UC Berkeley
UC Berkeley Electronic Theses and Dissertations

Title
From sim-to-real: learning and deploying autonomous vehicle controllers that improve transportation metrics

Permalink
https://escholarship.org/uc/item/0hg266sp

Author
Vinitsky, Eugene Aaron

Publication Date
2022

Peer reviewed|Thesis/dissertation

eScholarship.org Powered by the California Digital Library
University of California

From sim-to-real: learning and deploying autonomous vehicle controllers that improve transportation metrics

by

Eugene Vinitsky

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Engineering - Mechanical Engineering

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Koushil Sreenath, Chair
Professor Alexandre Bayen
Professor Mark Mueller
Professor Pieter Abbeel

Summer 2022

From sim-to-real: learning and deploying autonomous vehicle controllers that improve transportation metrics

Copyright 2022
by
Eugene Vinitsky

Abstract

From sim-to-real: learning and deploying autonomous vehicle controllers that improve transportation metrics

by

Eugene Vinitsky

Doctor of Philosophy in Engineering - Mechanical Engineering

University of California, Berkeley

Professor Koushil Sreenath, Chair

The recent wide availability of semi-autonomous vehicles with distance and lane keep capabilities have created an exciting opportunity to improve the throughput and energy efficiency of the highway by deploying modified control strategies. However, even at current penetration rates, the optimal mechanism for the design of these decentralized, cooperative strategies is an open problem. In this work, we use Multi-Agent Reinforcement Learning (MARL) to investigate, design, and deploy cooperative autonomous vehicles (CAVs) to achieve these goals and demonstrate a field deployment of an RL-based traffic smoothing controller.

We focus on multi-agent reinforcement learning as a mechanism for handling the complexity and non-linearity of large-scale traffic. We start by constructing a standardized suite of benchmark tasks for evaluating the efficacy of learning algorithms in designing controllers for CAVs; we evaluate these algorithms in the centralized setting where all CAVs are actuated by a single controller. We then extend one of these benchmarks, regulation of the inflow to a bottleneck via decentralized CAVs, to the multi-agent setting. We demonstrate that from both low to high penetration rates, CAVs are capable of improving the throughput of a scaled model of the San Francisco-Oakland Bay Bridge and investigate challenges in scaling our methods in open-network settings where vehicles can enter and exit the system.

In preparation for a road test intended to demonstrate stop-and-go wave smoothing on large scale networks, we next study energy optimization of a full-scale model of a section of the I-210 in Los Angeles. Using Proximal Policy Optimization with an augmented value function we demonstrate that we are able to sharply improve the miles-per-gallon of the system and that the resultant controller is robust to likely variations of the system such as system speed and CAV penetration rate. However, we observe that the resultant waves are very unrealistic and additional calibration using higher resolution data is needed.

With the goal of designing a more calibrated simulator, we pursue two approaches: one approach focuses on designing new driver models using available data-sets from Waymo and another approach focused on the use of collected data from the field deployment site. In the first approach, we design a new simulator that 1) efficiently represents the partially observable view-cone of human drivers and investigate whether learning safe driving policies in the simulator yields human-like behavior 2) serves as a challenging MARL benchmark. We observe promising signs of human-similarity from agents trained in the simulator. In the more direct approach, we collect data from the deployment site and use it to design a new, simplified simulator capable of using the collected data while maintaining a high simulation speed. We design energy-improving CAVs in this simulator and demonstrate that these CAVs can be successfully and safely used in a field deployment test.

Contents

List of Figures

List of Tables

Acknowledgments

A PhD has one author but is built on the support of dozens of people. Chance meetings, late night discussions, pair programming, have set the direction of my research as much as any intentional goal setting. In particular, there are a few people who have been immensely influential on my work and thinking in the past few years. First and foremost, my advisor, Alexandre Bayen, who took a chance on me when I transferred to Berkeley with no background in controls, machine learning, optimization, transportation, or much of anything that eventually became my PhD topic (though, in retrospect, I think he probably was not fully aware of this. I suppose he is now). I chose to join his group when one after another of his graduate students said something along the lines of "Alex will support you no matter what it turns out your goals are". At the time, I could not conceive of how right they were. Alex went above-and-beyond in supporting my research vision and career in ways that I would not expect any person to. I still cannot believe some of the emails he sent vouching for me early in my career. He sets the bar for how well an advisor can treat their students. Additional thanks are due to my committee for supporting and assisting with this work!

Then, credit must be due to my conspirators in the Mobile Sensing Lab. Cathy Wu served as a mentor for me for the two years we overlapped, introducing me, at the time totally new to the field, to interesting problems and challenges in transportation, autonomy, and control. Beyond that, seeing how she structured the Flow team, the ways in which she consistently created a welcoming and non-judgemental environment have shaped the ways I think about research and collaboration ever since. Much of what I have learned about being a good collaborator comes from her. Abdul Rahman Kriedieh and Kanaad Parvate were close partners in almost all the work I did in the first three years of my PhD and without them I would be much worse at programming and have gotten a good deal less done (possibly even nothing would have gotten done). Equal affection and credit is due to the rest of the early Flow team: Kathy Jang, Nishant Kheterpal, Yashar Zeynali Farid, and Fangyu Wu. It was an immense pleasure to both do research and spend so much time laughing while doing it.

In the latter half of my PhD, as I got more focused on the multi-agent learning piece of the work (and folks graduated) I got an opportunity to work closely with another group of folks. Natasha Jaques introduced me to the MARL community, has been an excellent collaborator, friend, occasional mentor, and co-organizer of the Multi-Agent Seminar. Michael Dennis has continually astonished me with the depth of his thinking on multi-agent topics and pushed me in turn to raise the quality of my thinking. He's also a relentlessly good faith arguer and can always be counted on to raise the quality of a discussion. The folks of the Multi-Agent Seminar have been a joy to meet with every week and have been a community I love being a part of. Chao Yu, Akash Velu, and Yi Wu were a delight to work with and helped put to rest a persistent source of confusion about the relative rankings of different MARL algorithms for the standard suite of cooperative tasks. While we have yet to publish a paper together (though we certainly will), Minqi Jiang, Jack Parker-Holder, and Ishita Mediratta have been a pleasure to continually brainstorm PAIRED variants with. Adit Singh has been an excellent undergraduate mentee and I look forwards to when he figures out exactly what research field he's going to succeed immensely in. Lastly, this section would be remiss without a long set of thanks to Nathan Lichtlé, who has been my closest collaborator on every paper I have written for the entirety of the past two years. 3/4 of the content of this thesis are papers that Nathan and I have written together; any of my successes are undoubtedly because I got a chance to work with him.

My internships have been a continual source of inspiration for helping my research stay tied to real-world problems. On this front, thanks to Andrej Karpathy and Russell Kaplan for hosting me at Tesla and showing me how AVs actually work (instead of how I had fantasized they worked). Joel Leibo and Raphael Köster were excellent mentors at DeepMind and introduced me to a wide variety of literature on cultural knowledge and learning that I had never heard about; if you happen to glance at the reference section of their papers, you will immediately understand why they are basically encyclopedias of knowledge on these topics. Finally, Brandon Amos was a wonderful host for the year I spent at FAIR; without his and Jakob Foerster's guidance and Xiaomeng Yang's unbelievable programming skills, Nocturne (our simulator for investigated partially observable driving) would likely never have seen the light of day.

I also want to acknowledge an incredible amount support from the open-source community. The SUMO community, in particular Jakob Erdmann's help debugging issues, made my PhD possible. Similarly, open-source RL libraries like RLlab, RLlib, Stable Baselines, Sample Factory, and their eager-to-help authors, in particular Richard Liaw, Eric Liang, and Aleksei Petrenko, prevented me from running into pitfalls that would have easily eaten years of my time. This thesis could not exist without the commitment to open source science and code made by so many folks over the years.

On the non-academic side, I've been blessed with a family that supports me even when they think my life plans are ridiculous (it's such a delicate balance to strike between being supportive and wondering whether I might perhaps be happier getting a job instead of being an academic). I certainly decided that academics could be fun after seeing how obsessed my father was with his field of biochemistry or how late my mother would stay up every day programming at her computer. As for my friends, it has been a joy to make endless, excessively involved jokes with Emmett Goodman and Lan Nguyen (for example, spending an evening in Stanford trying to catch a raccoon or elaborately engineering July 4th parties). Alex Jose has been a wonderful conspirator and I assure you now that my PhD is over we'll finally get around to building some of those dumb robots we often talk about. My roommate of many years, Jonathan Liu, was a steady source of support, piano music / singing, and excellent cooking. Hanna Vinitsky, my wonderful sister, was always available on the phone for absurdly long calls and jokes. Finally, Prastuti Singh, my wife, is the person for whom the concept of *agape* was invented. I could not conceive of a better person to have spent the PhD with, nor of someone funnier, more supportive, or loving. She is relentless in trying to live a life that she is proud of and happy with and I get to tag along. Now that this Ph.D. is over I promise we'll travel everywhere we can.

Finally, this PhD is dedicated to my grandmother and grandfather, Irene and Yuri Goldstein, who passed away during the course of my PhD. If I have successfully learned to be kind and enjoy my life, it was by imitating my grandmother. My grandfather always pushed me to be the best version of myself (sometimes by asking every day if my thesis is done). You are both loved dearly, missed, and never forgotten.

Chapter 1

Introduction

1.1 Possibilities of Cooperative Autonomy in Mixed Autonomy Settings

We are at the precipice of a relatively unheralded transition in our transportation infrastructure. While fully autonomous vehicles (AVs), ride-share programs, and scooters have dominated the headlines, there has been a quieter revolution in the level of control and sensing available on city streets and highways. Of particular relevance to this thesis, level-2 cruise controllers that can safely keep distance and lane have gone from a luxury item to an increasingly standard feature in vehicles. Given standardization of these features, it is not a question of *when* cruise controllers will start to reshape the macro and microscopic characteristics of our highways but *how* they are doing so and how we might use them to achieve more socially desirable outcomes. Indeed, it is already well established both empirically [131] and theoretically [36, 166] that even at low penetration rates such as 5%, which we are likely to hit in the coming decade (and indeed, may already have hit) it is possible for particular cruise control behaviors to have significant impact on metrics such as energy efficiency. We will refer to this low penetration rate regime as *mixed-autonomy traffic*.

What could we do with access to widely-deployed cruise controllers? This has been a long-standing question in the cooperative autonomous vehicles (CAVs) community. CAVs differ from human drivers in a wide number of ways that enable them to achieve outcomes impossible without automation and cooperation. CAVs have lower reaction times and better sensing capabilities of lead vehicle velocities via radar, enabling them to safely drive at close distances to their lead vehicle and sharply increase the potential throughput of the highway [127] or get energy improvements by minimizing wind resistance by platooning [3]. Since they are able to communicate with each other and potentially with sensing infrastructure like loop detectors and cameras, CAVs can avoid safety risks that are unavoidable for human drivers [50]. Using information from surrounding vehicles they can improve the smoothness of merges [35], adopt spacing patterns to minimize stop-and-go traffic, or as we will discuss in Chapter 4, be used to replace non-safety critical traffic light infrastructure.

Given these avenues of improvement and the potential to simply deploy new cruise controllers using existing capabilities and infrastructure, the challenge becomes to how to design the controllers. Even at low penetration rates of 1-10%, once the CAVs are deployed they will constitute tens to thousands of controlled vehicles on any high-throughput roadway. The massive number of vehicles in combination with the need for the vehicles to often cooperate to achieve their goals makes this by necessity a multi-agent problem.

From the perspective of multi-agent controller design, this is an exciting opportunity and an open challenge. Control and optimization of these systems correspond to massive, cooperative, multi-agent problems where controllers that are closer to optimal correspond to immediate improvements in societal welfare. However, as we shall establish throughout this thesis, the problems posed and investigated in this work, due to high agent numbers and their heavily partially observed nature, are difficult for current algorithms. Enabling algorithms to scale to these settings and determining the appropriate algorithms to use is an open challenge posed and partially investigated here.

In this thesis we will attempt to address this challenge, using multi-agent reinforcement learning (MARL) to design new CAVs controllers with the goal of a field deployment of some of the controllers. In particular, *the core goal of this work is to field-deploy RL-based controllers that are capable of smoothing stop-and-go waves at low penetration rate.* In small-scale studies on simple networks the ability of a single CAV has already been empirically established [132, 65]: we aim to demonstrate that this approach can work at highway scale. In this work we pursue this goal by approaching it in increasingly larger pieces, finding ways to get MARL to work on increasingly larger road networks. While we have significant success designing controllers using MARL in small networks, as we begin to scale we begin to be bottlenecked by simulation speed and accuracy. We address this by replacing our simulator by a data-driven approach in which the calibrated simulator is replaced with a data-driven simulator using collected data from the deployment site. We train a controller in this simulator and successfully deploy it and observe promising evidence of wave-reducing behavior.

1.2 Overview

Chapter 2 provides a discussion of the relevant concepts for following subsequent chapters. Following that, we begin in Chapter 3.2, where we provide an overview of reinforcement learning concepts and necessary information on CAVs and tools for simulating CAVs. As the task of controller design for CAVs is significantly distinct from the standard set of robotics tasks that RL is applied to, we then investigate in Chapter 3 how different algorithms fare on the type of tasks where control might be needed in transportation settings such as large Manhattan-like grids of traffic lights, on-ramp merges, and traffic bottlenecks. This work is a first step to establishing some standard benchmark values to compare methods against, allowing us to estimate how our algorithms were performing. Since the methods used in this work are centralized (i.e. all cars are controlled by a single controller), they also allow us to estimate the performance loss in moving from centralized to decentralized methods in later pieces of the work.

Due to limitations of existing libraries at the time, the work covered in Chapter 3 is focused on single-agent control setting. In Chapter 4 we extend one of these benchmarks to the multi-agent setting, showing that on a model of the San Francisco-Oakland Bay Bridge, we are able to improve throughput at even low penetration rates and equal the performance of an installed traffic light at 40% penetration rates. Surprisingly, we find that for this cooperative problem that use the of Centralized Training Decentralized Execution-style algorithms like MADDPG [89], which use centralized components during training but are fully decentralized at evaluation time, do not appear to be necessary and that independent training works well at low penetration rates. We later reproduce this insight across a wide variety of other benchmarks in [161] and show that independent algorithms appear to be sufficient in many fully cooperative settings.

In Chapter 5, having established that these multi-agent methods work in small networks, we begin to prepare for a field deployment on the I-24 in Tennessee in 2021; a site that has a well-established regular source of stop-and-go waves. While we waited for data to be collected on that network, we attempted to design a controller on a different network known for having stop-and-go waves: the I-210 in Los Angeles. Using origin-destination pairs provided by the Connected Corridors PATH project, we built a calibrated model of the I-210 and proceeded to train controllers on it to optimize fuel efficiency using a calibrated energy model for the vehicles. We demonstrated that we could use Proximal Policy Optimization [124] to design controllers for this network and showed a sharp improvement in fuel efficiency for all vehicles at likely penetration rates.

However, the aforementioned study alerted us to several challenges that would occur in attempting to deploy vehicles on the I-24. First, while the network maintained flow-rates consistent with the origin-demand pairs, the type of waves we observed in our simulation were more frequent and significantly distinct from the types of waves observed on real highways. Unfortunately, low frequency resolution data like vehicle counts, observed via loop detectors, are insufficient to calibrate waves. Second, the I-24 network we intended to deploy on was significantly larger than the chunk of the I-210 we had been working with; our I-210 model ran in real-time despite being simulated and took 30 hours to train a controller. Simply transferring our approach to the I-24 and hoping that the resultant controllers would work in a field deployment seemed highly unlikely.

To address these challenges we take two different approaches. First, since the waves are emergent from human driving behavior and our waves are miscalibrated, we focus on the insufficiency of existing human driver models and worked to develop tools and benchmarks for improving them. In Chapter 6, we describe our work on developing a simulator, *Nocturne*, that can be used to build realistic models of human driving by appropriately accounting for human perceptual limitations. Nocturne is built upon real world driving data and contains fast implementations of visibility checking that enable the simulator to efficiently construct visible state without resorting to rendering images. The intention of this simulator is to investigate if human-like behavior such as cooperative merges, aggressive lane changes, etc.

could be emergent from risk-minimization under uncertainty. While we do not resolve the question entirely in the work, we do demonstrate that MARL agents, learning via asynchronous PPO [108] and operating under human-like visibility, have quite high trajectory similarity to the data despite being still quite sub-optimal with respect to the collision rate. More effective agents might be sufficiently close to human-similar to generate accurate waves in a simulator.

On a more direct approach, Chapter 7 proposes the data-driven method that made it into a field deployment in August 2021. In this work we use data collected by a radar-equipped vehicle to construct a simplified single-lane simulator in which an RL car, following behind a leader replaying trajectory data, needs to smooth observed waves. We show that in this simulator we are able to sharply improve the miles-per-gallon (MPG) of both the RL car and the vehicles following behind it, particularly on the low-speed trajectories where the observed waves tend to occur. We test this controller in a field-deployment test and observe that the deployed behavior is quite similar to the simulated behavior, indicating that the sim-to-real gap between our field deployment and the simulation has likely been bridged. While there is insufficient data to determine conclusively that wave-smoothing has occurred, this work serves as a validation of the simulation strategy and a roadmap to a larger scale test intended to occur in Fall 2022.

Finally, in Chapter 8 we speculate on some open question unresolved by this work and exciting directions for the future study and use of CAVs. While this work has investigated some potential ways to design multi-agent controllers at scale, in the final step we find that we are bottlenecked by both simulation and algorithmic speed as well as the challenges of designing sufficiently calibrated simulators. Finding improved ways of overcoming these challenges is an essential topic; we speculate on algorithmic, simulation, and data-centric improvements that might help overcome these core difficulties.

To summarize, Chapter 2 provides an overview of relevant concepts in multi-agent vehicular network simulation and MARL techniques. Chapter 3 discusses our work in designing standard benchmarks for control of these networks using centralized controllers. Chapter 4 builds upon this work to design decentralized controllers for a scaled-up version of one of the benchmarks. Chapter 5 describes initial progress in designing controllers that improve fuel efficiency by eliminating stop-and-go waves in large networks. Chapter 6 digresses slightly and discusses our work on combining data-driven simulators, partial observability, and MARL to design models of human driving that one day may be able to produce calibrated waves as an emergent phenomenon. Finally, Chapter 7 is the capstone of this work, describing our design of a traffic-smoothing controller using RL in a data-driven simulator and the field-deployment of said controller. Chapter 8 wraps up the discussion by pointing out promising areas of future work for the design of CAVs.

Finally, we close with a summary of the contributions of this thesis. In the course of this work we:

- Release the first set of open-source benchmarks for reinforcement learning in mixed autonomy traffic.

- Use multi-agent reinforcement learning to design a fully decentralized, AV based controller for traffic bottleneck optimization.
- Show that multi-agent reinforcement learning can be used at scale to optimize the energy efficiency of a model of the I-210 in Los Angeles.
- Design, release, and test a data-driven simulator for partially observable control problems in driving scenarios.
- Show that the combination of a small amount of data and RL-designed controllers could be used to drive a platoon of traffic-wave smoothing cars in a field deployment test.

Chapter 2

Background

Here I will provide a brief overview of some of the core concepts needed and used in this thesis. In particular, I'll provide a brief overview of microsimulators, describe the role of Flow [154] (the library we built to interface between microsimulators, RL libraries, and the cloud), car-following models used to model human behavior in the simulators, and multi-agent reinforcement learning (MARL). Throughout this work we will frequently use the following terms:

- *ego vehicle*: the reference vehicle that is being controlled.
- *lead vehicle*: the vehicle in front of the ego vehicle.
- *headway* or *space gap*: the distance between the ego and the lead vehicle.

2.1 Traffic micro-simulators and FLOW

To understand how a small set of autonomous vehicles could influence human behavioral patterns at the trajectory level, it is necessary to be able to simulate human driving behavior at a fine granularity. For this, microscopic simulators (micro-simulators), which step the trajectories of individual vehicles at a fairly low time-step (on the order of 0.01 - 1.0 seconds) are frequently used. These can be contrasted with macro-simulators which are used to model flows, routing behavior, and route choice. Micro-simulators generally require the specification of car-following models that describe how vehicles follow their lead vehicle, lane-change, merge, and an imposition of an origin and destination for the vehicles (though this latter component can be determined dynamically). While there are several ubiquitous micro-simulation frameworks such as SUMO [88], AIMSUN [11], and VISSIM [45], in this work we exclusively use SUMO as it is open source, free, and has a pythonic interface, TRACI, that can be used to control the C++ code through a TCP connection.

Atop these micro-simulators we built *FLOW*[154], a python library that connects cloud computing and RL libraries (RLlib [85] and RLlab [39]) to the micro-simulators SUMO and

AIMSUN. Flow is intended to make it straightforward to construct new environments for control of a system either by controlling a number of AVs or controlling some traffic lights. At its core, an environment in flow is defined by the following features:

- A network. This defines the road architecture i.e. the connectivity between roads, the types of merges that are allowed on particular roadways, etc.
- An initial distribution of vehicles and flows. Flow supports the imposition of flows of vehicles per hour at the entrances of the network.
- A specification of which agents are controlled.
- A reward function to be optimized.
- A specification of how to apply actions to vehicles or traffic lights.
- A specification of what features of the environment are visible to controlled agents.

Given these specifications, FLOW constructs an environment that is compatible with the available RL libraries.

FLOW comes with a pre-built set of extensible environments that represent expected building blocks of standard large networks: a traffic bottleneck, a grid of traffic lights, an on-ramp merge, and a roundabout. These networks are described in more detail in Chapter 3. These environments can be pieced together like lego-blocks to build larger environments and have been used as multi-agent benchmarks in a variety of works such as [136, 130, 160].

2.2 Microscopic Car Following Models

Our micro-simulations require a model of human driving used to update the position and lanes of the vehicles. We will describe here models of *acceleration* i.e. following of the lead car. There are also models for discrete decision making and lane changing, both of which we will not describe here as both types of models are infrequently used in this work (a substantive description of common lane-changing models can be found in [42]). In particular, there are two acceleration models used in this work, the Intelligent Driver Model (IDM) [147] and the Krauss Model [75]. For the majority of the work described we default to the intelligent driver model though the Krauss Model is used in Chapter 4. These models are primarily used as their merging behavior in SUMO is sensible[1] and the IDM model can be made *string unstable* i.e. capable of producing waves. For a good reference on other models and the historical development of car-following models, see [21, 75].

[1]There is no citation for this as this has been conveyed informally

Intelligent Driver Model

The IDM model was first introduced in [144] as a low-parameter model capable of reproducing different types of congestion observed in German freeways by simply assuming inhomogeneities in the model parameters. The basic IDM equation for the acceleration is as follow:

$$a_{\text{idm}} = a\left[1 - \left(\frac{v}{v_0}\right)^{\delta} - \left(\frac{s^*(v,\Delta v)}{h}\right)\right]$$
$$s^*(v,\Delta v) = s_0 + vT + \frac{v\Delta v}{2\sqrt{ab}}$$

where v is the ego speed, Δv is the speed difference between the leader and the ego vehicle, and s is the headway. The remaining parameters are hyperparameters whose interpretation we will describe following [144]. If there is no lead vehicle or s is large, the second term velocity will approach the desired free-flow speed $v_0 = 30\frac{\text{m}}{\text{s}}$ with a maximum acceleration of $a = 0.73\frac{\text{m}}{\text{s}^2}$ when $v = 0$. The speed at which the acceleration falls off is controlled by the exponential parameter $\delta = 4$.

At equilibrium (i.e. $a_{\text{idm}} = 0$, $\Delta v = 0$) we can solve for the equilibrium headway $s_e(v)$ and get $s_e(v) = (s_0 + vT)\left[1 - \left(\frac{v}{v_0}{}^{\delta}\right)\right]^{-\frac{1}{2}}$ and at low speeds this reduces to $s_e(v) = s_0 + vT$ so the equilibrium spacing in congestion is a small distance $s_0 = 2\text{m}$ plus a distance corresponding to a safe time-headway of $T = 1.8\text{s}$. For large approach velocities we get $\frac{v\Delta v}{2\sqrt{ab}} >> s_0 + vT$ and we can neglect those small terms to see that the acceleration becomes $\dot{v} = \frac{(v\Delta v)^2}{4bs^2}$ where $b = 1.67\frac{\text{m}}{\text{s}^2}$ can be interpreted as a gain on the magnitude of the braking response. For the parameters of the model such as v_0 we have used standard values from the original paper but other values have been found and used in work explicitly fitting the model to human driving data [67].

To break symmetries and add stochasticity, we use two components. First, SUMO enables the setting of a Normal distribution over the desired free flow speed and others. Additionally, we will generally perturb the dynamics of the IDM vehicles with Gaussian noise. We scale the Gaussian noise by the square root of the time-step Δt according to the Euler-Maruyama method to get a noisy version of the IDM update. This version is used in all works that use the IDM model though with varying noise levels from chapter to chapter:

$$a_{\text{noise}}(v,\Delta v,s) = a_{\text{IDM}}(v,\Delta v,s) + \sqrt{\Delta t}\,\mathcal{N}(0,\sigma)$$

A key feature of the IDM model that is pertinent to this work is that it has regions of parameter space that are conducive to the formation of the stop-and-go waves i.e. it can be made string unstable using relatively simply conditions described in [36]. Finally, note that the IDM model has mathematical problems, namely, there are a set of circumstances under which its velocity can become unboundedly negative in finite time [4]. These problems are often addressed heuristically by simply clipping the velocity at zero; the aforementioned article [4] offers principled variants of the IDM model that do not require clipping.

Integration and Fail-safes

In this section we briefly describe the particular reasons why most micro-simulators employ a first-order update for integration the car-following dynamics, discuss the widely used ballistic method, and describe the fail-safes needed to ensure micro-simulators are collision free.

The car following models described are in continuous time and must be approximated by an integration scheme to be used in a microsimulator. While higher-order schemes such as forth-order Runge-Kutta are frequently used in approximating the evolution of differential equations in other fields of science, it is common in the microsimulation literature to use either a first-order Euler step or a first order "ballistic" method (by first-order we mean that the global error of the discretization is proportional to the step-size). This can be surprising as first-order methods can incur high error at large-discretization steps. The particular use of first-order methods in micro-simulation of driving systems stems from the discrete nature of many traffic phenomena such as stops, lane-changes, or traffic lights as well as the discontinuity of many car-following models (for example, IDM is not smooth at speed $v = 0$). As a consequence, the first order methods can sometimes be as accurate while requiring fewer evaluations of the acceleration [145], an important boon since computing the acceleration can be an expensive component of the simulation time for some RL methods discussed in this thesis.

