.

[10] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

[11] Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemyslaw Debiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, et al. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*, 2019.

[12] Dimitri P Bertsekas. *Dynamic programming and optimal control*, volume 1. Athena scientific Belmont, MA, 2005.

[13] Dimitri P Bertsekas and John N Tsitsiklis. Neuro-dynamic programming: an overview. In *Proceedings of 1995 34th IEEE Conference on Decision and Control*, volume 1, pages 560–564. IEEE, 1995.

[14] John R Birge and Francois Louveaux. *Introduction to stochastic programming*. Springer Science & Business Media, 2011.

[15] Francesco Borrelli, Alberto Bemporad, and Manfred Morari. *Predictive control for linear and hybrid systems*. Cambridge University Press, 2017.

[16] Ohio Supercomputer Center. Ohio supercomputer center, 1987.

[17] Weimin Cheng. Avoiding race conditions in cuda, May 2020. [Online; posted 4-May-2020].

[18] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[19] Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. *arXiv preprint arXiv:1805.12114*, 2018.

[20] Henggang Cui, Thi Nguyen, Fang-Chieh Chou, Tsung-Han Lin, Jeff Schneider, David Bradley, and Nemanja Djuric. Deep kinematic models for kinematically feasible vehicle trajectory predictions. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 10563–10569. IEEE, 2020.

[21] Marc Deisenroth and Carl E Rasmussen. Pilco: A model-based and data-efficient approach to policy search. In *Proceedings of the 28th International Conference on machine learning (ICML-11)*, pages 465–472. Citeseer, 2011.

[22] Marc Peter Deisenroth, Dieter Fox, and Carl Edward Rasmussen. Gaussian processes for data-efficient learning in robotics and control. *IEEE transactions on pattern analysis and machine intelligence*, 37(2):408–423, 2013.

[23] Stefan Depeweg, José Miguel Hernández-Lobato, Finale Doshi-Velez, and Steffen Udluft. Learning and policy search in stochastic dynamical systems with bayesian neural networks. *arXiv preprint arXiv:1605.07127*, 2016.

[24] Shreshta Rajakumar Deshpande, Shobhit Gupta, Abhishek Gupta, and Marcello Canova. Real-time eco-driving control in electrified connected and autonomous vehicles using approximate dynamic programming. *arXiv preprint arXiv:2108.02652*, 2021.

[25] Shreshta Rajakumar Deshpande, Shobhit Gupta, Dennis Kibalama, Nicola Pivaro, Marcello Canova, Giorgio Rizzoni, Karim Aggoune, Pete Olin, and John Kirwan. In-vehicle test results for advanced propulsion and vehicle system controls using connected and automated vehicle information. Technical report, SAE Technical Paper, 2021.

[26] Laurent Dinh, David Krueger, and Yoshua Bengio. Nice: Non-linear independent components estimation. *arXiv preprint arXiv:1410.8516*, 2014.

[27] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using real nvp. *arXiv preprint arXiv:1605.08803*, 2016.

[28] Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International Conference on Machine Learning*, pages 1587–1596. PMLR, 2018.

[29] Scott Fujimoto, David Meger, and Doina Precup. Off-policy deep reinforcement learning without exploration. In *International Conference on Machine Learning*, pages 2052–2062. PMLR, 2019.

[30] Mathieu Germain, Karol Gregor, Iain Murray, and Hugo Larochelle. Made: Masked autoencoder for distribution estimation. In *International Conference on Machine Learning*, pages 881–889. PMLR, 2015.

[31] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[32] Bo Gu and Giorgio Rizzoni. An adaptive algorithm for hybrid electric vehicle energy management based on driving pattern recognition. In *2006 ASME International Mechanical Engineering Congress and Exposition*, pages 249–258, 2006.

[33] Ge Guo and Yunpeng Wang. Eco-driving of freight vehicles with signal priority on congested arterial roads. *IEEE Transactions on Vehicular Technology*, 70(5):4225–4237, 2021.

[34] Lulu Guo, Bingzhao Gao, Ying Gao, and Hong Chen. Optimal energy management for hevs in eco-driving applications using bi-level mpc. *IEEE Transactions on Intelligent Transportation Systems*, 18(8):2153–2162, 2016.

[35] Shobhit Gupta, Shreshta R Deshpande, Punit Tulpule, Marcello Canova, and Giorgio Rizzoni. An enhanced driver model for evaluating fuel economy on real-world routes. *IFAC-PapersOnLine*, 52(5):574–579, 2019.

[36] Shobhit Gupta, Shreshta Rajakumar Deshpande, Daniela Tufano, Marcello Canova, Giorgio Rizzoni, Karim Aggoune, Pete Olin, and John Kirwan. Estimation of fuel economy on real-world routes for next-generation connected and automated hybrid powertrains. Technical report, SAE Technical Paper, 2020.