Since the ballistic method is non-standard in other communities, we reproduce it below:

$$\begin{aligned} x_{t+1} &= x_t + \Delta v_t + \frac{1}{2}\Delta t^2 a_t \\ v_{t+1} &= v_t + \Delta a_t \end{aligned}$$

where x_t, v_t, a_t are the position, velocity, and acceleration of the vehicle at time t and Δ is the discretization step. This essentially corresponds to an Euler-step for the speeds and a trapezoidal update for the position.

Unfortunately, at the simulation steps occasionally used in this work, the models are not necessarily guaranteed to be collision-free. For this reason, they are often supplemented with a fail-safe that clips the acceleration a vehicle takes based on presumed maximum deceleration of its leader vehicle. In short, we compute the maximal velocity such that if the lead vehicle starts braking with its maximum deceleration, the vehicle is still able to safely brake despite the fact that it will start braking one time-step later. When updating the speed we take the minimum between the new speed and the safe speed. Note that the calculation of the brake distance in the safe speed must use the discrete brake distance rather than the continuous brake distance $\frac{v^2}{2b}$ where b is the maximum deceleration of the vehicle under consideration.

2.3 Multi-Agent Reinforcement Learning

This thesis makes extensive use of multi-agent reinforcement learning as the tool for controller design. In this section we describe a few of the problem formulations that are relevant to this work as well as a brief overview of the algorithms used.

We will generally model our problem as either a *Partially Observable Markov Decision Process* (POMDP) [7] or *Decentralized Partially Observable Markov Decision Processes* (Dec-POMDP) [18] depending on whether the problem is multi-agent, partially observed, and cooperative (DEC-POMDP), or single-agent and partially observed (POMDP).

Dec-POMDPs and POMDPs

We define a Dec-POMDP via the tuple

$$\mathcal{S} \times (\mathcal{A}_0, \mathcal{O}_0, \gamma_0, T_0) \times \cdots \times (\mathcal{A}_n, \mathcal{O}_n, \gamma_n, T_n) \times \rho \times P \times \mathcal{R} \times \mathcal{Z}$$

where $\mathcal{S}$ is a set of world-states, $\mathcal{A}_i$ is a set of actions for agent i, $\mathcal{Z} : \mathcal{S} \times (\mathcal{A}_0 \times \cdots \times \mathcal{A}_n) \to (\mathcal{O}_0, \ldots, \mathcal{O}_n)$ describes how the world state is mapped into distributions over observations of the agents, $P : \mathcal{S} \times (\mathcal{A}_0 \times \cdots \times \mathcal{A}_n) \to \mathbb{R}_{\geq 0}$ is the transition probability distribution for moving from one set of agent states s to the next set of states s' given the set of actions $(a_0, \ldots, a_n)$, $\mathcal{R} : \mathcal{S} \times (\mathcal{A}_0 \times \cdots \times \mathcal{A}_n) \to \mathbb{R}$ is the reward function, $\rho : \mathcal{S} \to \mathbb{R}_{\geq 0}$ is the initial state distribution, $\gamma_i \in (0, 1]$ is the discount factor for agent i, and T_i is the horizon for agent i which in some settings may be functions of the state (i.e. if the agents can 'exit' the system or 'die'). Note that there is one shared reward for all agents in this setting. The POMDP setting is the same but with a single agents.

Reinforcement Learning

We provide a brief overview of the RL formalism; we use a single-agent MDP setting to simplify the notation. Generically, RL studies the problem of how an agent can learn to take actions in its environment to maximize its cumulative discounted reward. Specifically it tries to find a controller π^* that maximizes

$$J^{\pi^*} = \underset{\pi}{\operatorname{argmax}}\, \mathbb{E}_{\rho_0,\ p(s_{t+1}|s_t, a_t)} \left[\sum_{t=0}^{T} \gamma^t r_t \mid \pi(a_t|s_t) \right]$$

where r_t is the reward at time t and the expectation is over the start state distribution, the probabilistic dynamics, and the probabilistic controller π. While there are other formulations of the quantity to be maximized such as average-reward or distributional RL, colloquially reinforcement learning refers to the max-reward setting described. Note that we have temporarily dropped the dependence on agent index for clarity. The goal in RL is to use the observed data from the MDP to compute the controller $\pi : \mathcal{S} \to \mathcal{A}$, mapping states to actions, that maximizes J^{π}.

It is increasingly ubiquitous to parametrize the controller as a neural network (though not necessarily needed, see [114] for an argument that linear controllers are sufficient in many cases). Throughout this work we will denote the parameters of this controller, in this case the neural network weights, by θ and the controller by π_θ. A neural net consists of a stacked set of affine linear transforms and non-linearities that the input is alternately passed through. The presence of multiple stacked layers is the origin of the term “deep” reinforcement learning. In this work in all multi-agent settings we will use a shared neural network amongst all agents; in multi-agent settings each agent will act independently but use a copy of the exact same controller.

Chapter 3

Benchmarks for reinforcement learning in mixed-autonomy traffic

3.1 Introduction

In recent years, the success of Deep RL has been closely tied to benchmarks capable of evaluating progress in algorithms and methods. The *Arcade Learning Environment* (ALE) [14] has become a popular benchmark for evaluating algorithms designed for tasks with high-dimensional state inputs and discrete actions. The benchmarks released with rllab [39] contain 31 continuous control tasks, which range from simple tasks, such as cart-pole balancing, to challenging tasks such as high-DOF locomotion, tasks with partial observations, and hierarchically structured tasks. However, in *mixed-autonomy traffic control*, the study of human drivers interacting with CAVs, the lack of a standardized and challenging testbed for RL that is grounded in a real-world problem domain makes it difficult to quantify the practicality of the proposed methods in the literature. Systematic evaluation and comparison will not only further understanding of the strengths of existing algorithms, but also reveal their limitations and suggest directions for future research.

We attempt to address this problem and present a benchmark consisting of four traffic control tasks based in common, real-world traffic scenarios. Methods which solve these traffic tasks can greatly impact traffic control in cities, the integration of automated vehicles, and urban planning. We characterize a set of RL algorithms by their effectiveness in training deep neural network policies.

In the traffic analysis and control community, the number of standardized benchmarks is limited. For microscopic data on human driving behavior there is the NGSIM dataset [149], the 100-car naturalistic driving study [37], and Mobile Millenium [57]. For traffic control, there are a few recurring problems that are considered, including deriving controllers to minimize propagation of shockwaves in traffic [135, 64], AV assisted merging [41, 109] and energy minimization via platooning [16].

The work presented here constitutes the first set of standard set of benchmarks for traffic control in a micro-simulator nor a framework in which deep-RL can be applied to the control task. The coming future likely will involve a mixture of AVs interacting with human drivers and infrastructure, but there isn't yet a framework for simultaneously learning control for these interacting pieces. To this end, the present chapter presents four benchmark problems representing four key traffic control problems: shockwave minimization, inflow management, efficient merging, and intersection control. These tasks expose various RL-oriented problems, including exploration, partial observability, scaling in the state space, and scaling in the action space. Additionally, each problem exposes some questions in RL that may primarily arise in the context of transportation systems. By working on a standard set of benchmarks, we hope to make it possible for the mixed-autonomy and RL community to effectively compare their results, test new algorithms, and develop new techniques that are able to take advantage of the underlying structures inherent in the state and action spaces of transportation problems.

The key contributions of this article are:

- The description of a new set of benchmarks for mixed-autonomy traffic that expose several key aspects of traffic control
- A characterization of the RL problems involved in each benchmark as well as an initial description of the performance of a few gradient based and gradient free deep-RL algorithms
- A summary of the key traffic improvements resulting from the application of deep-RL algorithms

The article is organized as follows. In Sec. 3.4 we provide details on the released benchmarks. In Sec. 3.5 we detail the results of the application of three deep-RL algorithms to learning neural network-based policies and another deep-RL algorithm to learn purely linear policies. Finally, in Sec. 3.6 we discuss the performance of the algorithms as well as open questions and problems that can be tackled with these benchmarks.

3.2 Background

This section presents a brief overview into the reinforcement learning algorithms and traffic models used for the remainder of this article. In addition, sec. 3.3 introduces the computational framework used to construct the benchmarks described in sec. 3.4.

Algorithms used

A broad range of RL algorithms have been designed to compute an optimal policy θ^*. Gradient-based algorithms [125, 124] estimate a gradient for the policy from expected returns achieved during simulation. Gradient-free algorithms [118], on the other hand, treat the return as a black box function to be optimized in terms of the policy parameters. In particular, random search methods [94] training linear policies have recently been demonstrated to be a competitive alternative to gradient-based and gradient-free method in solving commonly used RL benchmarks. The success of linear policies suggests that these benchmarks are not as difficult as they were believed to be.

Mixed-autonomy traffic

The notion of using connected autonomous vehicles (CAVs) to improve traffic flow has an extensive history stemming back to the 1960s. Early works focused on designing controllers to direct a fleet of vehicles to form a dense, stable platoon [126, 19]. Among many others, these works represent the significant progress that has been made in the paradigm of vehicle platooning.

Recently, there has been a new emphasis on traffic control in the context of mixed-autonomy, where only a fraction of vehicles are CAVs and are interacting with human-driven vehicles. To this end, pioneering researchers proposed and tested several hand-designed control laws [70, 151]. Notably, in 2015, an experimental test of a control strategy named "slow-in, fast-out" strategy [102] was conducted by [141], which involved five vehicles in a closed course. A larger scale experiment was later conducted in 2016, where two other control methods were tested extensively with 21 humans drivers and one CAV in a circular track [134, 156]. It was suggested from the previous works that traffic throughput can be improved with a small percentage of autonomous vehicles.

Simulating mixed-autonomy performance

This article simulates the performance of mixed-autonomy traffic control policies through the use of microscopic traffic simulators, in which the states, actions, and transition properties of a transportation network are modeled at the level of individual vehicles. Traffic microsimulators have been broadly accepted in the transportation engineering community as an acceptable prelude to the empirical evaluation of several complex automated tasks such as cooperative adaptive cruise control (CACC) [95] and mixed-autonomy traffic flow control [135]. This has spurred interest in the development of several frameworks dedicated to accurate microscopic traffic reconstruction, with prominent simulators including propriety software such as Aimsun [25] and VISSIM [45], and open-source libraries such as SUMO [74]. Traffic networks in this article are modeled using the latter.

In the context of traffic microsimulations, human-driver models are used to recreate the lateral [113] and longitudinal [21] behavior of human-driven vehicles. For longitudinal, or acceleration, dynamics, human-driven vehicles in all experiments presented in this article use the *Intelligent Driver Model* [147], a state-of-the-art car-following model. Other behaviors, such as lane changing, merging, right-of-way at intersections, and traffic light compliance, are dictated by the simulator [74].

3.3 *FLOW*: facilitating benchmark generation

The mixed-autonomy traffic benchmarks presented in this article are designed and developed in *FLOW*, a Python library that interfaces the RL libraries RLlib [83] and rllab [39] with SUMO [74], a microscopic simulator for traffic and vehicle dynamics. *Flow* enables the systematic creation of a variety of traffic-oriented RL tasks for the purpose of generating control strategies for autonomous vehicles, traffic lights, etc. These environments are compatible with OpenAI Gym [22] in order to promote integration with the majority of training algorithms currently being developed by the RL community. For details on the architecture and on training autonomous vehicles to maximize system-level velocity, we refer the readers to [155].

3.4 Benchmarks

In this section, we detail the network structure, controllable parameters, provided state spaces, action spaces, and rewards of the benchmarks. The states and rewards associated with each benchmark are normalized in accordance with common RL practices as well as to minimize the amount of necessary hyperparameter tuning from benchmark to benchmark. We stress that most aspects of the benchmark (network structure, reward, observation space, etc.) can be easily modified if researchers are interested in doing so.

Figure eight: optimizing intersection capacity

The figure eight network (Fig 3.1a), previously presented in [152], acts as a closed representation of an intersection. In a figure eight network containing a total of 14 vehicles, we witness the formation of queues resulting from vehicles arriving simultaneously at the intersection and slowing down to obey right-of-way rules. This behavior significantly reduces the average speed of vehicles in the network.

In a mixed-autonomy setting, a portion of vehicles are treated as CAVs with the objective of regulating the flow of vehicles through the intersection in order to improve system-level velocities. The components of the MDP for this benchmark are defined as follows:

- States: The state consists of a vector of velocities and positions for each vehicle in the network, ordered by the position of each vehicle, $s := (v_i, x_i)_{i=0:k-1} \in \mathbb{R}^{2k}$, where k is the number of vehicles in the network. Note that the position is defined relative to a pre-specified starting point.
- Actions: The actions are a list of accelerations for each CAV, $a \in \mathbb{R}^n_{[a_{\min}, a_{\max}]}$, where n is the number of CAVs, and $a_{\min}$ and $a_{\max}$ are the minimum and maximum accelerations, respectively.

- Reward: The objective of the learning agent is to achieve high speeds while penalizing collisions. Accordingly, the reward function is defined as follows:

$$r := \max\Big(||v_{\text{des}} \cdot \mathbb{1}^k||_2 - ||v_{\text{des}} - v||_2,\ 0 \Big)\ /\ ||v_{\text{des}} \cdot \mathbb{1}^k||_2 \tag{3.1}$$

  where v_{des} is an arbitrary large velocity used to encourage high speeds and $v \in \mathbb{R}^k$ is the velocities of all vehicles in the network.

This task was previously been studied in the context of mixed-autonomy traffic [152]. Replacing one vehicle in the network with an autonomous vehicle (AV), the article reveals that the AV can (platooning behavior). A similar behavior is achieved when all vehicles are replaced with a set of connected and autonomous vehicles (CAVs), with one CAV acting as a leader and the other following it as closely as possible. However, if the reward function is modified to excessively penalize small bumper-to-bumper gaps between consecutive vehicles, the agent learns to space vehicles in such as way that (weaving behavior). This suggests that in the absence of reward shaping, the agent cannot overcome certain local maxima using vanilla gradient-based methods (in this case TRPO).

We explore varying levels of controllability by increasing the portion of CAVs within the network. These tasks are parametrized as follows:

- `figureeight0`: 13 humans, 1 CAV ($\mathcal{S} \in \mathbb{R}^{28}$, $\mathcal{A} \in \mathbb{R}^1$, $T = 1500$).
- `figureeight1`: 7 humans, 7 CAVs ($\mathcal{S} \in \mathbb{R}^{28}$, $\mathcal{A} \in \mathbb{R}^7$, $T = 1500$).
- `figureeight2`: 0 human, 14 CAVs ($\mathcal{S} \in \mathbb{R}^{28}$, $\mathcal{A} \in \mathbb{R}^{14}$, $T = 1500$).

Merge: controlling shockwaves from on-ramp merges

The merge network (see Fig. 3.1c) highlights the effect of disturbances on vehicles in a highway network. This network consists of a single lane highway of inflow rate $F_h = 2000$ veh/hr and a on-merge with inflow rate $F_m = 100$ veh/hr. Specifically, perturbations resulting from vehicles arriving from the on-merge lead to the formation of backwards propagating stop-and-go waves, thereby reducing the throughput of vehicles in the network. This phenomenon is known as convective instability [146].

In a mixed-autonomy setting, a percentage of vehicles in the main highway are tasked with the objective of dissipating the formation and propagation of stop-and-go waves from locally observable information. Moreover, given the open nature of the network, the total number of CAVs within the network may vary at any given time. Taking these into account, we characterize our MDP as follows:

- States: The state consists of the speeds and bumper-to-bumper gaps of the vehicles immediately preceding and following the CAVs, as well as the speed of the CAVs, i.e. $s := (v_{i,\text{lead}}, v_{i,\text{lag}}, h_{i,\text{lag}}, h_{i,\text{lag}}, v_i) \in \mathbb{R}^{n_{RL}}$. In order to account for variability in the number of CAVs (n_{CAV}), a constant n_{RL} term is defined. When $n_{CAV} > n_{RL}$, information from the extra CAVs are not included in the state. Moreover, if $n_{CAV} < n_{RL}$ the state is padded with zeros.

- Actions: The actions consist of a list of bounded accelerations for each CAV, i.e. $a \in \mathbb{R}^{n_{RL}}_{[a_{\min}, a_{\max}]}$ One again, an n_{RL} term is used to handle variable numbers of CAVs. If $n_{CAV} > n_{RL}$ the extra CAVs are treated as human-driven vehicles and their states are updated using human driver models. Moreover, if $n_{CAV} < n_{RL}$, the extra actions are ignored.
- Reward: The objective in this problem is, once again, improving mobility, either via the speed of vehicles in the network or by maximizing the number of vehicles that pass through the network. Accordingly, we use an augmented version of the reward function presented in sec. 3.4.

$$r := \max\Big(||v_{\text{des}} \cdot \mathbb{1}^k||_2 - ||v_{\text{des}} - v||_2,\ 0 \Big) \ / \ ||v_{\text{des}} \cdot \mathbb{1}^k||_2 - \alpha \sum_{i \in CAV} \max\big[h_{\max} - h_i(t), 0\big] \quad (3.2)$$

  The added term penalizes small headways among the CAVs; it is minimal when all CAVs are spaced at $h_{\max}$. This discourages dense states that lead to the formation of stop-and-go traffic.

We explore three levels of difficulty in the problem by increasing the percentage of vehicles that are autonomous, thereby resulting in larger state/action spaces.

- `merge0`: 10% CAV penetration rate ($S \in \mathbb{R}^{25}$, $A \in \mathbb{R}^5$, $T = 750$).
- `merge1`: 25% CAV penetration rate ($S \in \mathbb{R}^{65}$, $A \in \mathbb{R}^{13}$, $T = 750$).
- `merge2`: 33.3% CAV penetration rate ($S \in \mathbb{R}^{85}$, $A \in \mathbb{R}^{17}$, $T = 750$).

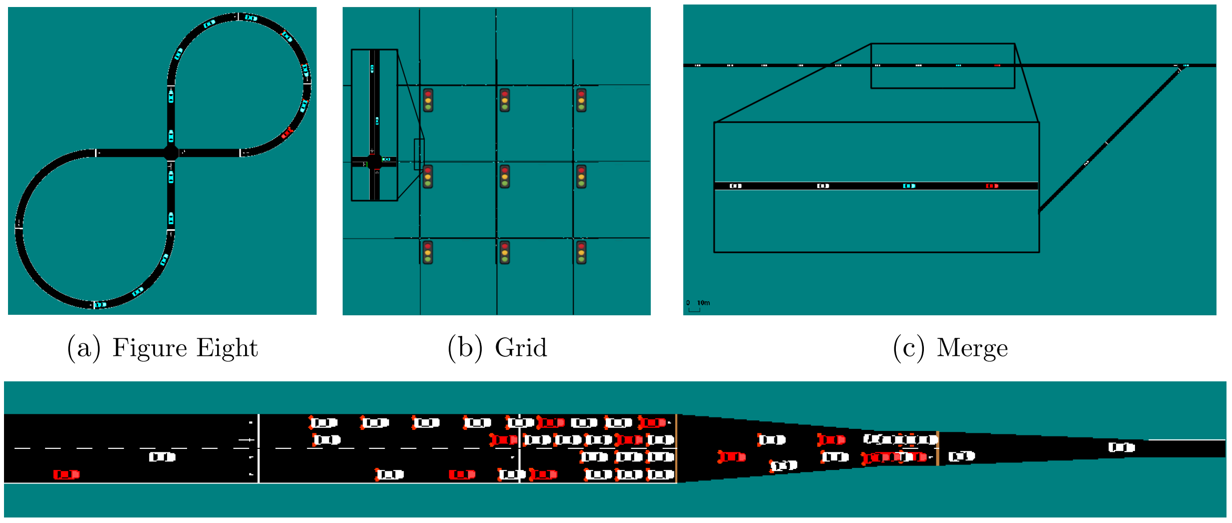
Grid: improving traffic signal timing schedules

The grid (see Fig. 3.1(b)) is an idealized representation of a city with a grid-like structure such as Manhattan. The purpose of this problem is to highlight issues that arise in coordination of traffic light control, particularly questions of partial observability and the scaling of RL algorithms with action dimension. Solutions to this problem will generate new traffic light control schemes that minimize the average per-vehicle delay while inducing some measure of fairness.

Vehicles enter at the corners of the grid. For simplicity of the problem, vehicles travel straight on their path. Each intersection has a traffic light that allows vehicles to flow either horizontally or vertically. If the light is green, it transitions to yellow for two seconds before switching to red for the purpose of safety.

The gym environment has the following state space, action space, and reward:

- States: Speed, distance to intersection, and edge number of each vehicle. The edges of the grid are uniquely numbered so the travel direction can be inferred. For the traffic lights we return 0,1 corresponding to green or red for each light, a number between [0, t_{switch}] indicating how long until a switch can occur, and 0,1 indicating if the light is currently yellow. Finally, we return the average density and velocity of each edge.


(a) Figure Eight (b) Grid (c) Merge

(d) Bottleneck

Figure 3.1: Network configurations for the various benchmarks presented in this article. In each of the figures, vehicles in red are autonomous (controlled by the RL agent), in blue are human-driven and directly observed in the state space, and in white are human-driven an unobserved.

- Actions: A list of numbers $a = [-1, 1]^n$ where n is the number of traffic lights. If $a_i > 0$ for traffic light i it switches, otherwise no action is taken.
- Reward: The reward present in Eq. 3.4 is once again used for this benchmark. Here the use of the 2-norm induces fairness, as a more equal distribution of speeds produces a larger reward.

The versions of the benchmark are as follows:

- `grid0`: 3x3 grid (9 traffic lights), inflow = 300 veh/hour/lane ($\mathbb{S} \in \mathbb{R}^{339}, \mathbb{A} \in \mathbb{R}^{9}$, $T = 400$)
- `grid1`: 5x5 grid (25 traffic lights), inflow = 300 veh/hour/lane ($\mathbb{S} \in \mathbb{R}^{915}, \mathbb{A} \in \mathbb{R}^{25}$, $T = 400$)

Bottleneck: maximizing throughput in a bottleneck structure

The bottleneck is an idealized version of the Oakland-San Francisco Bay Bridge. On the Bay Bridge, 16 non-HOV lanes narrow down to eight and subsequently to five. In our model, the lanes reduce from $4N$ to $2N$ to N, where N is a scaling factor. This system exhibits the phenomenon known as *capacity drop* [117], where the throughput, defined as number of vehicles passing through the system per hour, experiences a sharp drop in magnitude after the inflow increases past a critical value. While there are multiple proposed reasons underlying the capacity drop, ranging from hysteresis in traffic flow [117] to velocity variance [148], this

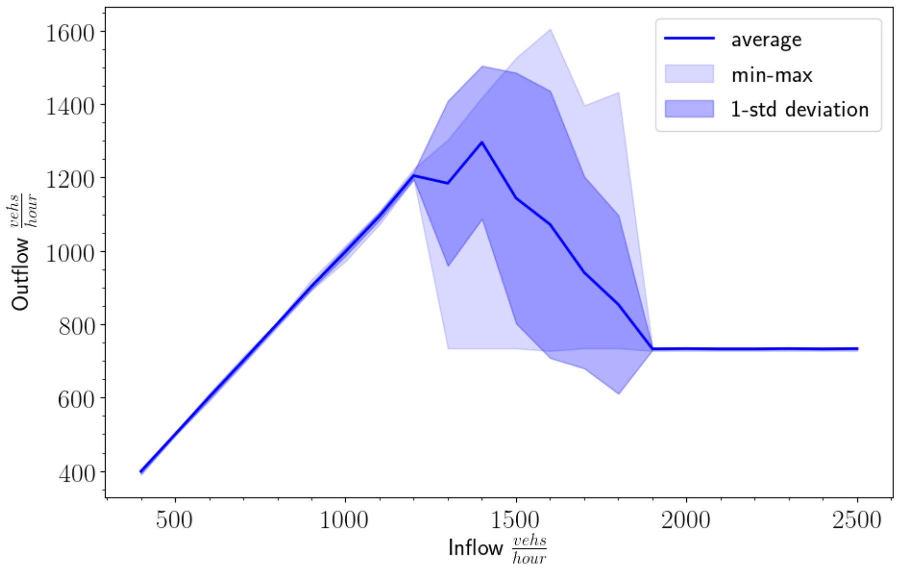


Figure 3.2: Inflow-outflow relation of the bottleneck. At an inflow of approximately 1500 veh/hour, the outflow begins to dip. Past the drop the minimum represents the stable value the system would decay to if run for sufficiently long.

phenomenon has been empirically observed and contributes to the inefficiency of highway traffic (see fig. 3.2).

Given this inflow-outflow relation, the goal of this problem is to learn to avoid the capacity drop and maximize the total outflow in a mixed-autonomy setting. In what follows, we term each distinct segment of road an *edge*. For each edge, users can specify for each edge whether it is observed and/or controlled, and how many segments it is divided into: we refer to this as an *edge-segment*. Although the observation space, action space, and reward can easily be modified, the provided environment operates on the following MDP:

- States: The mean positions and velocities of human drivers for each lane for each edge segment. The mean positions and velocities of the CAVs on each segment. The outflow of the system in vehicles per/hour over the last 5 seconds.
- Actions: For a given edge-segment and a given lane, the RL action shifts the maximum speed of all the CAVs in the segment from their current value. By shifting the max-speed to higher or lower values, the system indirectly controls the velocity of the RL vehicles.
- Reward: $r_t = \sum_{i=t-\frac{5}{\Delta t}}^{i=t} \frac{\frac{n_{\text{exit}}(i)}{5}}{\Delta t * n_{\text{lanes}} * 500}$ where $n_{\text{exit}}(i)$ is the number of vehicles that exited the system at time-step i. Colloquially, this is the outflow over the last 5 seconds normalized by the number of lanes, n_{lanes} and a factor of 500. This is to keep the scale of the reward in line with the other benchmarks.

The three benchmark problems are as follows:

- `bottleneck0`: $N = 1$, inflow = 1900 veh/hour, 10% CAV penetration. No vehicles are allowed to lane change. ($\mathcal{S} \in \mathbb{R}^{141}, \mathcal{A} \in \mathbb{R}^{20}$, $T = 1000$)
- `bottleneck1`: $N = 1$, inflow = 1900 veh/hour, 10% CAV penetration. The human drivers follow the standard lane changing model in the simulator. ($\mathcal{S} \in \mathbb{R}^{141}, \mathcal{A} \in \mathbb{R}^{20}$, $T = 1000$)
- `bottleneck2`: $N = 3$, inflow = 3800 veh/hour, 10% CAV penetration. No vehicles are allowed to lane change. ($\mathcal{S} \in \mathbb{R}^{281}, \mathcal{A} \in \mathbb{R}^{40}$, $T = 1000$)

3.5 Experiments

In this section, we present preliminary results from running four reinforcement learning algorithms on the benchmarks presented in sec. 3.4.

Candidate controllers

Experiments were conducted using gradient-based algorithms *Trust Region Policy Optimization* (TRPO) [125] and *Proximal Policy Optimization* (PPO) [124] as well as the gradient-free method *Evolutionary Strategies* (ES) [118] and an implementation of *Augmented Random Search* (ARS) [94]. For ARS a linear policy is used. For ES we use a deterministic MLP with hidden layers $(100, 50, 25)$ and tanh non-linearity whereas for PPO and TRPO the MLP is diagonal Gaussian. Moreover, in the PPO algorithm, a [256, 256] MLP with tanh non-linearity is used to compute a value function baseline, whereas a linear feature baseline is used for the TRPO algorithm. Other explored hyperparameters for each algorithm can be found in Table A.1 in appendix section A.1. Results are reported over three random seeds to account for stochasticity and to test training stability. Outside of the reported hyperparamters, we use the default parameters set in rllab and rllib as per the commits specified in section 3.5.

Reproducibility

In the spirit of reproducibility, the controllers used to generate the following results are stored as .pkl files and tensorflow checkpoints at `s3://public.flow.results/corl_exps/exps_final`. We have frozen the code used to generate the results as Flow 0.3.0 and commit number “bc44b21”. The commit number of SUMO, available at `https://github.com/eclipse/sumo` used to run the results is “1d4338ab80”. The version of RLlib used to run the code is available at `https://github.com/eugenevinitsky/ray` at commit “0f45f80”.

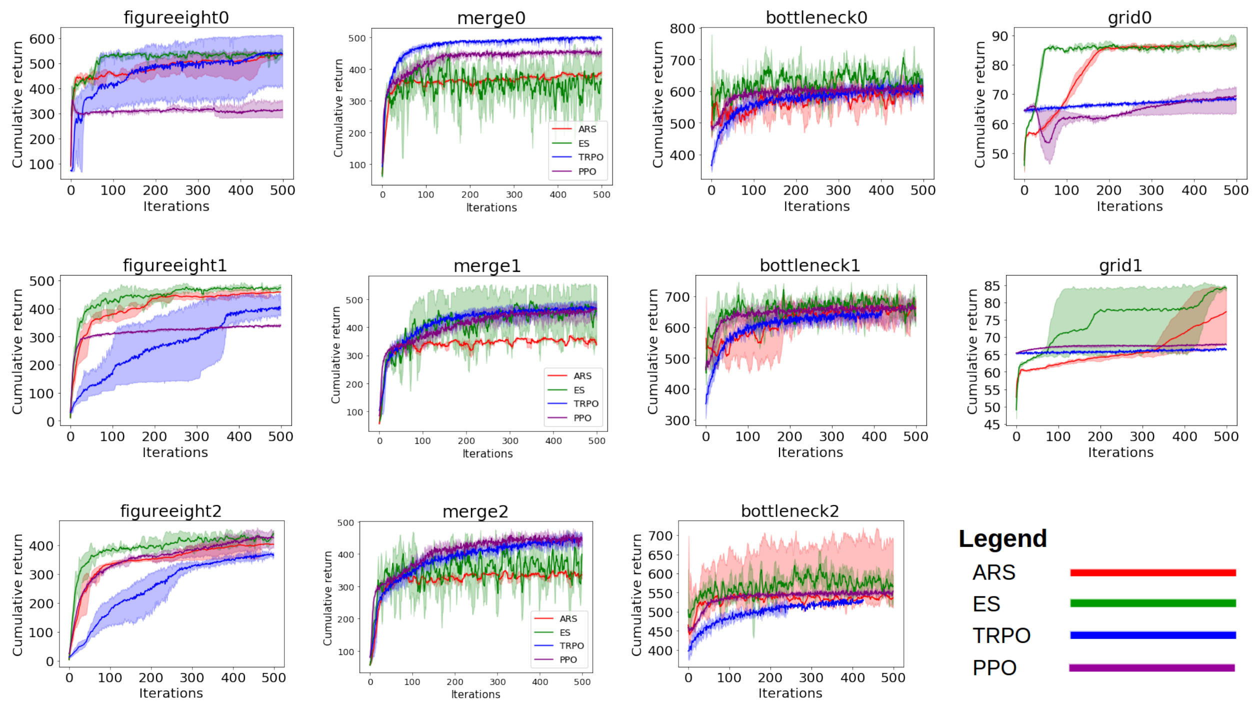


Figure 3.3: Learning curves for each benchmark. A iteration of training in each algorithm consists of 50 rollouts/trajectories. The reported cumulative rewards correspond to undiscounted returns.

Algorithm performance

Fig. 3.3 presents the training performance of each of the algorithms presented in sec. 3.5. The results suggest that gradient-based algorithms (TRPO and PPO) are more effective in merge scenarios, while gradient-free algorithms (ES) are better in figure eight, grid, and bottleneck benchmarks. This is likely due to the merge network reward providing richer feedback with regards to which states are good/bad, thereby assisting gradient-based algorithms in computing meaningful gradients. On the other hand, in environments populated with many local extrema associated with poor rewards, more exploration-oriented algorithms (ARS or ES) are less prone to being trapped in local optima.

Controller performance

We highlight the effectiveness of each learned policy on improving traffic performance. In order to do so in terms of understandable values that are not cluttered by the reward design process, simple scenario-specific performance metrics are defined for each benchmark. These metrics are:

- Figure eight: Average speed of vehicles in the network (m/s).

Table 3.1: Average optimal return and standard deviation based on performance metrics after 500 training iterations; average is over 40 rollouts. "–" indicates that the experiment did not have a complete number of seeds and thus was not reported.

Benchmark	ARS	ES	TRPO	PPO	Human
Figure Eight 0	7.31 ± 0.54	6.87 ± 0.08	8.26 ± 0.10	–	4.18 ± 0.09
Figure Eight 1	6.43 ± 0.01	–	5.61 ± 0.55	–	4.18 ± 0.09
Figure Eight 2	5.70 ± 0.01	5.96 ± 0.00	5.03 ± 0.23	–	4.18 ± 0.09
Merge 0	11.3 ± 0.31	13.31 ± 0.54	14.95 ± 0.12	13.66 ± 0.40	7.39 ± 0.55
Merge 1	11.06 ± 0.32	17.29 ± 0.40	13.74 ± 0.23	14.61 ± 0.47	7.39 ± 0.55
Merge 2	11.5 ± 0.54	17.36 ± 0.48	14.14 ± 0.24	14.54 ± 0.31	7.39 ± 0.55
Grid 0	270.2 ± 0.2	271.7 ± 0.6	296.2 ± 2.5	296.8 ± 5.3	280.8 ± 1.5
Grid 1	274.7 ± 1.0	274.3 ± 1.1	296.0 ± 2.0	296.2 ± 2.0	276.8 ± 1.7
Bottleneck 0	1265 ± 263	1360 ± 200	1298 ± 268	1167 ± 264	1023 ± 263
Bottleneck 1	1350 ± 162	1378 ± 192	1375 ± 61	1258 ± 200	1135 ± 319
Bottleneck 2	2284 ± 231	2324 ± 264	2131 ± 190	2143 ± 208	1889 ± 252

- Merge: Average speed of vehicles in the network (m/s).
- Grid: Average delay of all vehicles in the network (s) relative to traveling at the speed limit. In this case, smaller values are preferable.
- Bottleneck: Outflow over the last 500 seconds of a 1000 second rollout (veh/hr).

In order to get a sense of how well these policies are performing, we use an estimate of human-level performance acquired by running simulations of each environment in the absence of learning agents as a baseline. Note, for the grid we use the actuated traffic light baseline available in SUMO with the default parameters for minimum duration of green and red time; however, we set the yellow light to imitate our controller and switch every two seconds. Table 3.1 highlights the performance of each algorithm within the context of the above mentioned performance metrics. As we can see, the learned policy outperform human-level dynamics in all task.

3.6 Conclusions and Future Work

Given the current performance of the different RL algorithms, there is an abundance of open questions that remain for these benchmarks. For each of these problems, it remains to characterize whether the optimal solution has been achieved. Towards this end, we provide some intuition here that the optimal solution has not been achieved from the perspective of speed/delay of the resultant traffic. For the figure eight, although it is a toy example, CAVs in a fully autonomous setting do not succeed in coordinating themselves in a manner such that they are evenly spaced and move through the intersection at the speed limit (30 m/s) without collision. The likely existence of a more optimal designable control strategy suggests that there is an exploration problem to be solved. For the bottleneck, the capacity diagram in Fig. 3.2 occasionally reaches a maximum of 1550 vehicles per hour, suggesting that it is possible, at least over the observed horizon, to arrange vehicles so that they merge optimally without the sharp decelerations that eventually give rise to the bottleneck. Furthermore, for *bottleneck2* the problem is two copies of bottleneck 0 stacked together; the optimal reward is at least the value of *bottleneck0*, but that value is not achieved. For the merge, the trained policies started to gradually degrade after certain iterations, suggesting that the problem is difficult to solve with existing optimization methods. Finally, for the grid, the considered inflows are 300 vehicles per hour per edge for both small and large grids. While we do not have a proof that we have not discovered an optimal solution for the grid problems, visualizations of the solutions do not show the platooning that is expected of optimal solutions to heavy traffic inflows. This is a heuristic argument and further characterization of the optimum is worth pursuing.

While only a partial list, we highlight several additional key challenges that emerge in studying these problems. The first, apparent in the merge and the bottleneck, are the unfixed action and state space. At any given time, the number of controlled and uncontrolled vehicles vary. In the merge case, we resorted to throwing away extraneous actions when there were too few vehicles and having the excess vehicles imitate a human driven vehicle when there were too many. For the bottleneck, we chose to discretize, passing the same command to every vehicle in a given segment of the bottleneck. Similarly for the state space, we turned to aggregate statistics, simply recording the number of vehicles in a given segment rather than more fine-grained statistics. While both of these solutions appear to partially work, perhaps there is a more elegant solution that can adaptively scale to different sized-state and action spaces. Possible candidates include treating this as a multi-agent problem and just using local information for each agent, taking a giant space space that is mostly sparse when vehicles are missing, or using a neural network with sets of input and output matrices that can be selected from to accommodate the variations in size.

Several of these benchmarks admit problems similar to those experienced in video game environments, namely sparsity/credit assignment of the reward and redundancies and symmetries in the state space. The rewards are averages across the performance of the vehicles; because of the number of vehicles in the system for the larger problems like *bottleneck* and *grid*, any given actions positive or negative effect can be offset by the opposite effect elsewhere in the system. An equally interesting issue is the notion of symmetry; many transportation systems, exemplified here by the *bottleneck* and the *grid*, contain a high degree of symmetry. The symmetry affords a potential for a reduction in the dimension of the state space: exploring how to do this, whether via convolutional neural networks, graph neural networks, or explicit orderings of the state space that remove the symmetries, is an open question. Furthermore, many of the systems are clearly composed of distinct sub-units, suggesting that hierarchical approaches could be viable.

A few more open questions that are not explored in this work include: fairness, decentralization, generalization, and sample-efficiency. From the perspective of fairness, several of our results achieve high scores by inducing negative outcomes on a subset of the vehicles. For example, ARS on the bottleneck simply learn to block a lane. This reduces merge conflicts and increases outflow but has extremely negative effects on some of the vehicles. Our controllers are fully centralized; this is unlikely to be possible in real road networks and the effects of decentralization remain unexplored. As to generalization, an interesting question is whether any of the derived controllers will work well on networks or inflows that were outside of their training set. Finally, many of our experiments take over 10 hours to perform 500 iterations; to scale to real-world transportation networks it will be necessary to identify methods for increasing sample efficiency.

Finally, as evidenced by the large *grid* and *bottleneck* benchmarks on which we either do not learn an optimal policy or see flat training curves for several algorithms, there remain fundamental advances to be made in learning policies that can tackle the large action and state spaces that arise in trying to control city-scale road networks. It is our hope that the release of these benchmarks spur excitement and progress on these problems.

Chapter 4

Optimizing Mixed Autonomy Traffic Flow With Decentralized Autonomous Vehicles and Multi-Agent Reinforcement Learning

4.1 Introduction

In this chapter, we examine the usage of multi-agent RL (MARL) in many-vehicle decentralized control. Using it as a tool, we demonstrate how level-2 AVs, equipped with standard sensors like cameras and radars, can be used to improve the throughput of a simplified model of the San Francisco-Oakland Bay Bridge and similar bottleneck structures. We focus on using AVs to improve the throughput of a lane reduction, a road architecture where the number of lanes suddenly decreases. We will refer to successive lane reductions as a *traffic bottleneck*. Bottlenecks are believed to cause a phenomenon known as *capacity drop* [55, 31] where the inflow-outflow relationship at the bottleneck is initially linear but above some critical inflow value experiences a hysteric transition where the outflow suddenly and sharply drops (see Fig. 4.4 for an example). The imbalance between inflow and outflow leads to congestion and a reduction in the throughput of the bottleneck.

To avoid this reduction, it is necessary to restrict the inflow so that it never exceeds the critical value above which capacity drop occurs. One approach to tackling this is to introduce traffic lights into the network that meter/restrict the inflow [106] but this would require the installation of additional infrastructure. Instead, autonomous vehicles can be used as mobile metering infrastructure, essentially distributed traffic lights, that intelligently select when to meter and when to let the flow continue without restriction.

While there is work characterizing bottleneck control using vehicle-based control, it usually operates in the centralized regime where a single controller outputs commands to all the AVs in the system either via variable speed limits [128, 158, 90, 60] or centrally coordinated platoons [32]. Here we consider the challenging multi-agent problem where each AV operates in a fully decentralized fashion and control is applied at the level of individual vehicle accelerations. The AVs can still coordinate but only implicitly: they can only use common knowledge to decide which AV should go next. While decentralization adds additional difficulty in controller design, the resultant controllers should be realizable using existing cruise control technology and can consequently be implemented without relying on any improvements in vehicle-to-vehicle communication technology.

We investigate the potential impact of decentralized AV control on bottleneck throughput by studying a scaled-down version of the post-tollbooth section (see Fig. 4.1) of the San Francisco-Oakland Bay Bridge. In our scaled version, four lanes reduce to two which then reduce down to one lane (as opposed to 15 to 8 to 5 in the bridge). While the lane numbers differ, the overall road architecture is quite similar as each vehicle goes through two merges. To design the controllers, we will use multi-agent reinforcement learning (MARL). Even at a reduced scale, this problem is a difficult MARL challenge as it incorporates:

- a large number of agents, varying between 20-200 depending on the penetration rate.
- delayed reward structure. A given vehicle's impact on outflow isn't experienced until many seconds later.
- challenging credit assignment. The outflow is a global signal and it is difficult to disambiguate whose action led to the improved outcome.

This work tackles this challenging MARL problem and provides some initial characterization of the performance of decentralized control in these settings. The main contributions of this work are:

1. We introduce a challenging new benchmark in multi-agent reinforcement learning.
2. We demonstrate that appropriately chosen multi-agent RL algorithms can be used to design decentralized control policies for maximizing bottleneck throughput.
3. We show that effective control can be performed in the fully local sensing setting where vehicles do not have access to any macroscopic observations.
4. We demonstrate and formalize a challenging problem in open transportation networks where the Nash equilibrium can deviate from the social equilibrium. We introduce a simple trick to make the two equilibria align.
5. We design decentralized feedback control policies and show that, despite extensive tuning, the RL policies sharply outperform our feedback baseline. Additionally, the RL approach is able to equal the performance of a traffic-light baseline.
6. We demonstrate that the resultant control policies can be made robust to variations in the penetration rate.

4.2 Background

Off-Policy Reinforcement Learning

Here we briefly introduce off-policy reinforcement learning as well as some of the challenges in their use in multi-agent settings. For a more thorough discussion of the underlying algorithms see[48, 86] and for the particular challenges of multi-agent off-policy algorithms see[89].

Off-policy methods focus on using a buffer of data sampled from the environment to construct the policy. While they can suffer from instability relative to policy gradient methods[2], they tend to be more sample efficient and can often be effectively run on a single CPU. The basic idea is to periodically sample data from the buffer and compute an estimate of the Bellman error

$$\mathcal{L} = \frac{1}{N}\sum_{i=1}^{N}\left(Q(s_t^i, a_t^i) - r(s_t^i, a_t^i) - \gamma \operatorname*{argmax}_a Q(s_{t+1}^i, a)\right)^2$$

where i indexes a sample from the batch, γ is the discount factor, and Q is the Q-function

$$Q^\pi(s_t, a_t) = \mathbb{E}_\pi\left[\sum_{i=t}^{T}\gamma^{i-t} r_t(s_t, a_t) | s_t, a_t\right]$$

i.e. the expected cumulative discounted reward of taking action a_t and thereafter following policy π (we will use the terms policy and controller interchangeably in this section). The sum $r(s_t^i, a_t^i) + \gamma \operatorname*{argmax}_a Q(s_{t+1}^i, a)$ is referred to as the *target*. The algorithms then perform gradient descent on the loss $\mathcal{L}$ to learn an approximation of the Q-function. DDPG simultaneously learns a Q-function for estimating the values of states and a policy that selects actions that maximize the Q-function. Both policy μ and Q-function are learned simultaneously: the Q-function is learned by minimizing the Bellman error over a batch of data using gradient descent and the policy μ is learned by performing gradient ascent on the action component of the Q-function.

TD3 creates an empirically stabler version of DDPG by adding three simple tricks:

- The target is estimated using two Q-functions instead of one and taking the minimum of the two.
- The policy network is updated significantly less often than the value network.
- A small amount of noise is added to the action when estimating the value of the Q-function. This is based on the assumption that similar actions should have similar Q-values.

For more details, please refer to[48].

In this work we use Twin-Delayed Deep Deterministic Policy Gradient (TD3)[48] a variant of Deep Deterministic Policy Gradient (DDPG)[86]. It is important to note that TD3 is a single-agent algorithm and that in multi-agent settings, there is additional instability induced by the changing policies of the other agents in the environment. Essentially, because the other agents in the environment have also changed, samples inside the buffer are *stale*, they no longer correctly represent either the rewards that would be received for taking an action in a given state nor is the subsequent state after taking that action the state that we would likely transition to. Algorithms that fail to address this issue and simply perform Q-learning while ignoring it are referred to as *Independent Learners*. This challenge can be addressed by algorithms like Multi-agent DDPG[89] which use a Q-function that sees the states and actions of all active agents.

In this work, we simply use *Independent Learners* with a shared policy: all of our agents use the same neural network to compute their actions. Surprisingly, we find this to be effective despite the issue of *stale* buffer samples discussed above.

Car-Following Models

For our model of the driving dynamics, we use the default car-following model and lane-changing models in SUMO. We use SUMO 1.1.0 which has the Krauss car-following model[75]. For the parameters of the model, we use the default values in the aforementioned SUMO version. The lane-changing model is also the default model described in[43].

Flow

We run our experiments in `Flow`[153], a library that provides an interface between the traffic microsimulators SUMO[88] and AIMSUN[11], and RLlib[85], a distributed reinforcement learning library. `Flow` enables users to create new traffic networks via a python interface, introduce autonomous controllers into the networks, and then train the controllers on many-CPU machines on the cloud via AWS EC2. To make it easier to reproduce our experiments or to try and improve on our results, our fork of `Flow`, scripts for running our experiments, reproducing our results, and tutorials can be found at `https://github.com/eugenevinitsky/decentralized_bottlenecks`.

Feedback Control and ALINEA

As our baseline for the performance of our RL controllers, we implement the traffic light controller from[106] (referred to here as ALINEA) and additionally design a decentralized variant of ALINEA that can be performed using AVs. The basic idea underlying ALINEA is to select an optimal bottleneck vehicle density and then perform feedback control around that optimal value using the ratio of red-time to green-time of the traffic light as the control parameter. We use the particular scheme outlined in[129] with some slight modifications.

Instead of operating around density, we feedback around a desired number of vehicles in the bottleneck which we denote as n_{crit}, a hyperparameter that we will empirically determine for our network. We update the desired inflow q as

$$\begin{aligned}\tilde{q}_{k+1} &= q_k + K(n_{\text{crit}} - \hat{n}) \\ q_{k+1} &= \max(q_{\min}, \min(\tilde{q}_{k+1}, q_{\max}))\end{aligned}$$

where K is the gain of the proportional feedback controller, $\hat{n}$ is the average number of vehicles in the bottleneck over the last T seconds, and $q_{\min}$ and $q_{\max}$ are minima and maxima of q to prevent issues with wind-up. We set $T = 25$, $q_{\min} = 200$, $q_{\max} = 14400$, and perform hyperparameter searches over K, n_{crit}, q_0. This desired inflow is then converted into a red-green cycle time via

$$c_k = r + g = \frac{7200 * L}{q_k}$$

where r is the red time, g is a fixed green time, and L is the maximum number of lanes. In this work, we set g to 4 which we empirically determined to be the amount of time needed to let two vehicles pass into the bottleneck. We perform this feedback update every 30 seconds. Finally, we initialize each of the traffic lights to have a cycle that is offset from each other by 2 seconds to prevent the traffic lights from being completely in sync. Further details are provided in the code.

Fortunately, we can apply the exact same strategy using autonomous vehicles instead of traffic lights where c_k is now the amount of time that an AV will wait at the bottleneck entrance before entering. However, the hyper-parameters will differ sharply as a function of the penetration rate. For a given penetration rate percentage, p, the expected length of the human platoon behind a given AV will be $\frac{1}{p} - 1$. Whereas in the traffic light case we can set arbitrary inflows, here every time an AV goes, $\frac{1}{p} - 1$ vehicles will follow it on average. As a consequence, to avoid congestion at lower penetration rates, the inflow needs to be a good deal lower. As we will discuss in Section 4.4, this leads to the decentralized control scheme under-performing traffic-light based control. For the exact hyper-parameters swept, see the appendix.

4.3 Experiments

Experiment Setup

We attempt to improve the outflow of the bottleneck depicted in Fig. 4.2, in which a long straight segment is followed by two zipper merges sending four lanes to two, and then another zipper merge sending two lanes to one. This is a simplified model of the post-ramp meter bottleneck on the Oakland-San Francisco Bay Bridge depicted in Fig. 4.1. Once congestion forms, as in Fig. 4.3, the congestion does not dissipate due to lower outflow than inflow and begins to extend upstream.

An important point to note is that in this work lane-changing is disabled for all the vehicles in this system. As we discuss in Sec. 4.4, this enables higher throughput but would

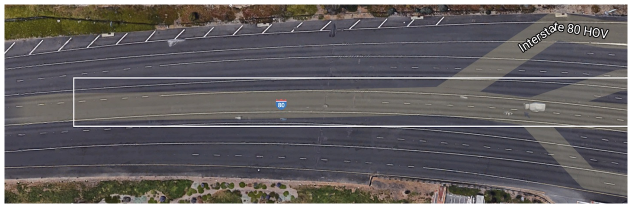


Figure 4.1: Bay bridge merge. The equivalent subsection that we study in this work is highlighted in a white square. Traffic travels from right to left.

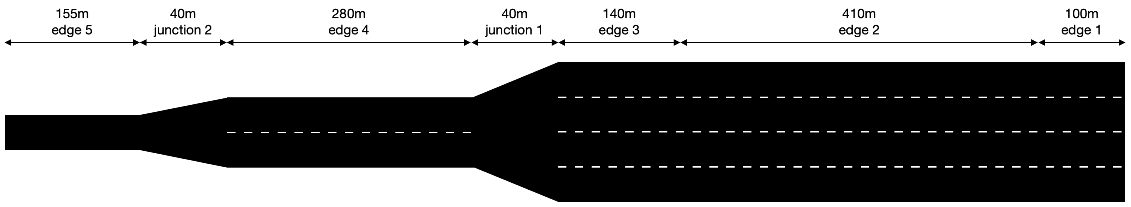


Figure 4.2: Long entering segment followed by two zipper merges, a long segment, and then another zipper merge. Scale is distorted to make visible relevant merge sections.

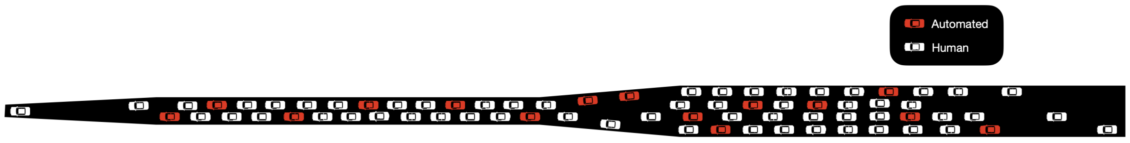


Figure 4.3: Congestion forming in the bottleneck. Congestion starts at the left (downstream) and propagates right (upstream). Red vehicles are automated, human drivers are in white.

require the imposition of new road rules at the bottleneck. Fortunately, this would only require painting some new lines that restrict lane-changing which should be relatively cheap.

Capacity Diagrams

Fig. 4.4 presents the inflow-outflow relationship of the uncontrolled bottleneck model. To compute this, we swept over inflows from 400 to 3500 vehicles per hour in steps of 100, ran 20 runs for each inflow value, and took the outflow as the average outflow over the last 500 seconds. Fig. 4.4 presents the average value and 1 std-deviation from the average across these 20 runs. Below an inflow of 2300 vehicles per hour congestion does not occur; above 2500 vehicles per hour congestion will form with high certainty. A key point is that once the congestion forms at these high inflows, at values upwards of 2500 vehicles per hour, it does not dissolve unless inflow is reduced for a long period of time. Identical inflow-outflow behavior is observed when lane-changing is enabled so a similar graph with lane-changing is

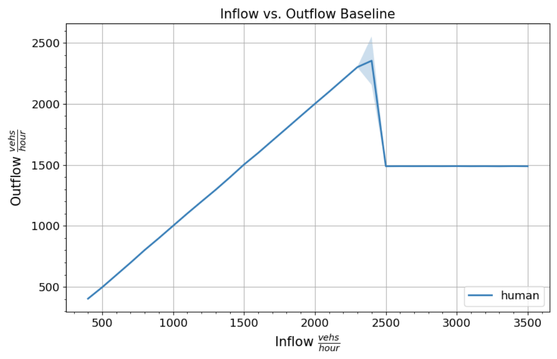


Figure 4.4: Inflow vs. outflow for the uncontrolled bottleneck. The solid line represents the average over 20 runs at each inflow value and the darker transparent section is the one standard deviation from the mean.

omitted.