[37] Lino Guzzella, Antonio Sciarretta, et al. *Vehicle propulsion systems*, volume 1. Springer.

[38] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pages 1861–1870. PMLR, 2018.

[39] Jihun Han, Ardalan Vahidi, and Antonio Sciarretta. Fundamentals of energy efficient driving for combustion engine and electric vehicles: An optimal control perspective. *Automatica*, 103:558–572, 2019.

[40] Nicolas Heess, Jonathan J Hunt, Timothy P Lillicrap, and David Silver. Memory-based control with recurrent neural networks. *arXiv preprint arXiv:1512.04455*, 2015.

[41] Nicolas Heess, Dhruva TB, Srinivasan Sriram, Jay Lemmon, Josh Merel, Greg Wayne, Yuval Tassa, Tom Erez, Ziyu Wang, SM Eslami, et al. Emergence of locomotion behaviours in rich environments. *arXiv preprint arXiv:1707.02286*, 2017.

[42] Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. In *Thirty-second AAAI conference on artificial intelligence*, 2018.

[43] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[44] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

[45] David R Hunter and Kenneth Lange. A tutorial on mm algorithms. *The American Statistician*, 58(1):30–37, 2004.

[46] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015.

[47] Wenzel Jakob, Jason Rhinelander, and Dean Moldovan. pybind11 — seamless operability between c++11 and python, 2016. https://github.com/pybind/pybind11.

[48] Yijuan Jiang and Xiang Li. Travel time prediction based on historical trajectory data. *Annals of GIS*, 19(1):27–35, 2013.

[49] Qiu Jin, Guoyuan Wu, Kanok Boriboonsomsin, and Matthew J Barth. Power-based optimal longitudinal control for a connected eco-driving system. *IEEE Transactions on Intelligent Transportation Systems*, 17(10):2900–2910, 2016.

[50] Ameya Joshi. Review of vehicle engine efficiency and emissions. *SAE International Journal of Advances and Current Practices in Mobility*, 1(2019-01-0314):734–761, 2019.

[51] Napat Karnchanachari, Miguel Iglesia Valls, David Hoeller, and Marco Hutter. Practical reinforcement learning for mpc: Learning from sparse objectives in under an hour on a real robot. In *Learning for Dynamics and Control*, pages 211–224. PMLR, 2020.

[52] Andrej Karpathy, Justin Johnson, and Li Fei-Fei. Visualizing and understanding recurrent networks. *arXiv preprint arXiv:1506.02078*, 2015.

[53] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[54] Daniel Krajzewicz, Georg Hertkorn, C Rössel, and Peter Wagner. An example of microscopic car models validation using the open source traffic simulation sumo. In *Proceedings of Simulation in Industry, 14th European Simulation Symposium*, pages 318–322, 2002.

[55] Stefan Krauß. Microscopic modeling of traffic flow: Investigation of collision free vehicle dynamics. 1998.

[56] Thanard Kurutach, Ignasi Clavera, Yan Duan, Aviv Tamar, and Pieter Abbeel. Model-ensemble trust-region policy optimization. *arXiv preprint arXiv:1802.10592*, 2018.

[57] Sergey Levine and Vladlen Koltun. Guided policy search. In *International conference on machine learning*, pages 1–9. PMLR, 2013.

[58] Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643*, 2020.

[59] Guoqiang Li and Daniel Görges. Ecological adaptive cruise control for vehicles with step-gear transmission based on reinforcement learning. *IEEE Transactions on Intelligent Transportation Systems*, 2019.

[60] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[61] Chan-Chiao Lin, Huei Peng, and JW Grizzle. A stochastic control strategy for hybrid electric vehicles. In *2004 American Control Conference*, pages 4710–4715, 2004.

[62] Long-Ji Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning*, 8(3-4):293–321, 1992.

[63] Michael Livshiz, Minghui Kao, and Anthony Will. Validation and calibration process of powertrain model for engine torque control development. Technical report, SAE Technical Paper, 2004.

[64] Pablo Alvarez Lopez, Michael Behrisch, Laura Bieker-Walz, Jakob Erdmann, Yun-Pang Flötteröd, Robert Hilbrich, Leonhard Lücken, Johannes Rummel, Peter Wagner, and Evamarie Wießner. Microscopic traffic simulation using sumo. In *The 21st IEEE International Conference on Intelligent Transportation Systems*. IEEE, 2018.

[65] Kendall Lowrey, Aravind Rajeswaran, Sham Kakade, Emanuel Todorov, and Igor Mordatch. Plan online, learn offline: Efficient learning and exploration via model-based control. *arXiv preprint arXiv:1811.01848*, 2018.

[66] Yuping Luo, Huazhe Xu, Yuanzhi Li, Yuandong Tian, Trevor Darrell, and Tengyu Ma. Algorithmic framework for model-based deep reinforcement learning with theoretical guarantees. *arXiv preprint arXiv:1807.03858*, 2018.