Partially Observed Markov Decision Process Structure

Here we outline the definition of the action space, observation function, reward function, and transition dynamics of the POMDP that is used in our controllers. We distinguish three cases that depend on the type of sensing infrastructure that will be available to the controller. The central concern is that the observations must give some way of identifying the state of the bottleneck (speed and density) to allow the AVs to intelligently regulate the inflow. Without some estimate of the bottleneck state, the AVs must be extremely conservative to ensure that the bottleneck does not enter congestion. The estimation can be done explicitly by acquiring the bottleneck state with loop detectors/overhead cameras or implicitly, by observing the behavior of vehicles around the bottleneck and inferring what the state of the bottleneck must be.

Figure. 4.5 provides a rough overview of our considered state spaces. We call the first state set the *radar state* as the required states would be readily available via onboard radar, cameras, and GPS. This is the state space that would be most easily implemented using existing technology on an autonomous vehicle and does not use any macroscopic information. In Fig. 4.5, *radar state* would consist of the distances and speeds of the blue and green vehicles. These observable vehicles correspond to one vehicle ahead and one vehicle behind in each of the lanes. For instance, if the vehicle is on a segment with four lanes, it will see up to 8 vehicles in its state, with padding for missing vehicles. We also note that although AVs are only controlled in a congested regime on edge 3 (cf. Fig. 4.2), in which leader and follower vehicles in all lanes are usually close by, a current limitation of our work is that we assume our ego vehicle has an infinite, unobstructed sensing range. However, we study the effect of restricting the sensing range in Sec. 4.4.

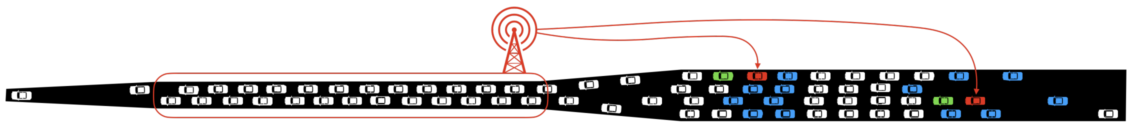


Figure 4.5: Diagram representing the different state spaces. The red vehicles can see the green vehicles in the minimal state space, and the blue and green vehicles in the radar state space. In the minimal and aggregate state spaces, the red vehicle also has access to information about the vehicle count in the bottleneck which we represent as a tower communicating information about the highlighted red segment. In addition to the states indicated here, the aggregate state space also contains the average speeds of edges 3, 4, and 5 (see Fig. 4.2).

We also provide a smaller state space we call the *minimal* state space which essentially consists of the speed and distance of only the green vehicle in Fig. 4.5 as well as the vehicle counts in the bottleneck. This is a small state space intended for fast learning; we have significantly pruned it by hand-picking what we believe to be a minimal set of states with which the task can be accomplished. We also investigate an *aggregate* state that provides macroscopic data about the bottleneck and can be added to any existing state space. The aggregate state consists of the number of vehicles in the bottleneck and the average speeds of vehicles on edges 3, 4, and 5 (see Fig. 4.2 for the numbering). These states would be available given appropriate loop sensing infrastructure or a sufficient number of overhead cameras distributed throughout the bottleneck.

We note that every state space contains the ego vehicle speed, its GPS position on the network, and a counter indicating how long the vehicle has stopped. This counter is used to enable the controller to track how long it has waited to enter the bottleneck. For a more detailed breakdown of which observations are available in the three previously defined states, see Appendix Sec. A.2

From these three potential sets of states, we form three combined state spaces that we study: radar + aggregate, minimal + aggregate, and minimal alone. Each of these represents a different set of assumptions on what sensing technology will be available ranging from full decentralization (radar alone) to having access to macroscopic information (minimal, aggregate). We characterize the relative performance of these different state spaces in Sec. 4.4. Note that each of these state spaces has some information about the state of the bottleneck, which we found critical for getting good performances; for this reason, we do not include state spaces that include no information about the bottleneck, such as radar alone or minimal without bottleneck counts.

The action space is simply a 1-dimensional acceleration. While we could include lane changes as a possible action, we leave this to future work. To prevent the vehicles from forming unusual patterns at the entrance of the bottleneck, control is only applied on edge 3 (edges numbered according to Fig. 4.2). However, states and rewards are received at every time-step and consequently, actions are computed at each time-step: we simply ignore the controller output and use the output of the car-following model unless we are on edge 3.

We are trying to optimize throughput, so for our reward function we simply use the number of vehicles that have exited in the previous time-step as a reward

$$r_t(s_t, a_t) = n_t/N$$

where n_t is the number of vehicles that have exited in that time-step and N is a normalization term that was used to keep the cumulative reward at reasonable values. We use $N = 50$ in this work. Since the outflow is exactly the quantity we are trying to optimize, optimizing our global reward function should result in the desired improvement. This is a global reward function that is shared by every agent in the network.

However, we note a few challenges that make this a difficult reward function to optimize. First, the reward is global which causes difficulties in credit assignment. Namely, it is not clear which vehicle's action contributed to the reward at any given time-step. Secondly, there is a large gap between when an action is taken and when the reward is received for that action. That is, a vehicle choosing to enter the bottleneck does not receive any reward directly attributable to that decision for upwards of 20 steps. Finally, the bottleneck being fully congested is likely a local minimum that is hard to escape. Once congestion has set, it cannot be removed without a temporary period where the inflow into the bottleneck is reduced. However, a single vehicle choosing to not enter the bottleneck would have a negligible effect on the inflow, making it difficult for vehicles to learn that decongestion is even possible.

Divergence Between Nash Equilibrium and Social Optimum

Here we provide a simple illustrative example of how, despite having a single, global reward function, open networks can lead to non-cooperative behavior. In the case where every vehicle receives the same reward at every time-step, it is simple to see that the Nash Equilibrium will be the same as the social optimum. However, vehicles sharing the same reward function but optimizing over different horizons can cause a divergence between the two equilibria. The key intuition is that although all of the vehicles are trying to optimize the same quantity, they only receive rewards while they are in the system as their trajectory terminates once they go through the exit. This creates a perverse incentive to remain in the system for longer than is socially desirable, leading to a divergence from the social optimum. Consider the following simplified, single-step variant of the bottleneck in which there are simply two vehicles. The problem has the following reward structure before we introduce the open-endedness:

- If both vehicles go, congestion occurs and they receive a reward of 1.
- If one vehicle goes and the other doesn't, no congestion occurs. Both vehicles receive a reward of 2.
- If both vehicles do not go, no outflow occurs and they receive a reward of 0.

Table 4.1: One time-step model of the bottleneck reward structure. Here we make sure to reward the vehicle that exited the system, even though in an MDP its trajectory would have ended and no reward would have been given.

		Vehicle 2	
		Go	No Go
Vehicle 1	Go	$(1, 1)$	$(2, 2)$
	No Go	$(2, 2)$	$(0, 0)$

Table 4.2: One time-step model of the bottleneck reward structure where vehicles do not receive reward after they exit.

		Vehicle 2	
		Go	No Go
Vehicle 1	Go	$(1, 1)$	$(0, 2)$
	No Go	$(2, 0)$	$(0, 0)$

This problem mimics the structure of the bottleneck where it is necessary to restrict the inflow to maximize the outflow. It is straightforward to see that the optimum is achieved when one vehicle goes and the other doesn't and that this is both the social optimum and a Nash equilibrium. The game is depicted in Table 4.1 where it can visually be verified that (No-Go, Go) and (Go, No-Go) are Nash Equilibria but (No Go, No Go) is not an equilibrium point.

Open networks modify the problem in that once a vehicle exits it ceases to receive any reward. Therefore, the vehicle that goes does not actually observe any outflow and will receive a reward of zero.

- If both vehicles go, congestion occurs and they receive a reward of 1.
- If one vehicle goes and the other doesn't, no congestion occurs. The vehicle that went receives a reward of 0 while the other one receives a reward of 2.
- If both vehicles do not go, they receive a reward of 0.

In this setting, depicted in Table. 4.2, (No Go, No Go) is now a weak Nash Equilibrium. From the perspective of learning, this is a ubiquitous equilibrium as the vehicles that chose not to go will tend to accumulate a lot of reward. An easy solution to remove this equilibrium is to adopt the perspective of the game in Table 4.1 and continue to reward vehicles even after they leave the system. However, this would create a very noisy reward function as agents that exit the system earlier would receive a lot of reward from states and actions that they did not particularly influence. An alternative variant is to keep all the agents persistently in the system: run the system for some warm-up time to accumulate a starting number of vehicles and after that, any vehicle that exits is rerouted back to the entrance. We adopt this choice and reroute the vehicles during the training. However, this could lead to vehicles manipulating the bottleneck across reroutes and so we turn this rerouting behavior off when testing the policies after training.

Experiment Details

For the training parameters for TD3, we primarily used the default parameters set in RLlib [84] version 0.8.0, a distributed deep RL library. The percentage of autonomous vehicles varies among 5%, 10%, 20%, and 40%. During each training rollout, we keep a fixed inflow of 2400 vehicles per hour over the whole horizon. At each time-step, a random number of vehicles are emitted from the start edge. Thus, the number of vehicles in each platoon behind the AVs will be of variable length and it is possible that at any time-step any given lane may have zero AVs in it. To populate the simulation fully with vehicles, we allow the experiment to run uncontrolled for 300 seconds as a warm-up. After that, we run an RL rollout for 1000 seconds.

We use the traffic micro-simulator SUMO[88] for running our simulations. At training time, we use the re-routing technique discussed in Sec. 4.3 where vehicles are simply placed back at the beginning of the network after exiting. It is essential to note that for the multi-agent experiments we used a *shared controller*, all of the agents operate in a decentralized fashion but share the same controller.

For more details, see the Appendix.

4.4 Results

In this section, we attempt to provide experimental results that answer the following questions:

1. How does the ability to improve bottleneck throughput scale with available sensing infrastructure? With penetration rate?
2. Is there a single controller that will work effectively across all penetration rates?
3. Can we construct an effective controller that uses purely local observations?

Effect of Sensing

Here we compare the relative performance of the different sensing options across different penetration rates. Fig. 4.6 compares the evolution of the inflow-outflow curve of the three state spaces to the uncontrolled case (labeled human), the traffic light baseline (labeled

ALINEA), and the hand-designed feedback controller operating at a 40% penetration rate. Each of the state spaces outperforms the hand-designed feedback controller at every penetration rate and provides a 15% improvement in the outflow even at the 5% penetration rate. To study the evolution of the outflow with penetration rate, Fig. 4.7 illustrates the outflow at an inflow of 2400 as a function of penetration rate. Only the *radar + aggregate* state space is able to consistently take advantage of increasing penetration rate. Excitingly, its performance at a 40% penetration equals the performance of a traffic light-based controller. Table 4.3 summarizes the values of each of the different state spaces at an inflow of 3500 vehicles per hour.

Table 4.3: Average outflow and its variance at an inflow of 3500 vehicles per hour, as a function of the penetration and the state space.

	minimal	**minimal + aggregate**	**radar + aggregate**
5%	1803 ± 83	1813 ± 114	1817 ± 80
10%	1829 ± 46	1863 ± 76	1888 ± 47
20%	1811 ± 29	1897 ± 46	1980 ± 48
40%	1878 ± 40	1910 ± 46	2034 ± 45

Is There a Universal Controller?

In Sec. 4.4 we train a separate controller for each penetration rate. Atop the additional computational expense needed to train a new controller for each data-point, having one controller per penetration rate might require an accurate online estimation of current penetration rates so as to switch to the appropriate control scheme. Here we point out (see Fig. 4.8) that at least for the controllers studied here, this concern is justified: a controller trained at one penetration rate and evaluated at another will underperform a controller trained at the latter penetration rate. We also investigate a simple dynamics randomization strategy where we randomly sample a new penetration rate at each rollout and confirm that this can yield a controller that performs effectively across penetration rates albeit with some small loss of performance.

The key challenge is that the appropriate amount of time needed to wait before entering the bottleneck is a function of the penetration rate. As a simplified model to generate intuition, imagine that the bottleneck deterministically congests if more than 11 vehicles enter it. We will refer to an AV with N vehicles behind it as a *platoon of length N*. At a penetration rate of 10%, the average platoon length is 9. If we have two AV platoons ready to enter the bottleneck, one of the platoons must wait until the other platoon is almost completely into the bottleneck or else it will congest. At a 20% penetration rate (a platoon of average length 4), however, two platoons can go at once without worrying about running into congestion.

As a result, a controller trained at low penetration rates may be too conservative when deployed at higher penetration rates while a controller trained at high penetration rates may not be conservative enough at low penetration rates. As demonstrated in Fig. 4.8, a controller trained at a 10% penetration rate does significantly worse when deployed at a 40% penetration rate. Thus, if we use the controllers trained in Sec. 4.4, it will be necessary to use either infrastructure or historical data to identify the current penetration rate and deploy the appropriate controller. This motivates our attempt to find a single controller that is stable across penetration rates. We demonstrate that in return for some small degradation in performance, we can construct a single controller that performs robustly across penetration rates. To achieve this, we use *dynamics randomization* and for each trajectory we sample a new penetration rate p uniformly from $p \sim U(0.05, 0.4)$. We refer to these as *universal controllers* and the controllers trained at individual penetration rates in Sec. 4.4 as *independent* controllers. Fig. 4.9 shows the performance of the universal controllers compared to the controllers trained at a penetration rate and evaluated at the same penetration rate for each of the three state spaces we are using.

From Fig. 4.9, we can see that the results do not have a consistent trend as the universal controllers both over-perform and underperform at different penetration rates. For example, the minimal + aggregate controller gives up 100 vehicles per hour at a 5% penetration rate but outperforms the independent controller by 250 vehicles per hour at high penetration rates. However, we note that even the worst outcome at low penetration rates, the universal minimal controller, outperforms the uncontrolled baseline of 1550 vehicles per hour. Additionally, the universal radar + aggregate controller consistently provides an outflow of 1850 vehicles per hour which achieves the desired goal of an effective controller independent of penetration rate.

Controllers Without Macroscopic Observations

The three state spaces studied earlier, minimal, minimal + aggregate, radar + aggregate, all contain the number of vehicles in the bottleneck (edge 4 in Fig. 4.2) in the state space as well as average speed data on edges 3, 4, 5 in the aggregate cases. Acquiring this information consistently (rather than through a lucky LIDAR or radar bounce picking up many vehicles ahead of the ego vehicle) would require either camera or loop sensing infrastructure. We would like to understand whether efficient control can be done without access to the number of vehicles in the bottleneck, a quantity we refer to as *congest number*. This would enable us to perform control that is fully decentralized in both action and observation, allowing us to deploy these systems with no additional infrastructure cost.

We attempt to answer this question by using the radar state space alone without any aggregate information. Thus, the vehicle only has access to the speeds and distances of the vehicles directly ahead of and behind it in each lane. Although it is not obvious how the controller will accomplish inference of the *congest number*, a few possible options:

1. There exists a scheme that does not actually depend on the number of vehicles in the

bottleneck.

2. The number of vehicles in the bottleneck can be inferred from the distance to the visible vehicles or the speed of the visible vehicles. For example, if a vehicle observed in the bottleneck is stopped that likely indicates congestion while high velocities would indicate free flow.

3. Vehicles can learn to communicate through motion by adopting vehicle spacing and velocities that observing vehicles can use to infer the *congest number*. For example, a velocity between 2 and 3 m/s could indicate 0-5 vehicles in the bottleneck, between 3 and 4 m/s could indicate 6-10 vehicles, and so on.

While we are not able to conclusively establish which of the above hypotheses are active, Fig. 4.10 demonstrates the effect of removing any macroscopic information from the state space. The radar state space with no macroscopic data, marked in orange, deviates from the aggregate state space by about 100 vehicles per hour but still sharply outperforms an uncontrolled baseline (marked human). Since radar sensors are now standard features on level 2 vehicles, this suggests that a fully decentralized controller can be deployed using available technology.

Robustness of Simplifications

In this section, we attempt to relax some of the simplifying assumptions made in the design of our problem. Specifically, we investigate the effects of the following changes:

- Lane-changing. In the prior results, we have disabled lane-changing. We now study whether our best controllers are able to handle adding lane-changing to the human driver dynamics without retraining the RL controller.

- Simplifications in the “radar” model. Our “radar” state space does not take into account occlusions and can return a vehicle an arbitrary distance away.

In Fig. 4.11, we enable lane-changing and examine how effective our controllers are as the penetration rate evolves. Unsurprisingly, at low penetration rates, there is a sharp reduction in outflow relative to the lane-changing disabled setting. The challenge is that when one of the AVs goes, other vehicles will rapidly lane-change into its lane which prevents the AV from restricting the inflow. As the penetration rate increases, when an AV at the front of the queue goes, a new AV rapidly arrives to replace it which consequently minimizes the impact of lane-changing. However, we note that when vehicles lane-change in SUMO, they instantly change lanes which may enable more aggressive lane-changing than is physically possible. Hence, the degradation in outflow might be lower in reality than it is in our simulator. Furthermore, disabling lane-changing at a bottleneck by painting new road lines should be relatively cheap.

As for the question of a more “realistic” radar model, Fig. 4.12 presents our attempt to restrict the range beyond which vehicles cease to become visible. We take a trained policy and cap how far it is allowed to see as an approximation of a restricted radar range. We try two restrictions, not seeing past 20 meters and not seeing past 140 meters. If there is no vehicle within that range, we set a default state with a distance of 20 or 140 meters respectively, a speed of 5 meters per second, and treat it as a human vehicle by passing a zero to the boolean that indicates whether a vehicle is human or autonomous. Other replacements in the state for vehicles that are too far away to see are possible, we could instead replace the missing states with a vector of -1's but found empirically that the replacement discussed led to better performance. We find that the *universal* controllers are relatively robust to this replacement which suggests that their controller is more independent of actual vehicle sensing and more dependent on macroscopic states. This ablation, simply ignoring vehicles that are too far away, is an extremely approximate model of how radar might work, and replacing it with more accurate radar models is a topic for future work.

Instability of Reward Curve

For purposes of reproducibility, we provide a few representative reward curves from some of the training runs. These should help establish a sense of what the expected reward is as well as provide a calibrated sense of what fraction of training runs are expected to succeed. Since we use 36 CPU machines and each training run requires 1 CPU, we are able to train 35 random seeds in parallel (1 CPU is used to manage all the trainings) and we keep the best performing one. The following figure presents the results of 9 of these random seed trials (for better visibility) from one of the training runs.

Fig. 4.13 and Fig. 4.14 represent reward curves from low penetration rate runs (10%) and high penetration rate runs (40%) respectively. A high-scoring run corresponds to an average agent reward of around 10, a reward slightly above eight corresponds to all the vehicles just zooming into the bottleneck without pausing, and a reward below 2 corresponds to the vehicles mostly coming to a full stop. As is clear from the figures, at low penetration rates the training is relatively stable and converges quickly. However, as the penetration rate increases the reward curves become extremely unstable, with rapid oscillations in the expected reward. This instability is not due to variations in the outflow as the std. deviation of the outflow is low but is likely the outcome of applying independent Q-learning in a multi-agent system, leading to non-stationarity in the environment. Methods that explicitly handle this non-stationarity by using a centralized critic such as MADDPG[89] may help reduce the instability in the training.

4.5 Conclusions and Future work

In this work, we demonstrated that low levels of autonomous penetration, in this case 5%, are sufficient to learn an effective flow regulation strategy for a severe bottleneck. We demonstrate that even at low autonomous vehicle penetration rates, the controller is able to improve the outflow by 300 vehicles per hour. Furthermore, we are able to use additional availability of AVs and at a 40% penetration rate get equal performance to actually installing a new traffic light to regulate the inflow.

However, many open questions remain. In this work, we use an independent Q-learning algorithm which leads to serious instability in the training. As discussed in Sec. 4.4, at high penetration rates many of the training runs become unstable, which makes it unclear if we are near to the optimal policy for those penetration rates. This is a challenging multi-agent RL task and it would be interesting to see whether multi-agent RL algorithms that use a centralized critic like MADDPG[89] and QMIX[116] would lead to more stable training. Furthermore, the training procedure takes 24 hours so finding algorithms that can perform with higher sample efficiency is critical.

One possible direction to pursue in future work is to increase the level of realism, adding both lane-changing and an accurate radar model that correctly accounts for obscurity. In preliminary investigations in Sec. 4.4, we found that lane-changing degrades the performance of our controllers as vehicles simply lane-change into the lane that is currently moving and thus avoid inflow restriction. It may be the case that this behavior can be avoided by more complex coordination between the AVs in which they explicitly arrange themselves to block this lane-changing behavior. Preliminary experiments we ran in which training was done with lane-changing enabled did not yield particularly strong results but this may be an artifact of our choice of the training algorithm.

Another open question is to investigate the effects of coordination between the vehicles. In the decentralized case, we still do not know the extent to which the AVs are coordinating in their choice of action. While implicit coordination is possible due to vehicles being aware of which nearby vehicles are also autonomous, we have only provided circumstantial evidence that this is actually occurring. In the case of lane-changing, such coordination may be needed to prevent human drivers from skipping lanes and decreasing the total outflow. Additionally, we do not use memory in any of our models which may be limiting the effectiveness of our controllers. Using memory-based networks such as LSTMs could be an interesting direction for future work.

One approach we intend to explore to explicitly enable coordination is to allow the AVs to communicate amongst themselves. Perhaps by broadcasting signals to nearby vehicles, the AVs can learn to coordinate platoons in such a way that changing lanes no longer appears advantageous to the human driver and they will remain in their lane. Furthermore, if communication proves useful, it is possible that the AVs may develop a “language” that they use for coordination. Examining whether such a “language” emerges is a future thread of work.

Finally, an exciting potential consequence of these results, given that they’re decentralized and only use local information available via vision, is that human drivers could potentially implement these behaviors. Investigating whether this scheme could be deployed via human driving, whether by constructing a mobile app that provides instructions or by teaching a new driving behavior, is a direction we hope to explore in the future.

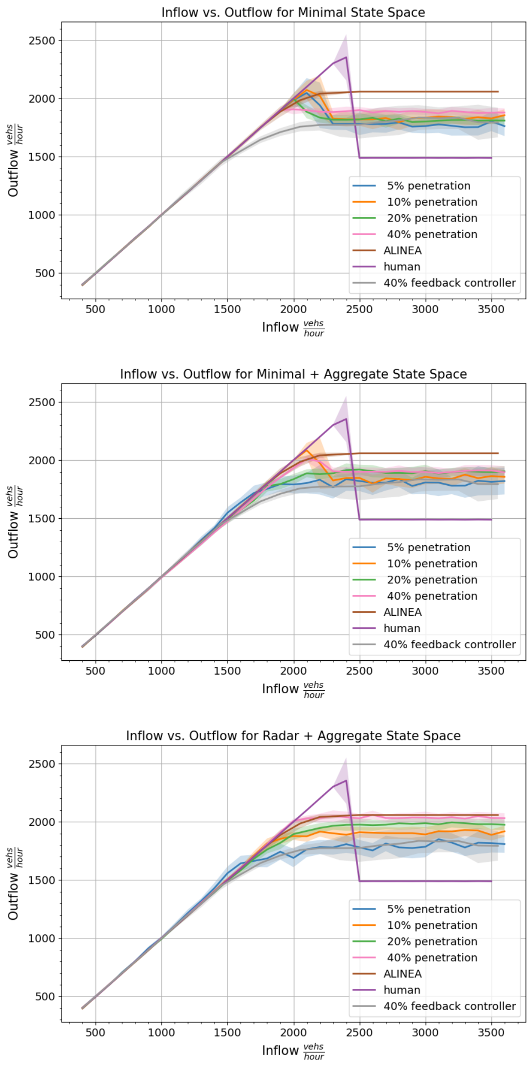


Figure 4.6: From top to bottom, the evolution of the inflow vs. outflow curve as penetration rates evolve for minimal, minimal + aggregate, and radar + aggregate. Within each figure we plot the performance of our controllers trained at four different penetration rates, the traffic light baseline ALINEA, the performance of our feedback controllers at a 40% penetration rate, and the uncontrolled curve marked "human".

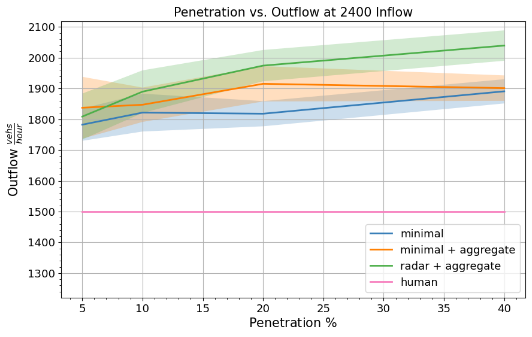


Figure 4.7: Evolution of the outflow as a function of the penetration rate for the three state spaces we are using. We also plot the uncontrolled human baseline for reference.

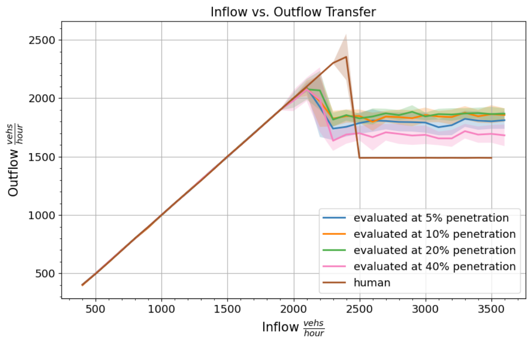


Figure 4.8: The inflow vs. outflow curve for a controller trained at 10% penetration on the minimal + aggregate state space, and evaluated at 5%, 10%, 20%, and 40% penetration. We also plot the uncontrolled human baseline for reference.

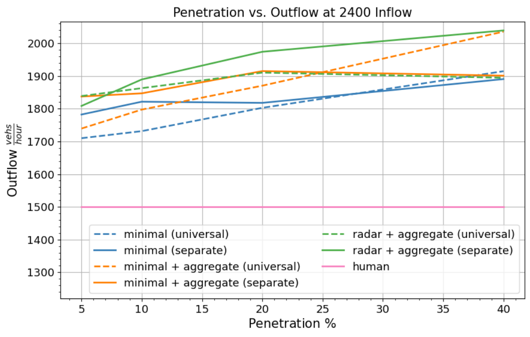


Figure 4.9: Evolution of the outflow as a function of the penetration rate for the three state spaces we are using. For each state space, we compare the universal controller trained using dynamics randomization and evaluated at different penetration rates, to the four independent controllers trained and evaluated at their own penetrations rate of respectively 5%, 10%, 20% and 40%. We also plot the uncontrolled human baseline for reference.

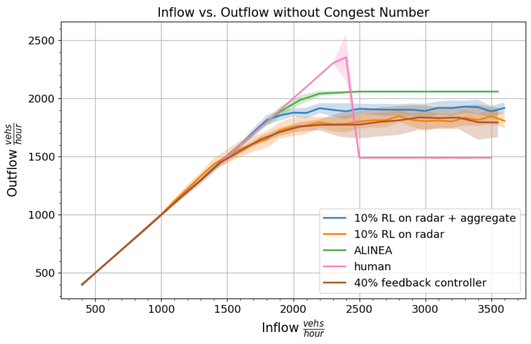


Figure 4.10: Evolution of the inflow vs. outflow curve for controllers trained at a penetration rate of 10%. We compare a controller trained on the full radar + aggregate state space to a controller only trained on the radar state space, which means it doesn't have access to the number of vehicles in the bottleneck. We also plot the performance of the traffic light baseline ALINEA, of our feedback controller at 40%, and of the uncontrolled curve marked "human".

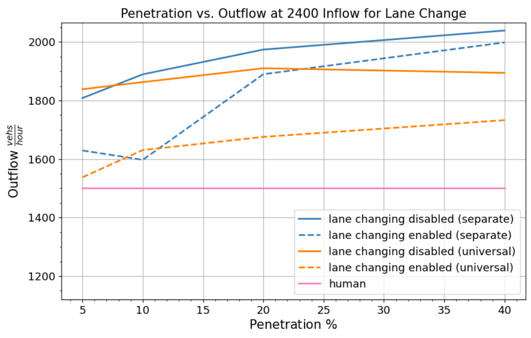


Figure 4.11: Evolution of the outflow as a function of the penetration rate for controllers trained on the radar + aggregate state space. We compare controllers that have been trained with lane-changing disabled, to those sames controllers when lane-changing is enabled at evaluation time. We compare both controllers trained on a fixed penetration rate of 5%, 10%, 20% or 40%, referred to as "separate", and controllers trained at a random penetration rate between 5% and 40% as explained in 4.4, referred to as "universal". We also plot the uncontrolled human baseline for reference.

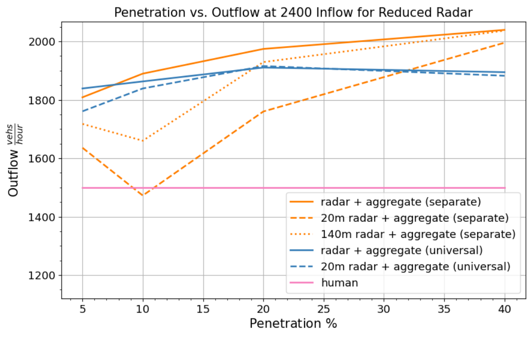


Figure 4.12: Evolution of the outflow as a function of the penetration rate for controllers trained on the radar + aggregate state space. We compare controllers that have been trained with the (normal) entire radar state space, to those same controllers when the radar is restricted to seeing only vehicles up to a distance of 20 meters or 140 meters at evaluation time. We compare both controllers trained on a fixed penetration rate of 5%, 10%, 20% or 40%, referred to as "separate", and controllers trained at a random penetration rate between 5% and 40% as explained in 4.4, referred to as "universal". We also plot the uncontrolled human baseline for reference.

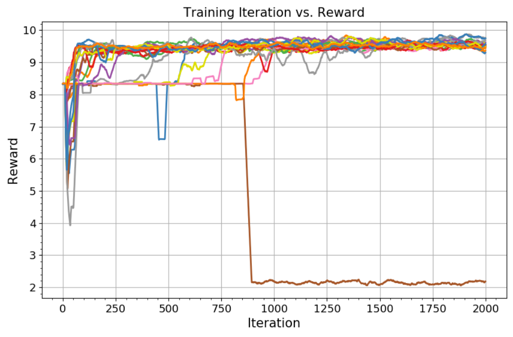


Figure 4.13: A sample of runs from a seed sweep at a penetration rate of 10%. Most of the seeds converge to a good policy.

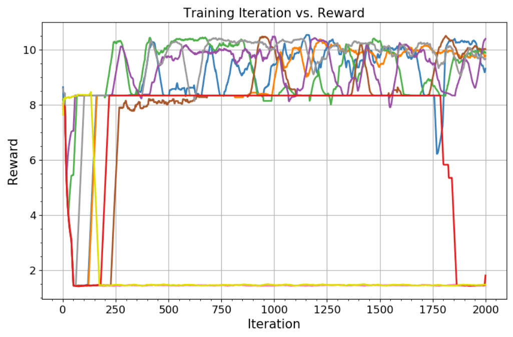


Figure 4.14: A sample of runs from a seed sweep at a penetration rate of 40%. There is significant instability that is expected from using single-agent algorithms in a multi-agent problem.

Chapter 5

Deep Reinforcement Learning for Fuel Consumption Reduction in Multi-Lane Road Networks

5.1 Introduction

Designing controllers and analyzing their energy impact is difficult due to the complexity of traffic: non-linear driving dynamics, lane changes, merges, etc. Hence, controllers are often designed and analyzed in simple settings whose relationship to actual highway networks is not entirely clear. For example, a significant fraction of recent traffic smoothing controllers are designed and analyzed with respect to a closed circular ring of dense traffic, a setting in which energy-consuming waves form spontaneously and persist throughout the network [138]. While this network is amenable to analysis and can model a single lane of traffic as it becomes infinitely long, the simplicity of the network makes it unclear how controllers designed in these settings will perform as complexity increases. Furthermore, these simple systems often have pernicious optimal solutions like slowing to a stop and gradually accelerating up to the equilibrium speed of the ring.

In this chapter, we focus on developing robust, traffic smoothing controllers for a system containing both traffic waves as well as lane changes. We build a multi-lane model (shown in Fig. 5.1) of a section of the Ventura Freeway in Los Angeles containing both on-ramps and lane drops. This system contains approximately one thousand vehicles and stretches about one mile, allowing us to see any possible long-range interactions between AVs, waves, and lane changing behavior. Using on-policy multi-agent reinforcement learning, we design traffic smoothing controllers that create a sharp increase in the energy efficiency of the traffic flow; these controllers also outperform a variety of available baseline controllers. The state input to our controller is easily implementable using radar or cameras, making it an easy add on to existing cruise controllers.

Since it is highly likely that our simulator is not accurate in a variety of ways (imperfect

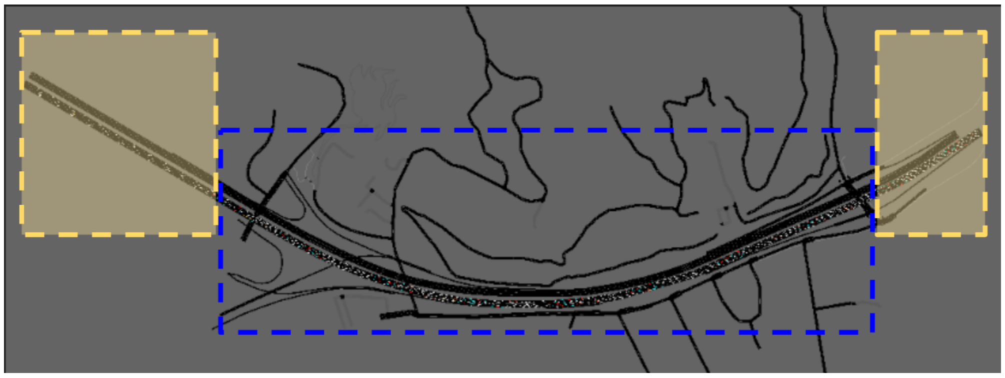


Figure 5.1: The I-210 network simulated within SUMO. Yellow areas represent the uncontrolled ghost cells, while the blue rectangle shows where control is applied and where metrics are computed.

model of human driving dynamics, vehicle dynamics, etc.), we demonstrate that our controller is a good candidate for deployment by performing a set of robustness tests. We sweep over a wide variety of the parameters that define driving behavior and system dynamics in our simulator and show that our controller maintains good performance under these changes. As we demonstrate, our controller appears robust to all these axes of variation.

The contributions we include in this chapter are as follows:

- We build and release a new, large-scale traffic network for investigating the effect and potential of traffic-smoothing autonomous vehicles.
- We use multi-agent reinforcement learning to construct controllers that sharply improve the energy efficiency of highway traffic. We demonstrate that our controllers generalize outside their training distribution and act like *controllers that know the equilibrium speed of the system.*
- We perform a variety of robustness checks and demonstrate that our controller is robust to a wide range of potential driving conditions.

5.2 Related Work

In the seminal work of [138] it was experimentally shown for the first time that traffic streams can exhibit what are known as 'phantom-jams' in which a moving traffic jam can form without any outside prompts, such as lane-reductions or accidents. Following work in [132] empirically showed that under the same setup the phantom-jams that form could be effectively dissipated using a single automated vehicle running a control algorithm. In addition to increasing the average speed, and reducing the speed variance of the system, a significant increase in fuel efficiency for the vehicles was also found [133, 157]. These result established the potential of mixed-autonomy control to improve throughput and energy efficiency in relatively simplified settings.

Extensive work has been done to find new mechanisms by which mixed-autonomy can be used to improve transportation systems. [66, 33], consider the use of platoons of autonomous vehicles operating as moving bottlenecks to both dampen stop-and-go waves and minimize the effects of capacity drop. Other works consider the potential of vehicle-to-infrastructure coordination as a tool for eco-driving, a concept in which a controlled vehicle modifies its speed and acceleration profile to realize energy gains. [9] demonstrates the deployment from simulation to the roadway of coordination between a vehicle and a signalized intersection and shows marked improvements in energy efficiency albeit at the cost of travel time. [6] demonstrates in a physical experiment with ghost cars that a CAV using prediction of the lead vehicle trajectory or communicating with AVs further ahead in the string can sharply improve the energy efficacy of a drive.

Recently, many controllers for mixed autonomy settings have been generated using techniques from Reinforcement Learning. Reinforcement learning has been used in [153] to demonstrate that a vehicle equipped with memory could equilibrate the system for a wide range of ring densities. Other works have focused on the potential of reinforcement learning to improve traffic at scale. In [35, 164, 98, 142], multi-agent RL was used to optimize merges in a fully decentralized fashion. Both [54] and [150] concurrently used decentralized multi-agent RL for optimizing a scaled model of the San Francisco-Oakland Bay Bridge. Reinforcement learning has also seen significant use in traffic light pattern optimization [163, 26] as well as to develop traffic light controllers that could quickly adapt to new settings using Meta-RL [162].

5.3 Smoothing in Multi-Lane Systems

Our goal is to study traffic smoothing in a setting with large numbers of vehicles, ubiquitous lane changes and multiple possible sources of onset mechanisms for wave formation. We choose a segment of the Ventura Freeway, or Interstate 210 (I-210), in California. This segment is approximately one mile long and can hold up to 2000 vehicles. It varies between five and six lanes over its length and has an on-ramp that can serve as a possible source of congestion formation; however, this on-ramp is disabled in this work since we use a different mechanism to generate congestion, as explained in Sec. 5.4. Due to the combination of the multi-lane nature and its high capacity, this network serves as an effective testbed for the complexity of realistic wave smoothing.

The challenge in this network is to improve the energy efficiency by eliminating traffic shockwaves that occur along this system, which we will refer to as *phantom jams*. These shockwaves are known to appear in real systems [123] and decrease the energy efficiency of travel by leading to patterns of braking and acceleration. By eliminating the phantom jams, we improve the energy efficiency of the system. As we will demonstrate, an interesting feature of these phantom jams is that they can be removed with minimal effect on the traffic flow: decreasing the fuel consumption of the roadway is something that can be achieved without any trade-offs on the system throughput.

5.4 Problem Formulation

I-210 model with phantom jams

The I-210 network has been imported from Open Street Maps into the microscopic traffic simulator SUMO [88]. The network is shown in Fig. 5.1. Traditionally, traffic congestion is hypothesized to be caused by 'bottlenecks' in which a road network cannot support as much flow through a downstream section as is being sent from the upstream. This discrepancy in capacity to receive compared to amount sent subsequently creates traffic congestion in which vehicles are forced to drive closer together and at a lower speed than they would otherwise.

One of the benefits of the ring-road as a well-posed traffic simulation environment is that congested regimes can be set directly by choosing a number of vehicles for a given ring length (i.e. the density is set directly). However, the ring lacks crucial components of realistic traffic such as lane-changing and routing choice. In order to allow for such traffic maneuvers the multi-lane, multi-edge, network present in the I-210 network is used. In addition, a subsequent downstream flow condition is imposed directly in the form of a decreased speed limit along a small portion of the end of the network. We refer to this speed limit as the *downstream speed*. By doing so, the congested regime for the traffic can be set in a very similar manner to the ring road. This downstream condition can then be varied to allow more or less flow through the end of the network, which allows for testing the proposed control framework across a number of traffic regimes.

Human controllers

An important component of micro-simulation is the *car-following* and *lane-changing* logic that individual vehicles in the simulation adhere to. Car-following refers to how vehicles manage their longitudinal motion within a lane as opposed to their lateral, lane-changing behavior.

For the lane-changing logic, we use the default model provided in SUMO [88], the traffic micro-simulator that we use. The dominant cause of lane-changing in this model mostly consists of a vehicle lane-changing for speed gain, i.e. it will lane-change if it can drive faster in the other lane.

As for the car-following logic, it is generally modeled as *ordinary differential equations* that dictate an ego vehicle's motion based on the state of the vehicle ahead of it. In this work a first-order discretization of the *Intelligent Driver Model* (IDM) [68] is used, which dictates a vehicle's longitudinal acceleration, and is of the form:

$$\begin{aligned} v_{t+1} &= v_t + \Delta t \times a \left[1 - \left(\frac{v_t}{v_0}\right)^{\delta} - \left(\frac{s^*\left(v_t, \Delta v_t\right)}{s_t}\right)^2\right] \\ &\quad + \sqrt{\Delta t}\,\mathcal{N}\left(0, \sigma\right) \end{aligned} \tag{5.1}$$

with

$$\begin{aligned} s^*\left(v_t, \Delta v_t\right) &= s_0 + v_t T + \frac{\max\{0, v_t \Delta v_t\}}{2\sqrt{ab}} \\ s_{t+1} &= s_t + \Delta t \Delta v_t \end{aligned} \tag{5.2}$$

where v_t is the ego vehicle speed at time t, Δv_t is the difference between the leading vehicle's speed and the ego vehicle's speed, s_t is the distance to the lead vehicle, and a, b, s_0, T, v_0 and δ are all parameters of the model. t indexes the time-step and Δt refers to the size of the simulation step. $\mathcal{N}\left(0, \sigma\right)$ is zero-mean Gaussian noise used to perturb the accelerations at each time-step and is intended to represent both aleatoric and epistemic noise.

For this model, we select values of a, b such that the resultant dynamics are *string-unstable*. When a car-following model is string-unstable [140], small disturbances can grow in magnitude into large disturbances that propagate along a string of vehicles, allowing the *phantom jam* to propagate rather than dissipate. In this work, the criteria by which a and b are set is drawn from [36] due to their simplicity of the polynomial condition therein for determining string instability. All other parameters are chosen as being the default IDM values specified in [68].

Finally, since discretized car-following models can lead to collisions, we clip the output acceleration values such that collisions are not possible. The condition we use for safety is maximally conservative: an acceleration is unsafe if the lead car, braking at its maximum deceleration, will unavoidably collide with the ego vehicle. The exact implementation of this condition can be found in [88]. On top of that safe velocity failsafe, we also clip the output acceleration to respect the road speed limit and the acceleration bounds of the vehicle.

Energy model

As our calibrated energy model, we use a polynomial fit to a black box model of a Midsize Sedan model provided by Toyota. The calibrated model is shown in Fig. 5.3, and is a function of the instantaneous speed and acceleration. This model assumes a constant vehicle mass of 1743 kg.

The model shown here is fitted with a third-order polynomial, effectively smoothing out the effects of gear shifting that might otherwise be present. We do not attempt to fit this as it would require excessively large polynomial coefficients and the particular positions of the jumps due to gear shifting will vary sharply from vehicle to vehicle. For the derivation, coefficients of the polynomials, and full model details see [78].

However, it is not obvious that optimizing the energy model we use will translate the heterogeneous traffic. We argue that minimizing the energy model is equivalent to regulating around an optimal speed. While that optimal speed will vary from engine model to engine model, as long as that optimal speed is greater than the downstream speed, the optimal behavior is irrespective of engine type. For the Midsize Sedan model, the optimal operating speed is around 16.7m/s, which is well above any congestion speeds that might occur. A survey of the energy models available in Autonomie [1] suggests that the optimal speed for most engines is above this value. Since the downstream speed sets a system speed limit, the optimal solution for most engines will consequently be elimination of the waves and so we expect our results to hold generally across different energy models.

Multi-agent reinforcement learning

In this section, we discuss the notation and describe in brief the key ideas used in reinforcement learning. The system described in this article solves tasks which conform to the standard structure of a finite-horizon, discounted, decentralized multi-agent POMDP (Dec-POMDP) [103], an abstraction in which groups of agents with partial access to the true world state seek to optimize a discounted reward function across time. The Dec-POMDP is defined by the tuple $(\mathcal{S}_0, \mathcal{A}_0, \mathcal{O}_0, r_0, \rho_0, \gamma_0, T_0) \times \cdots \times (\mathcal{S}_n, \mathcal{A}_n, \mathcal{O}_n, r_n, \rho_n, \gamma_n, T_n) \times \times P \times \mathcal{Z}$, where n is the number of agents, $\mathcal{S}_i$ is a (possibly infinite) set of states for agent i, $\mathcal{A}_i$ is a set of actions for agent i, $\mathcal{Z} : (\mathcal{S}_0 \times \mathcal{A}_0) \times \cdots \times (\mathcal{S}_n \times \mathcal{A}_n) \rightarrow (\mathcal{O}_0, \ldots, \mathcal{O}_n)$ is a function describing how the world state is mapped into the observations of the POMDP, $P : (\mathcal{S}_0 \times \mathcal{A}_0 \times \mathcal{S}_0) \times \cdots \times (\mathcal{S}_n \times \mathcal{A}_n \times \mathcal{S}_n) \rightarrow \mathbb{R}_{\geq 0}$ is the transition probability distribution for moving from one set of agent states s to the next set of states s' given the set of actions $(a_0, \ldots, a_n)$, $r_i : (\mathcal{S}_0 \times \mathcal{A}_0) \times \cdots \times (\mathcal{S}_n \times \mathcal{A}_n) \rightarrow \mathbb{R}$ is the reward function for agent i, $\rho_i : \mathcal{S}_i \rightarrow \mathbb{R}_{\geq 0}$ is the initial state distribution for agent i, $\gamma_i \in (0, 1]$ is the discount factor for agent i, and T_i is the horizon for agent i.

The goal for a given agent i is to find a controller π_i that optimizes

$$J^{\pi_i} = \mathbb{E}_{\rho_0,\ p(s_{t+1}|s_t, a_t)} \left[\sum_{t=0}^{T} \gamma^t r_t \mid \pi(a_t|s_t) \right]$$

where r_t^i is the reward of agent i at time t and the expectation is over the start state distribution, the probabilistic dynamics, and the probabilistic controller π.

5.5 Controller design

Optimization criterion

Our goal is to reduce the average energy consumption of the system. However, the energy minimizing solution is for all vehicles to come to a full stop. To avoid this degenerate solution, we will impose the constraint that all vehicles exit the system. In this section, we describe how this constraint is converted into a reward function so that our desired optimized quantity can be used in a standard reinforcement learning procedure.

Let L be the length of the controlled portion of the network and $E(v_t, a_t)$ the instantaneous energy consumption at time-step t, v_t and a_t being the velocity and acceleration respectively. For notation simplicity, we will only consider the trajectory of one AV, as the reward for each AV is computed independently of the others, and we assume that the trajectory starts at time $t = 0$ and ends at time $t = H$.

Ideally, we would like to maximize the cumulative miles per gallon value for each AV

$$\frac{L}{\sum_{t=0}^{H} E\left(v_t, a_t\right)} \tag{5.3}$$

Unfortunately, that quantity cannot be computed until the end of a trajectory, making the reward sparse. Sparse rewards are generally difficult to optimize so we propose a simple heuristic that approximates this quantity.

We attempt to turn the sparse cumulative miles per gallon reward into a per-step reward by noticing that since L, is a constant, maximizing Eq. 5.3 is equivalent to maximizing $\sum_{t=0}^{H} -E(v_t, a_t)$ as long as energy consumption is positive. We can thus give the agent a reward $r(s_t, a_t) = -E(v_t, a_t)$ at time-step t.

However, the issue that the optimum consists in coming to a full stop will still persist here. Amongst options considered, we observed that giving the agent a semi-sparse reward for making forwards progress achieved the largest improvement in fuel efficiency.

$$r(s_t, a_t) = \begin{cases} -E(v_t, a_t) & \text{if } c_t < M \\ -E(v_t, a_t) + B & \text{if } c_t \geq M \end{cases} \tag{5.4}$$

Here c_t is a counter of the total distance that we have travelled since receiving the last bonus and B is a bonus for completing M meters. c_t is reset back to zero every M meters. Essentially, every time the vehicle completes M meters, it receives a bonus for doing so. We can think of this as approximately distributing a penalty for failing to exit the network across the spatial extent of the network but we note that the exact equivalence to the cumulative miles per gallon objective (Eq. 5.3) is now lost.

Finally, since the goal is still to optimize the energy consumption for the whole system, we also add the energy consumption of the N vehicles following the AV to its reward function, which are the vehicles that it has the most impact on.

$$r(s_t, a_t) = -E(v_t^0, a_t^0) - \sum_{i=1}^{N} E(v_t^i, a_t^i) + B_t \tag{5.5}$$

with

$$B_t = \begin{cases} 0 & \text{if } c_t < M \\ B & \text{if } c_t >= M \end{cases}$$

where we index the velocities and accelerations of the vehicles, 0 being the AV, 1 the vehicle following it, and N its n^{th} follower. Although this requires non-local information at training time, the reward is not part of the controller state and thus the controller will still only rely on local information.

Dec-POMDP design

We focused on picking controller inputs that could tractably and easily be placed onto a vehicle equipped with standard level-2 technology such as forwards facing radar, cameras, and GPS. The careful design of the state space is essential as the state space choice will have strong consequences for the generalization capabilities of the agents. As an example, consider an agent that has GPS coordinates as part of its input. This agent now has two potential generalization failure modes: 1) it may use the GPS position to block the network entrance and artificially reduce the inflow 2) it will adjust its behavior to perfectly optimize the particular network architecture that the agent is trained in and may be less likely work for different road network architectures.

Based on the criteria of maximizing likely generalization, we adopt the following Dec-POMDP:

- State space / Observation function: $[v, h, v_{\text{lead}}, c]$ where v is the ego speed, h is the distance to the leader, v_{lead} is the speed of the vehicle directly in front of the AV, and c is the distance travelled which is reset every m meters. This state space can be used in arbitrary networks and allows us to easily transfer learnt controllers between different network architectures. It is also easily implemented with radar and cameras.
- Action space: accelerations bounded between $[-2.6, 4.5]$. We do not allow the AVs to lane change.
- The reward function is described in Sec. 5.5.

Algorithm / Controller

As our training algorithm, we use Independent Proximal Policy Optimization [124], a ubiquitous policy gradient algorithm. All agents are homogeneous, that is, there is one controller that is duplicated across all agents although actions are still computed locally. The controller is a two layer fully connected neural network with 64 hidden units at each layer and a hyperbolic tangent non-linearity.

We make one small modification to the standard PPO algorithm and provide the total distance traveled by the agent at time t as an input to the value function. The value function is used exclusively during training for variance reduction (see [124] for details) and so non-local information can be used. The value V^π function estimates the reward-to-go from a given state s_t

$$V^\pi(s_t) = \mathbb{E}\left[\sum_{j=t}^{T} \gamma^i r(s_j, a_j) | s_j\right] \tag{5.6}$$

and since the reward-to-go strongly depends on the total distance remaining to the exit, it is difficult to estimate without this information.

Experimental setup

We ran the reinforcement learning training using the PPO implementation provided in RLlib [84][1] version 2.0.0.dev0, a distributed deep RL library. We use a learning rate of $3 \cdot 10^{-4}$, a training batch size and SGD minibatch size both equal to 500000, a number of SGD iterations of 5 and run the simulations for 220 iterations, each with 19 workers running in parallel with a horizon of 500 environment steps. Importantly, we set `multiagent/count_steps_by` to `agent_steps` so that steps are counted by agent step and not environment step. We also set `batch_mode` to `complete_episodes`, `gamma` to 0.995, `lambda` to 0.97 and `kl_target` to 0.02. The other RLlib and PPO parameters are left to their default value. Training for 220 iterations took about 2 days, running on a machine with 20 Intel Xeon E5-2670 v2 CPUs. The evolution of the reward function during training can be seen in Fig. 5.4.

Both the controller (policy network) and the value function network are feedforward neural networks (MLP) with two fully-connected hidden layers of size 64, and tanh activations. For our reward function, we used parameters $M = 50$m, $N = 5$ vehicles and $B = 2.5$.

We use the traffic micro-simulator SUMO [88] for running our simulations. To populate the simulation fully with vehicles, we allow a warmup period of 720 seconds during which the experiment runs uncontrolled after which 10% of the vehicles are turned into AVs. We keep a fixed inflow of 2050 vehicles per hour over the whole horizon. 90% of these vehicles are humans with an IDM controller, and the remaining 10% are AVs. The downstream speed limit is fixed to 5m/s. The IDM controller is used with parameters $a = 1.3$, $b = 2$, $v_0 = 30$, $T = 1$, $\delta = 4$ and $s_0 = 2$. Finally, taking note that the standard benchmark for ATARI games repeats each action four times [13], agent actions are actually sampled once for every 3 time-steps and the same action is applied for all 3 time-steps. We use an individual time step size of $0.4s$. This means that a horizon of 500 environment steps will run for 10 simulated minutes.

5.6 Results

Evaluation procedure

In this section we evaluate the performance of our trained controller and perform sweeps around its training distribution in order to assess its generalization capabilities. For each sweep value, we run from 30 to 60 simulations using that sweep value and compute the mean and standard deviation of the results obtained during each rollout, both of which are shown in the plots below. Metrics are computed from averaging data collected over all vehicles (or AVs) post warm-up time, except those located in the ghost cells (see Fig. 5.1). For time-space diagrams, we use the 2nd lane from the left as the plotted lane; this lane runs through the whole network.

[1]`https://github.com/ray-project/ray/python/ray/rllib`

We benchmark our RL controller against the FollowerStopper (FS) [132], a control algorithm that achieved wave smoothing in a physical experiment. FS aims to drive at exactly a desired velocity v_{des} whenever safe (*i.e.*, as in a standard cruise controller), but will command a suitable lower velocity $v_{\text{cmd}} < v_{\text{des}}$ whenever safety requires. Importantly, it attempts to smoothly transition between those objectives. We keep the desired speed constant over each simulation, and use the same hyperparameters for other portions of the controller as in [132].