[67] Peter Marbach and John N Tsitsiklis. Simulation-based optimization of markov reward processes. *IEEE Transactions on Automatic Control*, 46(2):191–209, 2001.

[68] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937, 2016.

[69] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

[70] Philipp Moritz, Robert Nishihara, Stephanie Wang, Alexey Tumanov, Richard Liaw, Eric Liang, Melih Elibol, Zongheng Yang, William Paul, Michael I Jordan, et al. Ray: A distributed framework for emerging {AI} applications. In *13th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 18)*, pages 561–577, 2018.

[71] Scott Jason Moura, Hosam K Fathy, Duncan S Callaway, and Jeffrey L Stein. A stochastic optimal control approach for power management in plug-in hybrid electric vehicles. *IEEE Transactions on control systems technology*, 19(3):545–555, 2010.

[72] Andrew Y Ng and Michael I Jordan. On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. In *Advances in neural information processing systems*, pages 841–848, 2002.

[73] Pete Olin, Karim Aggoune, Li Tang, Keith Confer, John Kirwan, Shreshta Rajakumar Deshpande, Shobhit Gupta, Punit Tulpule, Marcello Canova, and Giorgio Rizzoni. Reducing fuel consumption by using information from connected and automated vehicle modules to optimize propulsion system control. Technical report, SAE Technical Paper, 2019.

[74] Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray

Kavukcuoglu. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016.

[75] OpenStreetMap contributors. Planet dump retrieved from https://planet.osm.org . `https://www.openstreetmap.org`, 2017.

[76] Engin Ozatay, Simona Onori, James Wollaeger, Umit Ozguner, Giorgio Rizzoni, Dimitar Filev, John Michelini, and Stefano Di Cairano. Cloud-based velocity profile optimization for everyday driving: A dynamic-programming-based solution. *IEEE Transactions on Intelligent Transportation Systems*, 15(6):2491–2505, 2014.

[77] George Papamakarios, Theo Pavlakou, and Iain Murray. Masked autoregressive flow for density estimation. *arXiv preprint arXiv:1705.07057*, 2017.

[78] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *International conference on machine learning*, pages 1310–1318. PMLR, 2013.

[79] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[80] F Payri, José M Luján, Carlos Guardiola, and B Pla. A challenging future for the ic engine: new technologies and the control role. *Oil & Gas Science and Technology–Revue d'IFP Energies nouvelles*, 70(1):15–30, 2015.

[81] Wilson Perez, Amit Ruhela, and Punit Tulpule. Benchmarking computational time of dynamic programming for autonomous vehicle powertrain control. *SAE Technical Paper*, (2020-01-0968), 2020.

[82] Andrea Pozzi, Sangjae Bae, Yongkeun Choi, Francesco Borrelli, Davide M Raimondo, and Scott Moura. Ecological velocity planning through signalized intersections: A deep reinforcement learning approach. In *2020 59th IEEE Conference on Decision and Control (CDC)*, pages 245–252. IEEE, 2020.

[83] Peng Rong and Massoud Pedram. An analytical model for predicting the remaining battery capacity of lithium-ion batteries. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 14(5):441–451, 2006.

[84] Ugo Rosolia and Francesco Borrelli. Sample-based learning model predictive control for linear uncertain systems. In *2019 IEEE 58th Conference on Decision and Control (CDC)*, pages 2702–2707. IEEE, 2019.

[85] Bart Saerens. Optimal control based eco-driving-theoretical approach and practical applications. *Ph. D. Thesis, Katholieke Universiteit Leuven*, 2012.

[86] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897, 2015.

[87] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015.

[88] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[89] Antonio Sciarretta, Giovanni De Nunzio, and Luis Leon Ojeda. Optimal eco-driving control: Energy-efficient driving of road vehicles as an optimal control problem. *IEEE Control Systems Magazine*, 35(5):71–90, 2015.

[90] Shai Shalev-Shwartz, Shaked Shammah, and Amnon Shashua. Safe, multi-agent, reinforcement learning for autonomous driving. *arXiv preprint arXiv:1610.03295*, 2016.

[91] Junqing Shi, Fengxiang Qiao, Qing Li, Lei Yu, and Yongju Hu. 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, 2018.

[92] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

[93] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. 2014.

[94] Jeffrey S Simonoff. *Smoothing methods in statistics*. Springer Science & Business Media, 2012.

[95] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[96] Chao Sun, Jacopo Guanetti, Francesco Borrelli, and Scott Moura. Optimal eco-driving control of connected and autonomous vehicles through signalized intersections. *IEEE Internet of Things Journal*, 2020.