We compare the results obtained by our RL controller to the uncontrolled human baseline where all vehicles are IDM, to the FS controller (with different desired speeds v_{des}), and to a variant of the FS that has v_{des} set to the downstream speed. We refer to the latter as *cheating* FS as the downstream speed is non-local information that would not be available using on-board sensors; external infrastructure would be needed to observe the downstream speed.

Controller performance and robustness

Figure 5.5 shows the effect of the introduction of the RL controller on the time-space diagram of the system. Without control, at speeds of both 3 and 5 m/s, waves are visible as dark lines slopping from top-left to bottom-right. When RL control is introduced, the waves become markedly lessin number and occasionally completely dissipate. Gaps formed by the RL agents can be seen as white lines sloping from bottom-left to top-right. These gaps terminate near the boundaries of waves as they dissipate the wave and are consumed in the process of doing so.

Figure 5.6 examines the effects of the wave reduction on the average fuel efficiency (in Miles per Gallon i.e. mpg) of the system. For a fair comparison, we sweep the desired velocity of the FS controller over all possible values of the downstream speed. As can be observed, the RL controller improves markedly on the fuel efficiency of the system and achieves the best performance of each of the FS controllers up until 7 m/s. Essentially, up until 7 m/s, the RL controller acts almost as effectively as a controller that *knows what the downstream speed is.* Figure 5.6 shows these same results as a percentage improvement over the uncontrolled baseline. Here the cheating FS is added as an additional baseline, showing the close match in performance between the RL and the non-local controller.

Finally, we investigate potential robustness issues with our controller. Figure 5.8 provides a sanity check that the improvement in fuel consumption does not come at the cost of reduced outflow up until 7 m/s. As this reduction in outflow is potentially undesirable, the controller could be switched off around this boundary. Additionally, in Figure 5.9, we investigate the effects of changing penetration rate on the controller. Our controller is trained at a fixed penetration of 10%, shown as the yellow line in the Figure. Since at any time, randomness could cause the penetration rate to vary from this value, it is important that controller performance be preserved away from the training regime. The RL controller performance, shown in th red, indicates that performance improvements are maintained outside of the training distribution, with values close to 10% performing almost identically.

Additionally, there is generalization outside of this value and energy improvements are seen at all penetrations.

Behavior analysis

In this section, we provide both qualitative and quantitative analysis to explain the energy improvements induced by the RL controller. In Figure 5.10, we plot the acceleration profile of the different controllers as a function of the speed of the lead vehicle and the space gap i.e. distance to the lead car. Since the acceleration profile is 3-dimensional, we show slices of the acceleration at 3, 5, and 7 m/s for the speed of the controller car. Note that these acceleration profiles are the output after post-processing of the desired output with the safety controller discussed in Sec. 5.4. As can be observed in the lower-half of the plots, the RL controller has a wide region where it accelerates at an almost fixed acceleration rate, and a vanishingly small region where it brakes. The RL controller is slowly accelerating at a fixed rate, with the magnitude of positive acceleration decreasing as the AV speed passes from 3 to 7 m/s. Above 7 m/s, the RL controller only brakes, which explains the reduction in outflow at downstream speeds above 7 m/s observed in Figure 5.8. Essentially, the RL controller is accelerating most of the time and then relying on the safety controller to brake sharply at the appropriate moment.

Finally, Figure 5.11, examines the net acceleration of the controllers. As can be seen, the RL controller has a consistently higher amount of acceleration than the cheating FS but is able to outperform it in MPG at both 5 and 6 m/s despite the higher accelerations at those values. This is possible due to an asymmetry in the energy function; braking incurs zero energy cost while the energy cost increases super-linearly with increasing acceleration. By maintaining low accelerations and braking sharply at the last possible second, the RL controller is able to reduce energy expenditure while maintaining reasonable speeds.

5.7 Conclusions and Future Work

In this work we set forth a challenging new network for phantom jam smoothing and demonstrate that multi-agent reinforcement learning could be used to design effective controllers for optimizing the network. We find that controllers designed in this way are remarkably robust and, despite having no memory with which to perform system-identification, have the same efficacy as controllers that know the system equilibrium across a wide range of potential wave-inducing conditions. We qualitatively analyze the characteristics of these controllers relative to a standard baseline and additionally demonstrate that our controller functions effectively across varied penetration rates. Future work will investigate how well these controllers transfer to new networks as well as their robustness to a larger range of potential human driving dynamics.

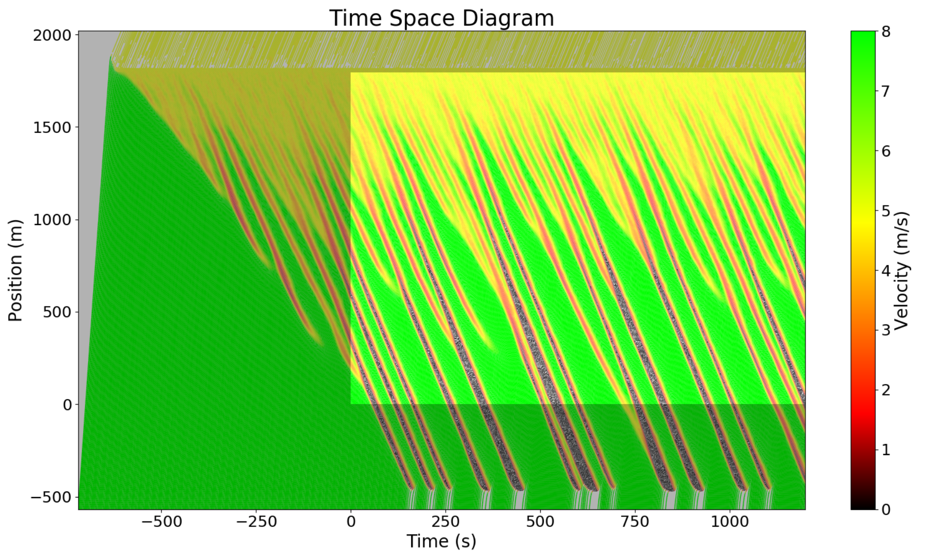


Figure 5.2: Time-space diagram of one lane of the I-210, showing the average velocity of the network as a function of time and position. The shaded areas correspond to the warm-up period and the ghost cells, and represent times and positions that are not considered in control or evaluation. Waves are visible as the downwards-sloping black lines.

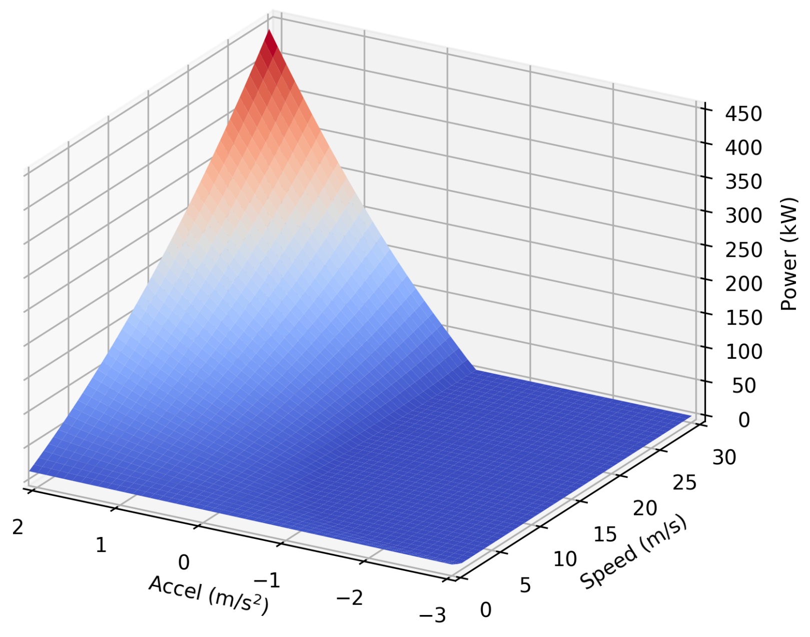


Figure 5.3: Polynomial fit of power consumption as a function of velocity and instantaenous acceleration from a Midsize Sedan model provided by Toyota. This model was made for a vehicle of mass 1743kg, and we assume a constant road grade of 0. We use the conversion 1 gallon/hour = 33430 watt.

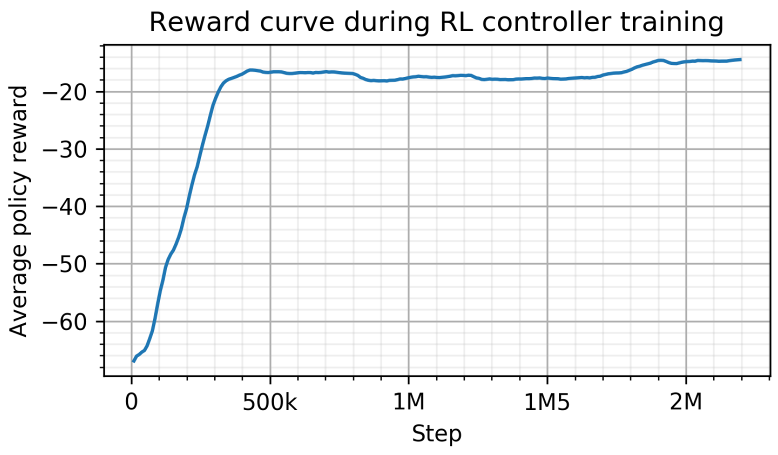


Figure 5.4: Evolution of the reward for an agent over two days of training; the reward depicted here is averaged across all the agents. The policy is close to converged approximately 25% of the way into training. Note that this represents the average sum of all rewards received by an agent, and not the discounted return.

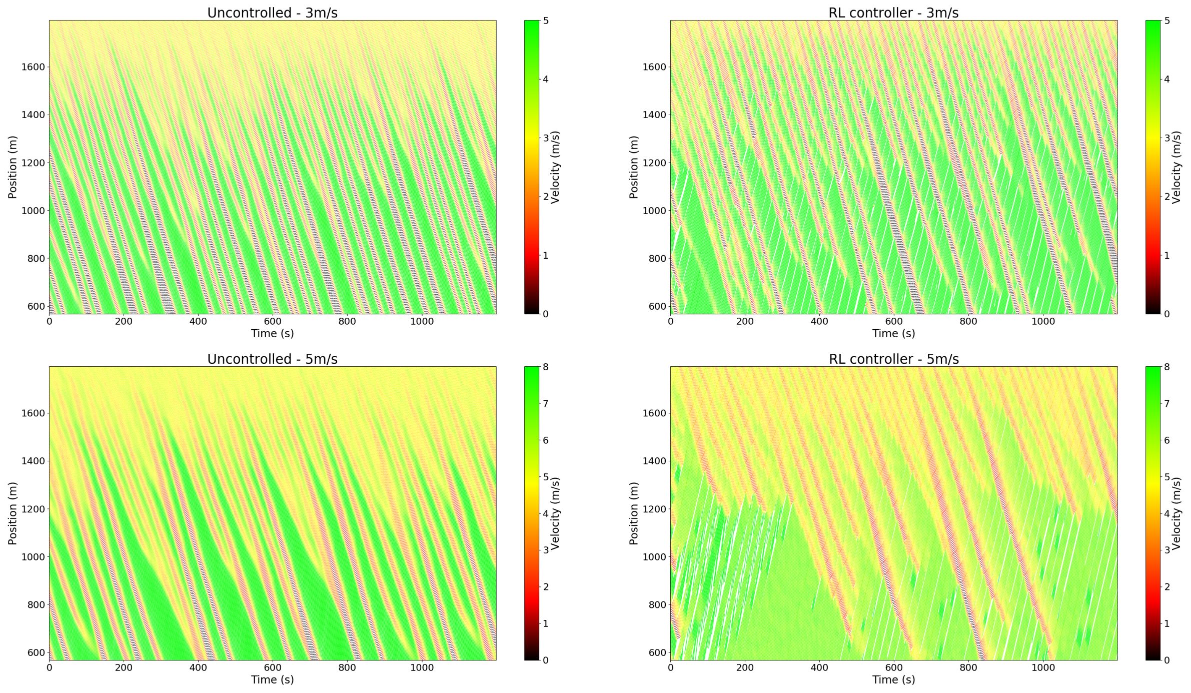


Figure 5.5: **Left:** time-space diagram of the I-210 simulation with no control applied, for a downstream speed of 3m/s (top) and 5m/s (bottom). **Right:** time-space diagram of the I-210 simulation when 10% of vehicles are AVs using our RL controller, for a downstream speed of 3m/s (top) and 5m/s (bottom). Time-space diagrams show the (average) vehicles velocities as a function of their position on the highway and simulation time. Ghost edges and warm-up time are not shown in these graphs.

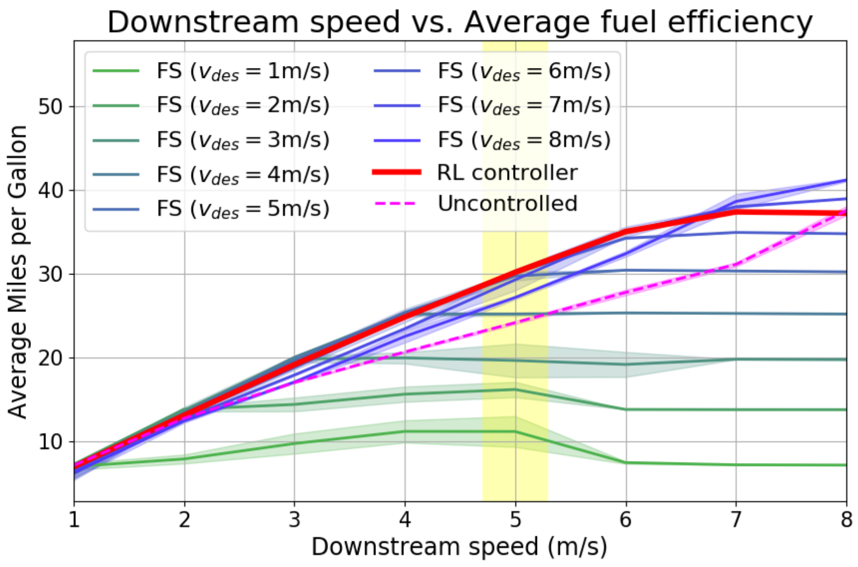


Figure 5.6: Average fuel efficiency of the RL controller on its training downstream speed of 5m/s (highlighted in yellow), and generalization to speeds outside that range. Miles per gallon fuel consumption is also shown for the FS controller with desired speed ranging from 1m/s to 8m/s and for the uncontrolled human baseline, as a function of the downstream speed. All plots are computed using a fixed penetration rate of 10% and using the energy model presented in Sec. 5.4.

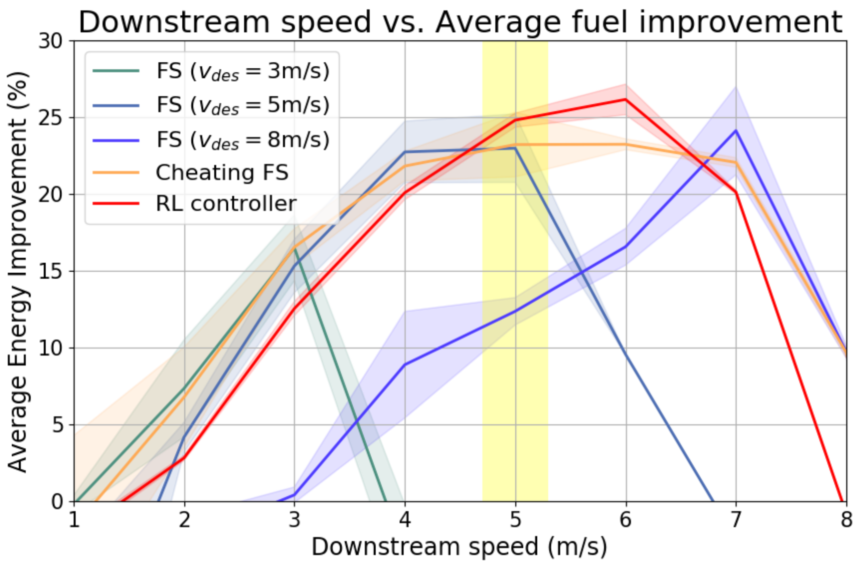


Figure 5.7: Fuel efficiency improvement of the RL controller on its training downstream speed of 5m/s (highlighted in yellow) over the uncontrolled human baseline, and generalization to speeds outside that range. Fuel improvement is also shown for the FS controller with a desired speed of 3m/s, 5m/s and 8m/s as well as for the cheating FS. All plots are computed using a fixed penetration rate of 10% and using the energy model presented in Sec. 5.4.

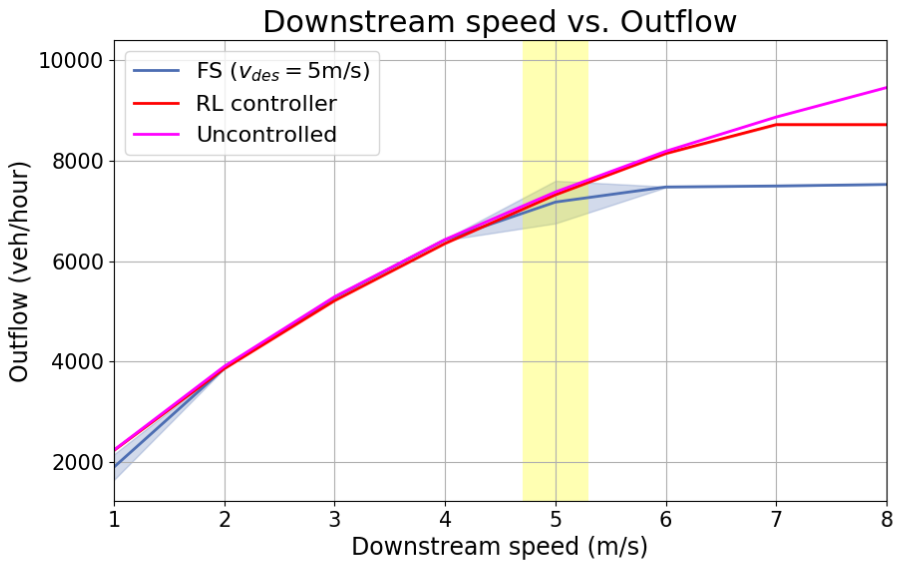


Figure 5.8: System outflow as a function of the downstream speed, shown for the RL controller, the FS controller with desired speed equal to 5m/s, and the uncontrolled human baseline. The yellow area highlights the downstream speed which the RL controller was trained on, outside of which it is acting in complete generalization. All plots are computed using a fixed penetration rate of 10%.

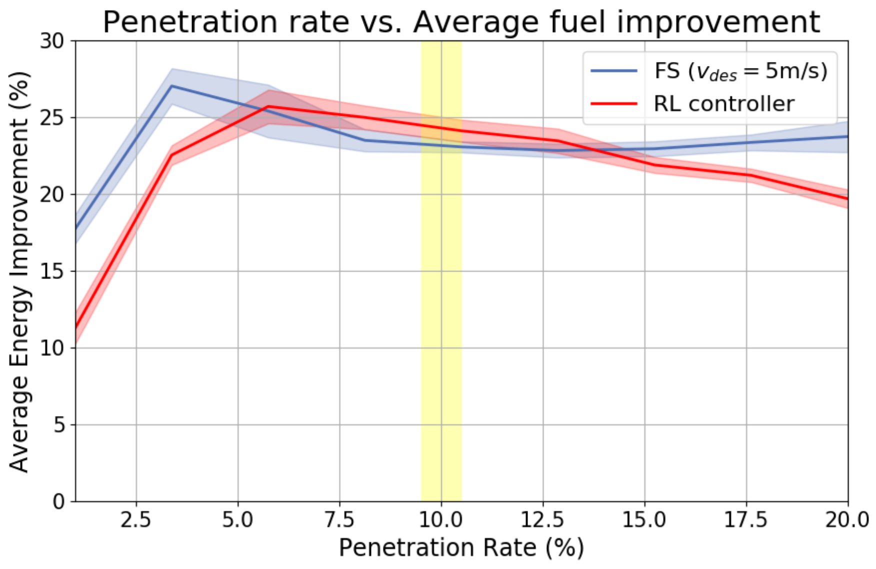


Figure 5.9: Fuel efficiency improvement of the RL controller on its training penetration rate of 10% (highlighted in yellow) over the uncontrolled human baseline, and generalization to penetration rates outside that range. Fuel improvement is also shown for the FS controller with a desired speed of 5m/s. Both plots are computed using a fixed downstream speed of 5m/s and using the energy model presented in Sec. 5.4.

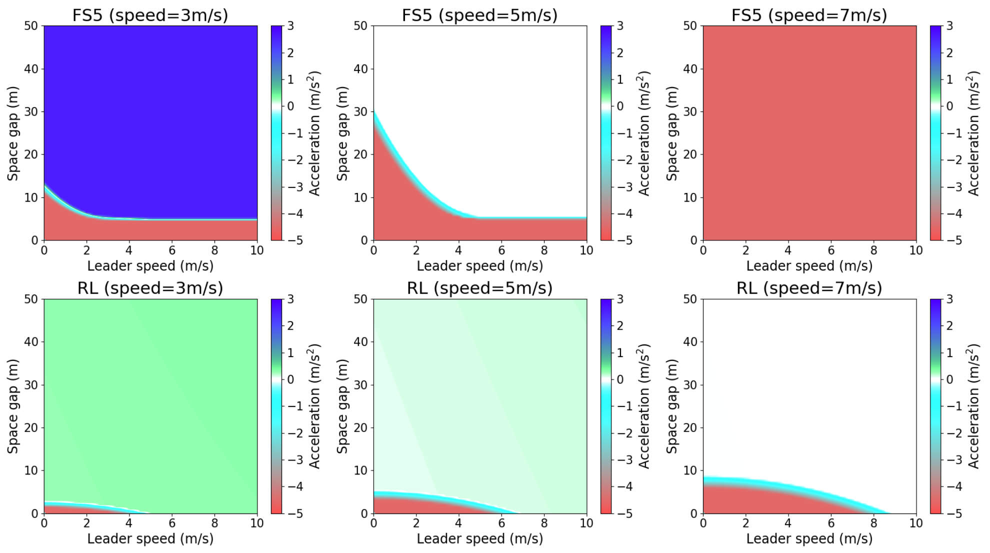


Figure 5.10: This figure demonstrates the difference in acceleration profile between our RL controller (bottom) and the FS controller set with a desired speed of 5m/s (top). The instantaneous acceleration output of both controllers is plotted as a function of the AV speed (left: 3m/s; middle: 5m/s; right: 7m/s), the leader speed and the space gap to the leader.

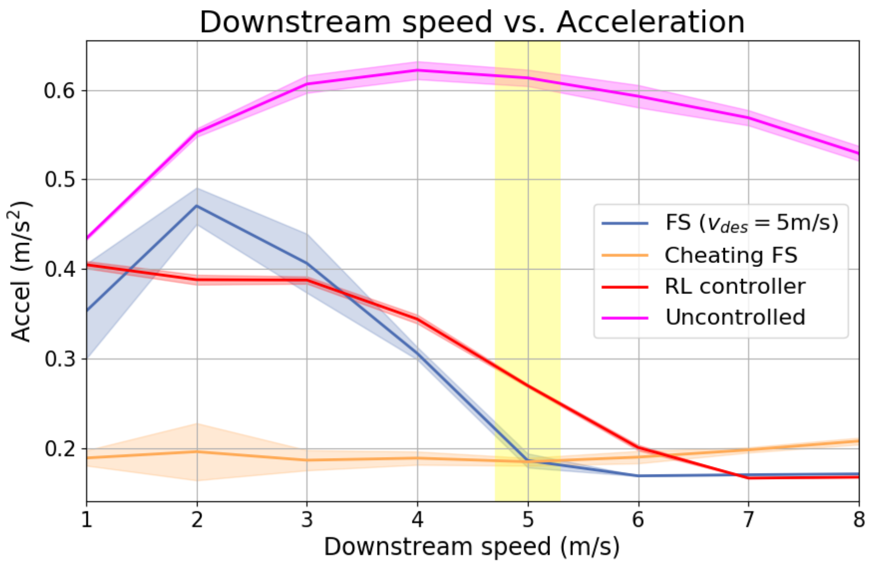


Figure 5.11: Average AV acceleration (absolute value of the instantaneous acceleration) using the RL controller at its training downstream speed of 5m/s (highlighted in yellow) and generalization to speeds outside that range. It is also plotted for the uncontrolled human baseline, the FS controller with a desired speed of 5m/s, and the cheating FS controller. All plots are computed using a fixed penetration rate of 10%.

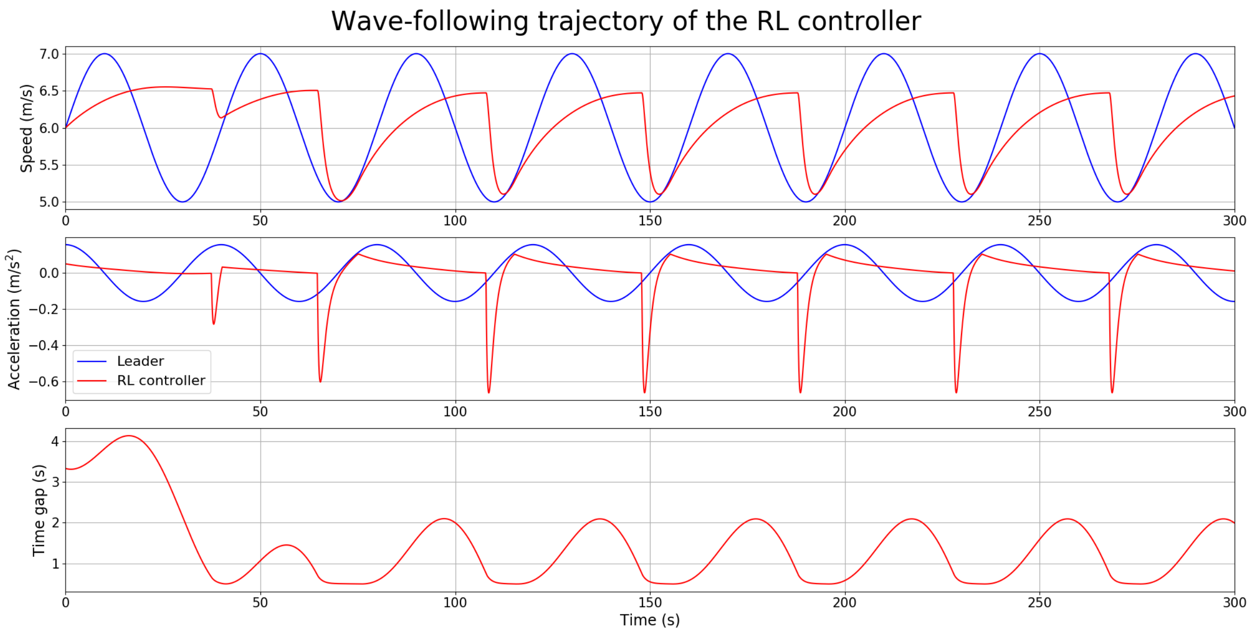


Figure 5.12: Acceleration, velocity and time gap profiles of an AV using the RL controller, following a leader trajectory that has a sinusoidal velocity centered around 6m/s with an amplitude of 1m/s and a period of 40s, whose velocity and acceleration profiles are also plotted.

Chapter 6

Nocturne: a scalable driving benchmark for bringing multi-agent learning one step closer to the real world

6.1 Introduction

This chapter presents *Nocturne*, a new simulator and benchmark for multi-agent driving under human-like sensor uncertainty that is intended to aid the process of studying real-world multi-agent coordination and learning. Instead of combining the challenges of coordination with feature extraction from images, Nocturne is a 2D simulator that generates vector representations of the set of objects and road points that would be visible to an idealized human driver (see Fig. 6.1 for an example) and supports head-tilt to acquire additional information about blind spots. In contrast to driving benchmarks that achieve partial-observability by using a camera input, Nocturne uses efficient visibility-checking methods and a C++ backend to construct observations and step the dynamics of a single agent at 2000 to 4000 steps per second. This speed is key to its use in multi-agent learning settings where frequently billions of environment interactions are needed to learn expert agents [12, 79]. In contrast to many existing multi-agent learning benchmarks, Nocturne is neither zero-sum nor fully-cooperative but mixed-motive, combining the challenges of coordination and cooperation.

Crucially, Nocturne is not just a simulator. Instead, it is built upon open-source driving data and features a diverse set of real-world scenes (see Fig. 6.2) that probe the ability of agents to safely navigate and coordinate in complex scenes such as intersections, roundabouts, parking lots, and highways. We use this data as a source of experts for imitation, to flexibly vary the number of controlled agents in the scene, or as a train-test split for validating the generalization ability of a human-driver model.

The challenge we propose is to learn (or otherwise design) policies that achieve the same

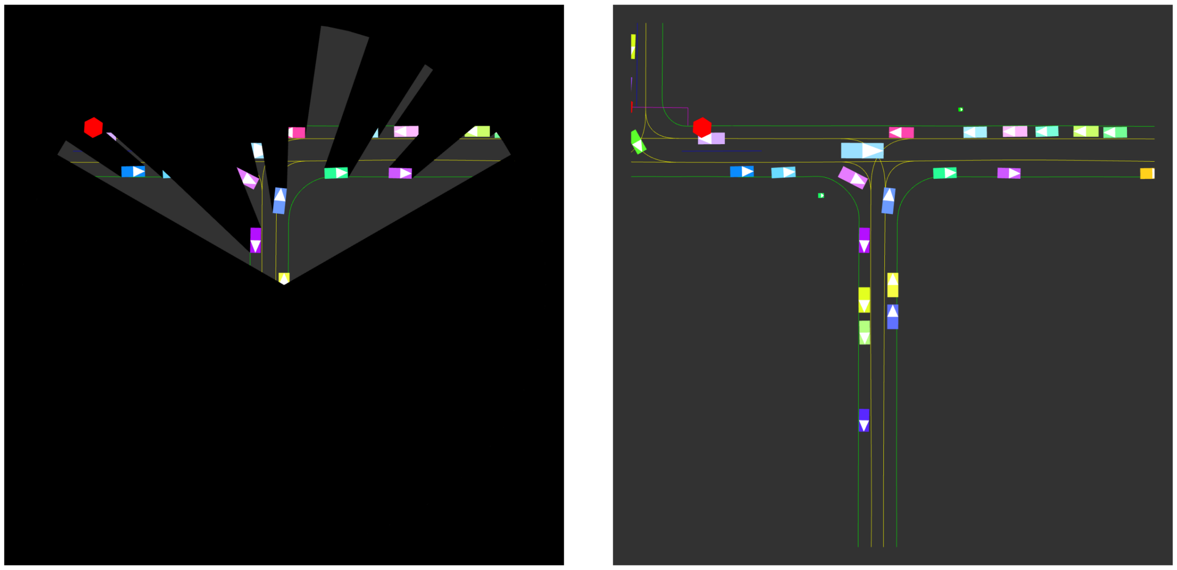


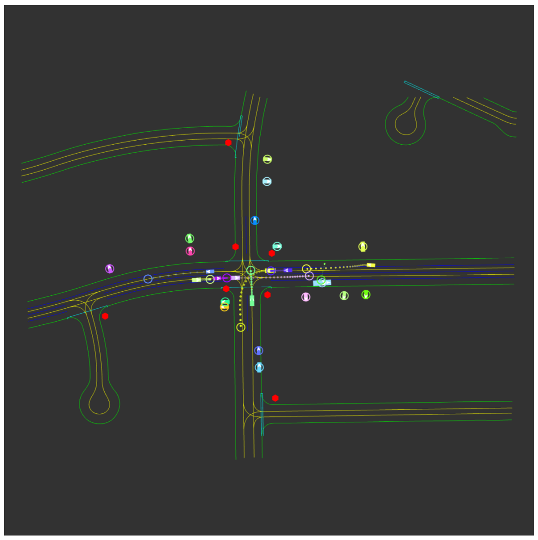
Figure 6.1: A visual depiction of the obstruction model used to represent objects that are visible to the agents. (Left) Obstructed view with the viewing yellow agent in the center of the cone. (Right) Original scene centered on the cone agent with an unobstructed view.

set of final states as human experts (throughout this work we will call this the *goal rate* and the final state the *goal position*) while achieving a 0% collision rate. Achieving a high goal rate alongside a negligible collision rate is challenging for any policy design scheme (both learning and non-learning) due to the combination of the high-dimensional state space, the partial observability of the scene, the large number of interacting agents, and the decentralization of the policies at test time. The secondary challenge in Nocturne is to find policies that closely mimic human behavior at the trajectory level. To facilitate these goals, on top of the human data we provide an evaluation scheme and off-the-shelf baseline implementations including evaluations.

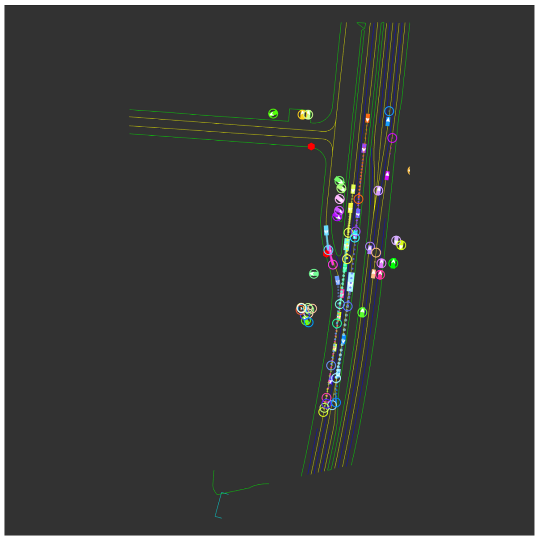
We provide a guide to the benchmark as well as some preliminary work on using Nocturne to design agents and test their capabilities and human-similarity. We demonstrate that learning effective agents in Nocturne is challenging; tuned RL and imitation learning baselines struggle to successfully complete the highly interactive scenes. Finally, we demonstrate that the agents achieve relatively low distance to the expert trajectories and that there does not appear to be a zero-shot coordination problem [58] at this level of agent capability.

6.2 Related Work

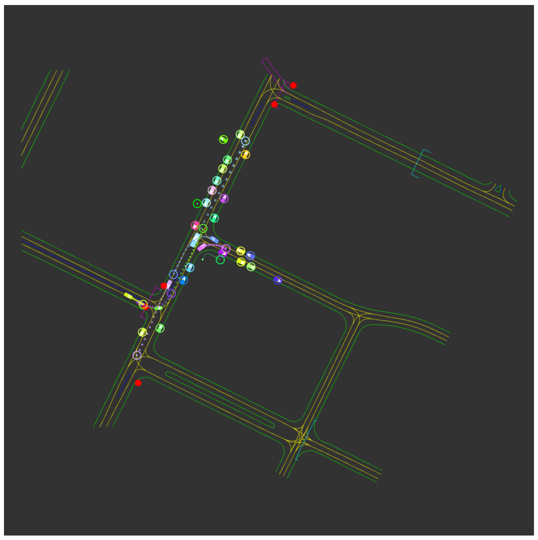
Multi-agent traffic simulation tools and benchmarks:
In terms of multi-agent driving benchmarks that require both acceleration and steering



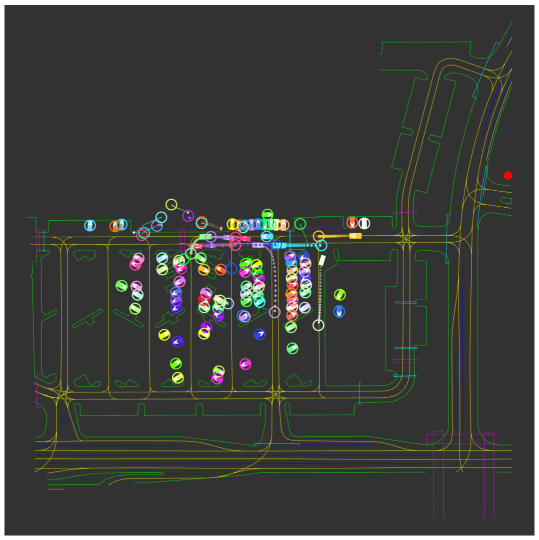
(a) A four-way stop.



(b) A straight road with merging vehicles.



(c) Unprotected turns with conflicting routes.



(d) A crowded parking lot.

Figure 6.2: Four scenes demonstrating the diversity of the navigable scenes in Nocturne. Colored circles represent the goal position of the corresponding colored agent. Dots represent the trajectory of the agent, with opacity increasing as time goes on. Links to videos of experts negotiating these scenes can be found at nathanlct.com/research/nocturne.

Table 6.1: Comparison of representative driving simulators. State-based partial observability refers to whether the set of visible objects can be queried. Expert data refers to whether expert human data is available for the scenes. Baseline human models refers to whether pre-existing human driver models are available to drive agents in the scene.

Simulator	Multi-agent Support	2D/3D	State-Based Partial Observability	Expert Data	Baseline Human Models
CARLA [38]		3D			✓
SUMMIT [24]	✓	3D			✓
MACAD [105]	✓	3D			✓
Highway-env [80]		2D			✓
Sim4CV [97]		3D			
Duckietown [107]		3D			
SMARTS [167]	✓	2D			✓
MADRaS [120]	✓	2D			✓
DriverGym [72]		2D		✓	✓
DeepDrive-Zero [111]	✓	2D			
MetaDrive [81]	✓	3D		✓	✓
VISTA [81]	✓	3D		✓	✓
Nocturne	✓	2D	✓	✓	

control, there are several closely related benchmarks which differ in terms of ease of implementing multi-agent interaction, being data-driven, 2D vs. 3D, or the mechanism by which they support partial observability. We summarize these differentiating features in Table 6.1.

The closest works to ours are BARK [17], SMARTS [167], and MetaDrive [81]. SMARTS [167] supports multi-agent driving in a wide variety of interactive driving scenarios and offers a wide array of default human driving behaviors to use for testing autonomous vehicles. BARK [17], like Nocturne, is a 2D goal-driven simulator with support for external datasets and contains a wide variety of large-scale scenarios and multi-agent support. Finally, MetaDrive [81] also supports real-world data and multi-agent interaction and is able to achieve 300 FPS image rendering by rendering lower fidelity images. The main differentiating feature from these works is the support for acquiring the set of objects that are visible without requiring the rendering of camera images; to our knowledge Nocturne is the only available simulator that can compute an agent's visible objects and step the agents dynamics at above 2000+ steps-per-second. Additional simulators are summarized in Table 6.1.

Partially observed multi-agent benchmarks:
There are a wide variety of partially-observable multi-agent benchmarks not focused on driving that differ from our work along the axes of underlying game structure, level of partial

observability, and accessibility of expert data. Within the card-playing domain, Hanabi [12] is frequently used to investigate coordination under partial observability, is fully coopeartive, and features between 2-5 agents. The Starcraft multi-agent benchmark (SMAC) [119], perhaps the most ubiquitous MARL benchmark, features many agents and high-dimensional observations but does not come with human data, is mostly fully-observed, and many algorithms now achieve perfect performance in this challenge [161]. Melting Pot [79] features a huge diversity of many-agent mixed-motive challenges but does not have available expert data. Google Research Football [76] provides a few zero-sum multi-agent environments featuring two teams consisting of several agents.

6.3 Benchmark construction

Defining a Nocturne Scene

In the following sections, we will refer to objects that can move (vehicles, pedestrians, cyclists) as *road objects* and anything that cannot move (lane lines, road edges, stop signs, etc.) as *road points*. *Road points* are connected together to form a polyline. We will refer to the type of road polyline that should not be crossed by vehicles as a **road edge**.

Nocturne scenes require a map consisting of polylines, a set of initial and final road object positions, and optionally a set of trajectories for the road objects. Nocturne currently acquires its scenes, goals, and expert trajectories from the Waymo Motion Dataset [44] but can be configured to support any dataset that represents its road features as points or polylines and that contains start and end position coordinates for any road objects. The Waymo Motion dataset consists of 487004 nine-second trajectory snippets discretized at a rate of 0.1 Hz with the first second intended to be used as context and the latter eight seconds to be used for prediction.

One challenge of selecting and constructing the scenarios for Nocturne is that these trajectories are collected by labeling the vehicles observed by a Waymo car as it drives. Consequently, we do not have a complete birds-eye view of the scene. Hence, there are cars that may have been in the scene that were not visible to the Waymo vehicle and therefore are not included in the dataset; for the same reason, the expert trajectories are also incomplete and may not persist throughout the entire duration of the rollout. In other words, the expert trajectories contain agents that unpredictably flicker in and out of existence. Similarly, the set of traffic lights that were visible to the Waymo vehicle may be insufficient to uniquely determine the underlying traffic light state. For this reason, this first version of the Nocturne benchmark comes with a few restrictions: we do not use traffic lights due to the incomplete state and for the posed challenges we do not include pedestrians or cyclists as replaying the expert trajectories may cause them to unpredictably appear on top of a vehicle trajectory and cause an unavoidable collision. We also filter out scenes that contain traffic lights which leaves the majority of the remaining scenes as roundabouts, unsignalized intersections, arterial roads, and parking lots. This leaves the benchmark consisting of 134453 snippets.

Partial Observability Model and Collision Handling

To support investigation into coordination under partial observability, agents in Nocturne come with a configurable view-cone as is shown in Fig. 6.1. An element (*road object* or *road point*) is considered observable if there is a single ray in the cone that intersects with that element that does not pass through any *road object* on its path to the element. Stop signs are always visible if they are within the agent view-cone even if they would be otherwise obscured on the assumption that they would be raised at a sufficiently high level. We select the angle of the cone to be 120 degrees (approximately the range of binocular vision) and 80 meters radius. We note that this does not perfectly mimic a human visual model which has additional complexities such as dynamic variations in the functional field of view [34], phenomena relating to interactions between visible objects such as crowding [20], and the occasionally necessary ability to pay attention to objects further than 80 meters away at high speeds.

The primary challenge in computing which road points and objects are observable is their relatively large number: a scene contains 30 vehicles and 4700 road edge points on average. We use a Bounding Volume Hierarchy (BVH) to maintain the road objects and select candidates for potentially observed vehicles. We build the BVH using approximate agglomerative clustering [51] to generate a high-quality BVH. When computing the observed objects, we first use the BVH to select the candidates that lie in the *axis-aligned bounding box* (AABB) of the conic view field. Since there are a comparatively larger number of all types of road points (order of 15000 on average), we use a 2D range tree [15] to maintain all of the road points. When computing the observed road points, we do a range search in the 2D range tree to select the candidates that lie in the AABB of the conic view field. For both vehicles and road points, once we have the candidates in the conic view-field, we perform a brute-force visibility check for the object and road points respectively by ray-casting from the viewing agent to all the candidate vertices; an object or road point is visible if there is a ray reaching any point of it that does not pass through any *road object*. We accelerate this procedure using the auto-vectorization property of compilers which gives a 500% speed-up to this step.

We use a similar procedure to the visibility checking to accelerate our collision detection of vehicle-vehicle and vehicle-road collisions. For vehicle-road collisions, we consider a vehicle to have collided if its body intersects with any line segment of the polylines that constitute the road edges. A vehicle-vehicle collision occurs if the vehicle rectangles intersect. For both vehicle-vehicle and vehicle-road collision detection, we query the road object BVH and a separate road line segment BVH (formed once at the beginning of the episode) to generate candidates that are then evaluated for collision.

Construction of the Partially Observable Stochastic Game

Definition of the state space

Nocturne supports two possible state representations: a rasterized image and a vectorized representation of that image. While we provide default state representations, we consider it fair game on the benchmark to use any other representation as long as only objects that are visible under the conditions in Sec. 6.3 are presented to the agent. Conforming to these consistent rules about visibility allows for a fair comparison between different algorithmic approaches.

For the default vectorized representation, we adopt a fully descriptive set of features (speed, angle, width, length, etc.) for the road objects and use the VectorNet [49] representation for the road points. We will refer to the vehicle whose observation is being returned as the *ego vehicle*. Note that by default all features that can be placed into relative coordinates are returned in relative coordinates to the ego vehicle (e.g. speed is relative speed, heading is relative heading, etc). The agent goal is set to be the final position, speed, and heading of the expert agent. An agent is considered to have achieved its goal if it is within 1 meter of the final position, within 1 $\frac{\text{m}}{\text{s}}$ of the final speed of the agent, and .3 radians of the final heading.

The features of the ego object are:

- The speed of the object.
- The distance and angle to the goal.
- Its width and length.
- The relative speed and heading to the target speed and heading.

Road object features are:

- The speed of the object.
- The angle between the velocity vectors of the object and the ego vehicle.
- Its width and length.
- The angle and distance to the road object's position relative to the ego vehicle.
- Its heading relative to the ego vehicle.
- A one-hot vector indicating whether the object is a vehicle, pedestrian, or cyclist.

As per the VectorNet representation, each road point is part of a discretized poly-line with an approximate discretization size of 0.5 m (though cross-walks and speed-bumps are discretized with a much larger spacing). To ensure that any vehicle is able to discern which road points are connected to each other, we include a vector pointing to the neighbor of the point in the polyline. As with the road object representation, all points are in the frame of the observing vehicle. Road point features are:

- The angle and distance to the road point's position relative to the ego vehicle.
- A 2D element representing the vector pointing from the current road-point to its neighbor in the polyline.

– A one-hot vector indicating whether the road point is a lane-center, a road line, a road-edge that can be collided with, a stop-sign, a crosswalk, a speed-bump, or is of unknown type.

Since likely control architectures require the input vector to be of a fixed size, we select a fixed-size subset of visible road points and objects if there are too many and pad with a vector of 0.0 if there are too few. We sort both the road points, objects, and stop signs by distance and return the $500, 16, 4$ closest ones respectively where all three values are configurable by user. We note that this leads to a fairly high dimensional state and intentionally do not prune it to be smaller: dealing with this high dimensional input is a meaningful part of the benchmark. However, since we expect that users will likely want to construct their own state vectors, utility functions are also provided to allow users to directly query the set of of observed road points and objects.

Action Space

Vehicles are driven by acceleration and steering commands that are passed to a bicycle model to update the vehicle state. The angle at which the human driver views the scene is controlled by the agent *head-tilt*. In the experiments used in this paper we use 6 discrete actions for acceleration, 21 discrete actions for steering, and 5 discrete actions for head tilt with the acceleration actions uniformly splitting $[-3, 2]\ \frac{m}{s^2}$, the steering actions between $[-0.7, 0.7]$ radians, and the head tilt between $[-1.6, 1.6]$ radians. All these bounds on the actions are configurable.

Environment Dynamics and Goals

The total length of the expert data is 9 seconds, discretized into steps of size 0.1 seconds. For the first 1 second of the episode, all vehicles obey the expert policy. This is used to construct a history of observations for each agent that can be used to initialize or warm up the policy. After this transitory period, the episode continues for a fixed length of $T = 80$ steps. Agents are provided with a target position, speed, and heading that are taken from the final position, speed, and heading of the expert trajectories. If an agent achieves their goal within an environment specified tolerance they receive a reward of T and are removed from the system. A vehicle will also be removed if it collides with any road edge or object.

In addition, there is a process for selecting the set of vehicles that are controlled in the environment. First, we only control vehicles that at some point have a speed above $0.05\ \frac{\mathrm{m}}{\mathrm{s}}$ and that are more than $0.2\,\mathrm{m}$ from their goal; in general, these are vehicles that need to move to get to their goal. From this set of vehicles, we remove all vehicles that are already at their goal. Next, a small number of vehicles that have infeasible goals are also set to be experts (see Sec. 6.3 for how we compute this). Finally, of the remaining vehicles, we randomly select up to a maximum of 20 of them and set the remainder to replay expert trajectories. On average there are 10 vehicles in a scene so this generally leaves most vehicles as controlled.

Unusual features of the Benchmark

Finally, for completeness we note a few oddities of the benchmark that we believe are critical for potential solution designers to be aware of. These challenges are mostly properties of labeling noise and the egocentric view under which the data was collected.

Filtered out pedestrians, cyclists, and traffic lights

While the dataset contains scenes with traffic lights, pedestrians, and cyclists, the first set of Nocturne environments, NocturneV1, operates solely on scenes that have been filtered to not include traffic lights. Additionally, when constructing the environment, we remove all pedestrians and cyclists from the scene. Future versions of the benchmark will consider joint learning of pedestrians, cyclists, and vehicles but for NocturneV1 we only consider vehicles as appropriate modeling of pedestrian and cyclist dynamics is left for future work.

Egocentric data collection

The Waymo dataset that forms the basis of the first version of Nocturne is collected by driving a sensor-equipped car and recording the trajectories of all visible vehicles. Thus, in the original data, vehicles may have trajectories that are shorter than the full 9 seconds and may only appear midway through the trajectory or appear and disappear throughout the trajectory when they are obscured from the view of the sensing car. Rather than suddenly teleport cars into the scene midway through an episode, we choose to only use cars that were visible to the sensing car at the beginning of the episode (at the conclusion of the 1 second of expert replay described in Sec. 6.3). This reduces the total number of vehicles that might appear in the episode but does not change the feasibility of any of the agent goals.

Infeasible goals

Roughly 3% of the vehicles in the dataset have an expert trajectory that crosses an impassible road. This is due to labeling error in the dataset where, for example, small gaps in road edges are occasionally missed that make the road edge crossable. To ensure a benchmark where all goals are achievable, we compute all trajectories where crossing a road edge was necessary to achieve the goal and set these vehicles to replay their expert trajectory rather than be controlled; this corresponds to 3% of all vehicles. Finally, 2% of vehicles in the dataset are initialized in a colliding state. This primarily occurs in parking lot scenes where a vehicle slightly overlaps with the boundary of a parking spot or a nearby vehicle. These vehicles are also removed.

For computing the percentage of vehicles that must cross a road edge to get to their goal, we use the following procedure. We take the vehicles and shrink their width by 0.1 and their length by 0.3. These vehicles are very thin and so do not accidentally collide with a road edge due to small errors in their position. The scaling of the length ensures that the vehicles are not placed in a starting position where they have collided. Using this procedure we calculate the 3% statistic.

Illogical goals

Occasionally, agents may have goals that seem unlikely; for example, agents may be asked to come to a full stop in the middle of a highway. While these goals may seem odd, they are actual states that were achieved by humans driving on the roadways. These odd goals are likely a consequence of unobserved objects that were not observed by the egocentric data collection process. For example, a stop in the middle of an arterial road is likely caused by a queue of unobserved vehicles.

Rules of the Benchmark

We outline here a few rules that we expect solvers to respect to ensure consistency between solutions.

- The size of the view cone is fixed to 120 degrees and a distance of 80 meters. This ensures consistency between the level of partial-observability each controller must handle. Users can also tilt the head of the driver by 90 degrees in either direction to acquire more information.
- Only the first 1 second of the trajectory can be used as context or to warm-start a memory-based controller; control of the vehicles must start at 1 second into the trajectory.
- A trajectory that successfully reaches the goal is only considered valid if it reaches the goal within the 8 second time-window. All the scenes are easily solved without a time-constraint by simply creeping forwards slowly.
- The environment comes with default rewards and observations but any amount of reward-shaping or observation sharing at training time as valid. However, at test time only information that is directly observable to the agent can be used as input to the policy. For example, sharing information about other agents' goals would be valid at train time but not at test time.
- Adding additional map information to the agent state space beyond the information provided by default is valid.
- The bounds on the action space should respected: the acceleration should be bounded between $[-6, 6]\,\frac{\mathrm{m}}{\mathrm{s}^2}$, the change in heading should not be faster than 40 degrees per second, and driver head tilt should be maintained within 90 degrees in either direction. This rules out solutions that rapidly get to goal by using accelerations that are outside of possible vehicle speed bounds or excessively sharp turns.

We note that these rules may make some of the scenes unsolvable. For example, real human drivers are not randomly initialized into a scene with a maximum of 1 second of prior context; this context may be critical to safely navigating the scene under the time constraints. Additionally, our model of human perception is not an exact match for true human perception which can extend well beyond 80 meters under certain circumstances. It is possible that there may be missing context at this viewing distance that is crucial for safe navigation. We view these constraints as similar to the challenge of label noise in supervised datasets; our constraints may place an currently undetermined upper limit on the percentage of goals that can be achieved but the benchmark still constitutes a valid comparison between methods as long as the constraints are respected.

6.4 Experimental Results

We run several learning methods to demonstrate that these tasks do not appear to be easily solved even at billions of steps and have a train-test generalization gap. We also briefly investigate whether the policies appear to have a zero-shot coordination challenge wherein policies perform well when paired with agents from their seed but are incompatible with policies from a different training run. Finally, we examine the human-similarity of our trained agents.

We test the following methods:

- APPO [108] trained in multi-agent mode with a shared policy i.e. every agent is controlled by the same policy but in a decentralized fashion.
- Behavior Cloning [10, 121].