[97] Olle Sundström, Daniel Ambühl, and Lino Guzzella. On implementation of dynamic programming for optimal control problems with final state constraints. *Oil & Gas Science and Technology–Revue de l'Institut Français du Pétrole*, 65(1):91–102, 2010.

[98] Olle Sundstrom and Lino Guzzella. A generic dynamic programming matlab function. In *Control Applications,(CCA) & Intelligent Control,(ISIC), 2009 IEEE*, pages 1625–1630. IEEE, 2009.

[99] Richard S Sutton. Dyna, an integrated architecture for learning, planning, and reacting. *ACM Sigart Bulletin*, 2(4):160–163, 1991.

[100] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[101] Richard S Sutton, David A McAllester, Satinder P Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, pages 1057–1063, 2000.

[102] Yuval Tassa, Tom Erez, and Emanuel Todorov. Synthesis and stabilization of complex behaviors through online trajectory optimization. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4906–4913. IEEE, 2012.

[103] Brijen Thananjeyan, Ashwin Balakrishna, Ugo Rosolia, Joseph E Gonzalez, Aaron Ames, and Ken Goldberg. Abc-lmpc: Safe sample-based learning mpc for stochastic nonlinear dynamical systems with adjustable boundary conditions. In *International Workshop on the Algorithmic Foundations of Robotics*, pages 1–17. Springer, 2020.

[104] Brijen Thananjeyan, Ashwin Balakrishna, Ugo Rosolia, Felix Li, Rowan McAllister, Joseph E Gonzalez, Sergey Levine, Francesco Borrelli, and Ken Goldberg. Safety augmented value estimation from demonstrations (saved): Safe deep model-based rl for sparse cost robotic tasks. *IEEE Robotics and Automation Letters*, 5(2):3612–3619, 2020.

[105] Ardalan Vahidi and Antonio Sciarretta. Energy saving potentials of connected and automated vehicles. *Transportation Research Part C: Emerging Technologies*, 95:822–843, 2018.

[106] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double Q-learning. In *2016 AAAI Conference on Artificial Intelligence*.

[107] Hado P van Hasselt, Arthur Guez, Matteo Hessel, Volodymyr Mnih, and David Silver. Learning values across many orders of magnitude. In *Advances in Neural Information Processing Systems*, pages 4287–4295, 2016.

[108] Robin Verschueren, Gianluca Frison, Dimitris Kouzoupis, Jonathan Frey, Niels van Duijkeren, Andrea Zanelli, Branimir Novoselnik, Thivaharan Albin, Rien Quirynen, and Moritz Diehl. acados: a modular open-source framework for fast embedded optimal control, 2020.

[109] Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev, et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, 2019.

[110] Nianfeng Wan, Ardalan Vahidi, and Andre Luckow. Optimal speed advisory for connected vehicles in arterial roads and the impact on mixed traffic. *Transportation Research Part C: Emerging Technologies*, 69:548–563, 2016.

[111] Meng Wang, Winnie Daamen, Serge Hoogendoorn, and Bart Van Arem. Potential impacts of ecological adaptive cruise control systems on traffic and environment. *IET Intelligent Transport Systems*, 8(2):77–86, 2014.

[112] Tingwu Wang, Xuchan Bao, Ignasi Clavera, Jerrick Hoang, Yeming Wen, Eric Langlois, Shunshi Zhang, Guodong Zhang, Pieter Abbeel, and Jimmy Ba. Benchmarking model-based reinforcement learning. *arXiv preprint arXiv:1907.02057*, 2019.

[113] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Hasselt, Marc Lanctot, and Nando Freitas. Dueling network architectures for deep reinforcement learning. In *International conference on machine learning*, pages 1995–2003. PMLR, 2016.

[114] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

[115] Christopher John Cornish Hellaby Watkins. Learning from delayed rewards. 1989.

[116] Daan Wierstra, Alexander Foerster, Jan Peters, and Juergen Schmidhuber. Solving deep memory pomdps with recurrent policy gradients. In *International Conference on Artificial Neural Networks*, pages 697–706. Springer, 2007.

[117] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

[118] Ronald J Williams and Jing Peng. Function optimization using connectionist reinforcement learning algorithms. *Connection Science*, 3(3):241–268, 1991.

[119] A. Zanelli, A. Domahidi, J. Jerez, and M. Morari. Forces nlp: an efficient implementation of interior-point... methods for multistage nonlinear nonconvex programs. *International Journal of Control*, pages 1–17, 2017.

[120] Zhaoxuan Zhu, Shobhit Gupta, Abhishek Gupta, and Marcello Canova. A deep reinforcement learning framework for eco-driving in connected and automated hybrid electric vehicles. *arXiv preprint arXiv:2101.05372*, 2021.

[121] Zhaoxuan Zhu, Yuxing Liu, and Marcello Canova. Energy management of hybrid eletric vehicles via deep q networks. In *2020 American Control Conference (ACC)*. IEEE, 2020.