For the RL method, in addition to the fixed-bonus for achieving the goal, we add the following dense reward to encourage the agent to make progress towards the target position, speed, and heading:

$$r_t = 0.2 \times \left(1 - \frac{||x_t - x_g||_2}{||x_0 - x_g||_2}\right) + 0.2 \times \left(1 - \frac{||v_t - v_g||_2}{40}\right) + 0.2 \times \left(1 - \frac{f(h_t, h_g)}{2\pi}\right) \tag{6.1}$$

where x_t is the position at time-step t, x_g is the goal position, v_t is the speed at time-step t and v_g is the target speed, h_t and h_g are the current and target heading in world coordinates (i.e. not in a relative frame). $f(h_t, h_g)$ returns the minimum angle between h_t and h_g. Since coordinates are in a relative frame and the agent cannot observe the world frame, note that $f(h_t, h_g)$ is an observation provided to the agent. Note that there are values for the discount factor for which the optimal policy under the dense reward will hover near the goal and then only reach the goal at the terminal step instead of achieving it as soon as possible: the addition of the dense reward will affect the trajectory the agent will take. The use of this dense reward is not a necessary component of the benchmark and here is just used to generate good policies.

The APPO experiments are each run for two days on 1 V100 GPU and 10 CPUs (Intel Xeon CPU E5-2698 v4 @ 2.20GHz) and each experiment is run for 5 seeds. For the RL experiments we run 5 conditions over 5 seeds for 2 days leading to a total of 1200 GPU hours and 12000 CPU hours. For the Imitation Learning experiments we run 5 seeds for three hours leading to 15 GPU hours and 100 CPU hours.

The architecture for the APPO agent is a two-layer neural network with 256 hidden units and Tanh non-linearities followed by a Gated Recurrent Unit [28] with 256 hidden states. The Behavior Cloning experiment is run on a single GPU for five seeds, trained on 1000 files, for 600 gradient steps with a batch size of 512. Here instead of using a recurrent neural network we stack the current state and the prior 4 states as an input and pass this through a three-layer (1025, 256, 128) neural network that then outputs two categorical heads, one for acceleration and one for steering. The acceleration head is binned into 15 bins equally spaced between $[-6, 6]\,\frac{\mathrm{m}}{\mathrm{s}^2}$ and steering to 43 equally spaced bins between $[-0.7, -0.7]$ radians. As there is no corresponding head tilt in the expert actions we extend the angle of the visibility cone to π radians. Note that this violates a rule on the cone shape set forth in Sec. 6.3; figuring out how to add head tilt to imitation agents remains an open question.

For APPO we use almost all the default hyperparameters of SampleFactory [108] at commit aed6cc92a7eb3510c4d4bcfac083ced07b5222f9. However, we use a batch size of 7168 and scale the observations by 10.0. For Imitation Learning we use a learning rate of $3 \cdot 10^{-4}$, a batch size of 512, and stack a total of 5 states together to endow the agents with memory.

6.5 Analysis

Success rate of baselines

Fig. 6.3 shows the performance of the agents on the training set after $3e9$ steps which takes approximately two days. Each line corresponds to a policy trained on a fixed subset of the training scenes. We score our policies in two primary ways: the fraction of vehicles that achieve their goal (goal rate) and the fraction of vehicles that collide with another vehicle or road edge (collision rate). Note that this use of collision rate differs from its use in papers such as [59, 139] which compute collision rate as the fraction of scenes that contain a vehicle-vehicle collision and do not count a vehicle-road intersection as a collision.

We investigate the effect of the size of the dataset on the train and test performance. In the right half of Fig. 6.3 we can see that the inclusion of a larger training dataset decreases the performance of the algorithms up to 1000 files as they struggle with the diversity of the data. However, the inclusion of additional data narrows the magnitude of the train and test gap and closes it fully at 10000 files. However, it is still possible that a divergence between train and test might re-emerge as agents become more capable on the train set and approach a 100% goal rate.

Finally, Table 6.2 compares the APPO and BC agents. For the APPO agent, we include the results for the agent trained on 10000 training files. The expert playback row refers to

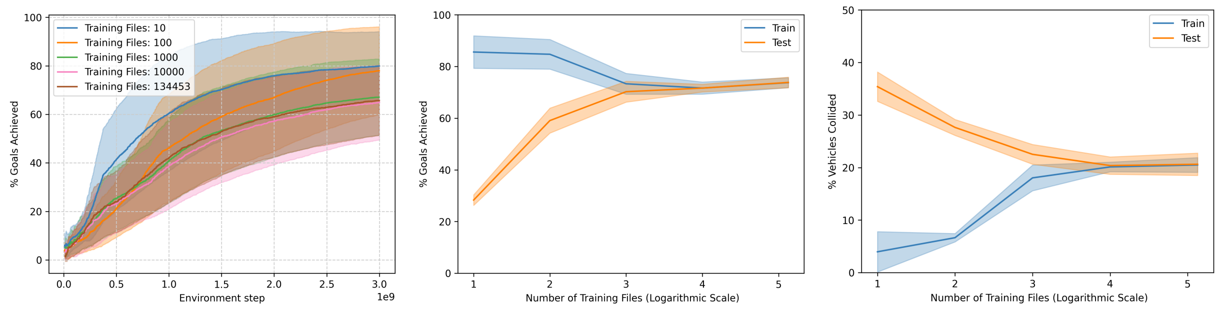


Figure 6.3: (Left) Success at getting to the specified goal on the training data as a function of number of environment steps. Files=X means the agent was trained on X fixed scenes sampled from the training dataset. (Middle) Percent of agents that achieved their goals. (Right) Percent of agents that collided. The logarithmic scale is base 10.

replay of the expert trajectories.

Table 6.2: Overview of metrics across methods for an 8 second rollout.

Algorithm	Collision Rate (%)	Goal Rate (%)	ADE (m)	FDE (m)
Expert Playback	4.9	100	0	0
APPO	20.3 ± 0.8	71.7 ± 0.7	3.1 ± 0.2	6.1 ± 0.3
BC	$38.2 \pm .1$	25.3 ± 0.1	5.6 ± 0.1	9.2 ± 0.1

Human-agent trajectory similarity

We analyze the results of our experiments with respect to how human-like the resultant policies are using displacement error between expert and agent trajectories as our metric (i.e. L2 distance between the agent and expert trajectories). To align with the definition used in other works [59, 139] we disable the removal of vehicles upon collision / reaching goal. However, we note a few dissimilarities that make comparisons with other works difficult. First, agents are provided with a goal position, a feature that is often not available to other predictive methods. Second, our experts are stepped in scenes that may contain pedestrians or cyclists but our agents are replayed in the same scene without the corresponding pedestrians or cyclists. This can make the magnitude of the displacement error not directly comparable to the values in other works; the low value of the displacement error may simply indicate that a majority of the scenes have unique optima. Fig. 6.4 examines the average difference in position between the agent and expert trajectories averaged across 5 training runs and demonstrates that the influence of more training data on displacement error is flat after 1000 files.

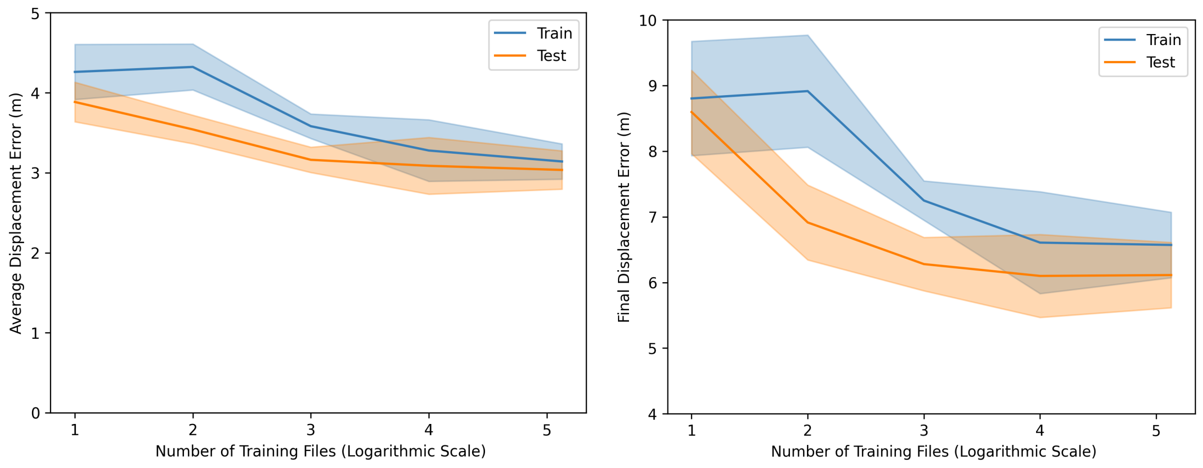


Figure 6.4: (Left) Average displacement error (mean l2-distance between an agent and an expert at each time-step). (Right) Final displacement error (l2-distance between an agent and an expert at the final time-step that an expert has a valid state). The logarithmic scale is base 10.

Policy Failure Modes

Here we investigate the mechanisms under which our policies fail to achieve their goals and collide to shine a light on potential avenues for improvement. The key failure mode we qualitatively observe are failures in scenes in which agents are required to interact with another agent either by waiting or merging. While we cannot measure interactivity directly, we measure a proxy by looking at the intersection of the expert trajectories. We play the experts forwards, record their trajectory as polylines, and consider a vehicle to have as many interactions as there are intersections of the vehicle's expert trajectory polyline with other vehicle's expert trajectory polylines (i.e. if two vehicles' trajectories in time cross at a point, that's an interaction). This captures interactions such as crossing at a four-way stop, merges, and others but also may unintentionally pick up interactions such as driving behind another vehicle. Close to 25% of vehicles have at least one interaction. The collision and goal rates as a function of interactions are plotted in Fig. 6.5 and demonstrate that the goal rate declines precipitously and the collision rate increases sharply as the number of interactions increase. This suggests that our agents have learned to get to their goals but perform poorly in settings where getting to the goal requires coordination with another agent.

Zero-shot coordination

We investigate whether the agents are learning incompatible conventions across seeds (ZSC) by taking the seeds from our best performing agent on the test set (this is the training run where we trained the agent on all files) and sampling half the agents from one seed and

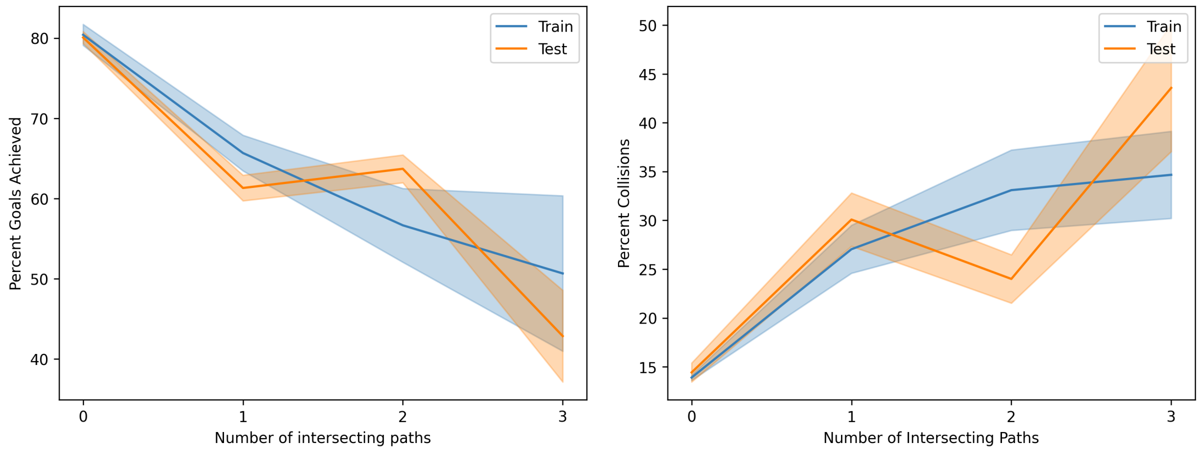


Figure 6.5: Goal rate (left) and collision rate (right) of vehicles as a function of the number of times that their corresponding expert trajectory intersected with another expert trajectory (intersections). As more than 3 interactions are rare, creating noisy statistics, the results are placed into the 3 interaction bin. The logarithmic scale is base 10.

half from another. We then perform this procedure across all 5 seeds, running each pair on the same set of 100 files. The results of this are summarized in Fig. 6.6 and appear to indicate that at the current level of agent performance there does not appear to be a zero-shot coordination issue. However, it is still possible that one might emerge as the agents become more capable and learn arbitrary symmetry-breaking conventions that are not compatible across agents. This argument lines up with the results of Sec. 6.5 which shows that the agents fail at high rates in interactive scenes; it may be necessary to get to lower failure rates in such settings for ZSC issues to emerge.

6.6 Conclusions and Future Work

We have introduced Nocturne, a simulator and benchmark intended to aid in the study of human-like decentralized coordination for driving systems. We present results on the applications of RL and imitation to this system, however, there still remains work to be done to build agents that operate with the collision and goal rate that humans achieve as well as how to learn these agents efficiently. Given better agents, human-like rules and conventions may be emergent properties of driving safely in these settings [104].

There is also ample remaining work to be done on new benchmarks. Due to the ego-centric data collection, we are forced to remove vehicles once they achieve their goal (i.e. the last observed position of the driver in the data). However, it may be possible to use generative models or other generative mechanisms to sample new goals for the agents to continue their trajectory once they achieve the goals set out in the data. Similarly, generative models could

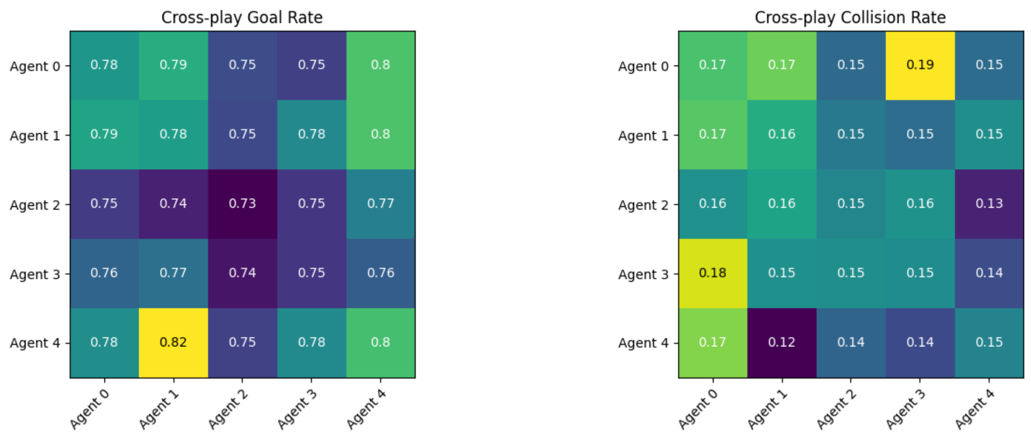


Figure 6.6: Goal rate (left) and collision rate (right) of agents when 50% are sampled from one side and 50% from another. The diagonal represents the baseline performance of the seed while element i, j represents seed i playing vs. seed j.

be used to complete the traffic light states and enable the inclusion of the traffic light scenes.

Finally, one open question is how to use Nocturne agents as predictive models of human driving. At the moment a Nocturne agent requires a goal to which it is driving. To use Nocturne agents for prediction (say for an autonomous vehicle trying to predict the motion of agents in the scene), a method for inferring goals from the 1-second context needs to be implemented. A topic for future work is to use supervised methods to predict the goals and enable fully decentralized prediction of Nocturne agents from egocentric observations.

6.7 Reproducibility and Ethical Statement

We do not release trained models as the Waymo Motion dataset restricts the release of trained models. While the dataset contains images that have been anonymized, only trajectory data is used in this work. The benchmark and all files needed to run it are publicly available at `https://github.com/facebookresearch/nocturne`.

Chapter 7

From Sim to Real: A Pipeline for Deploying Traffic Smoothing Cruise Controllers

7.1 Introduction

This chapter synthesizes the work of previous sections, modifying it to be amenable to synthesizing controllers that we are able to use in a field deployment. Prior work [132] has shown that even at current low penetration rates of less than 4%, empirical and theoretical evidence suggests that AVs can significantly reduce stop-and-go traffic, a pernicious transitory phenomenon in which vehicles alternate between starting and stopping, consuming extra fuel in the process. However, prior approaches have a unifying limitation: they are developed and analyzed in simplistic settings such as vehicles traveling around a closed ring or hand-designed input perturbations. Testing on more complex settings is difficult as: 1) real-world highway sensor data are sparse and lack required resolution and detail needed for accurate modeling; 2) developing simulators that properly reproduce emergent traffic phenomena from many-vehicle-interactions is challenging.

Even leaving aside the software engineering challenge of designing large, calibrated micro-simulations, building complex models of a highway is heavily data constrained. Loop detectors only yield macroscopic statistics such as the number of vehicles crossing them and their speed, while cameras tend to cover only a small portion of the roadway. This lack of available data is a fundamental issue as the trajectories of vehicles traveling through waves depends on the wave speed [46], and yet the wave speed and constituent frequencies are difficult to estimate with available stationary sensors. However, without an accurate means of reconstructing the stop-and-go traffic that is likely to occur on a particular highway, it is difficult to validate how a controller will perform when deployed on that highway. Consequently, it is unclear whether progress on control design for real-world smoothing is being made.

The contribution of this contribution of this chapter is a pipeline which avoids these aforementioned modeling challenges by using field-deployment data to accurately reproduce the distribution of waves observed on the I-24 in Tennessee. Instead of attempting to build a high fidelity simulation, we evaluate and train our controllers on collected highway trajectory data, ensuring that our controllers are designed to smooth a realistic representation of waves from the particular highway on which we intend to deploy AVs. We construct a simplified controller evaluation procedure in which a simulated mixed platoon of AVs and human drivers follows directly behind trajectories collected by a human driver on I-24, an interstate highway in Tennessee. We score controllers by their ability to improve energy consumption while maintaining traffic throughput. This approach sidesteps the aforementioned difficulties in calibrating both the waves and the microscopic car following dynamics. Given this simulator, we can then begin to design and evaluate controllers to test their efficacy in smoothing realistic waves.

In this chapter, we focus on designing controllers that are able to smooth waves using only local information that would be accessible via radar. Note that this precludes us from smoothing low-frequency waves that are distributed over a wide spatial distance; these waves are likely not observable via radar alone and would require the inclusion of downstream information from loop sensors or cameras. Using Proximal Policy Optimization [124], an RL policy gradient algorithm, we learn a controller that decreases the fuel consumption of the platoon in simulation by 16% for the AV and 10% on average for the platoon vehicles. Finally, we deploy the controller on real vehicles in highway traffic, showing that the resultant controllers exhibit behaviors that are robust to potential mismatches between field-deployment and our simplified simulator.

The rest of this paper is organized as follows: in Section 7.2 we discuss related work, in Section 7.3 we discuss the data collection, cleaning, and analysis, in Section 7.4 we discuss the design of the RL environment, algorithm, and training details. Section 7.5 describes the technical details required to port our controller to the physical vehicle and associated procedure for testing the controller for safety properties. In section 7.6 we discuss the simulation results, controller analysis, and experimental results from the field deployment. Finally in Section 7.7 we discuss and provide practical considerations to be considered in future work.

7.2 Related Work

Prior work has investigated the efficacy of traffic smoothing controllers on settings such as rings or hand-designed input perturbations. The most closely related works are [132, 65, 91, 62]. In [132], the authors showed that a single AV could be used to dampen stop-and-go waves on a ring with 21 human drivers, yielding sharply improved fuel efficiency. The work in [65] studies traffic smoothing with connected AVs and demonstrates that the connectivity can be used for more effective dampening of waves on a single-lane, eight-mile-long public road. The work in [91] conducted an experiment in which three control vehicles were lined up across three highway lanes and their preferred speed is selected by an external centralized controller with the goal of smoothing traffic flow. Finally, [62] uses a similar approach as this work, using the highD dataset [73] to generate realistic waves that are then dampened by an following RL-controlled vehicle. The primary distinction between this work and [62] is the use of a calibrated energy model and the physical deployment of our system onto the roadway.

Other works have considered the wave dampening properties of existing commercially-available cruise controllers, with [93],[69],[53] all observing that the vehicles they tested were string unstable (see [30] for a definition of string instability). Finally, [61] has studied some of the sim-to-real challenges in deploying RL-learned cruise controllers into more realistic settings. Prior work has also studied the use of reinforcement learning (RL) and optimal control for developing micro-level controllers that optimize mixed autonomy traffic. [154] learns memory-based policies that infer ring densities and consequently outputs near-optimal policies for the ring, [35] uses multi-agent reinforcement learning (MARL) to optimize the throughput of a merging region, and [142] employs MARL to investigate the potential impacts of altruistic autonomous driving on a merge scenario. At a network level, RL has been used to learn routing behaviors for AVs that induce the human drivers to select paths that lead to decreased congestion [77].

7.3 Training Set

Here we detail the procedure that was used to collect data from human drivers. The data serves as the basis upon which we train wave smoothing controllers. We then briefly describe the data cleaning process and analyze the distribution of trajectories collected.

Data Collection

We collect data by recording trajectory data on a 14.5-kilometer-long segment (displayed in Fig. 7.1) of the I-24 located southeast of Nashville, Tennessee. Each drive is conducted in an instrumented vehicle that logs CAN data via libpanda [23] and GPS data from an onboard receiver. Collected measurements from the vehicle CAN data include the velocity of the *ego vehicle* (the vehicle being driven), the relative velocity of the *lead vehicle* (the vehicle in front of the ego vehicle), the instantaneous acceleration, and the *space-gap* (bumper-to-bumper distance). GPS data are recorded in parallel.

The drives are varied in the time of day, day of the week, direction of travel on the highway, and level of congestion. Each drive is made up of one or more passes through the highway stretch of interest. The data used to train the algorithm in this work are made publicly available at [100], along with more details on the data. The data used in this work is a subset of the total available data at [100].

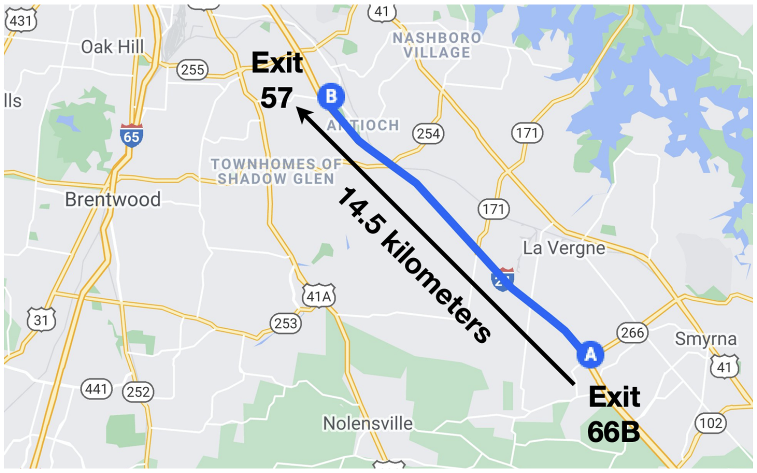


Figure 7.1: Portion of the I-24 highway on which we collected most of the dataset (section 7.3) and where we ran the experiments described in section 7.6.

Data Cleaning

The raw data for a given drive were recorded in two files: a CAN data file and a GPS file. The pertinent data are pulled from the CAN data and interpolated to the GPS time, which is measured at 10 Hz. High-frequency CAN data are down-sampled and linearly interpolated to match the GPS time, and low-frequency CAN data undergo linear interpolation to match the 10 Hz GPS time as well. Distance traveled and direction of travel are computed using the GPS position data. We observe that the westbound data contains a more regular congestion so we focus on westbound data in this work. The westbound data contain 60 trajectories, representing 8.8 hours and 772.3 kilometers of driving.

Dataset Analysis

The data are collected over a wide range of traffic conditions ranging from congested traffic that is nearly stopped to free-flow, max-speed traffic and includes many acceleration and deceleration patterns corresponding to stop-and-go traffic. Fig. 7.2 shows an example velocity and space-gap profile from a trajectory in the dataset, where we can observe the ego vehicle going quite rapidly from low to high speeds. While our main interest is in smoothing high-frequency waves, the distribution of speeds in the training dataset, shown in Fig. 7.3, tends towards higher speeds. While we could filter the dataset to only contain low speeds, likely making the learning problem simpler, Fig. 7.2 suggests that regions of congestion are often quickly followed by regions of high speed. To ensure our controller behaves appropriately at high speeds and in transitions between high and low speed regions, we keep both low and high velocities in the training dataset.

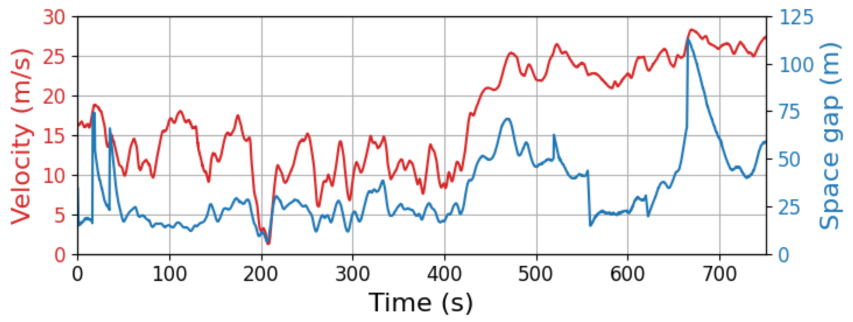


Figure 7.2: Velocity of the ego vehicle (blue) and space-gap to the lead vehicle (red) for a single trajectory in the dataset, containing sharp variations in both velocity and space-gap.

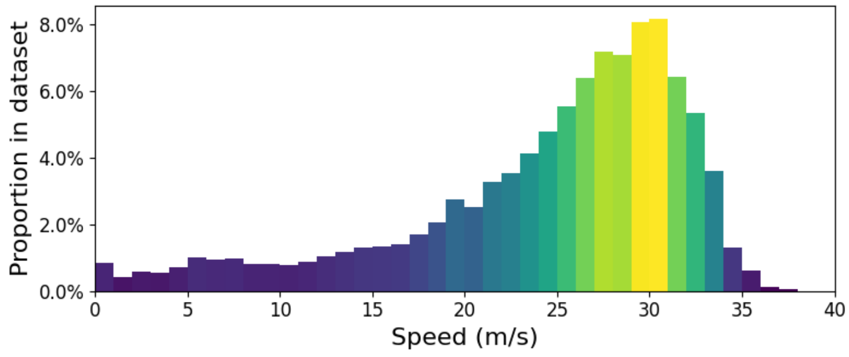


Figure 7.3: Histogram showing the distribution of velocities of the ego vehicle in the dataset.

Constructing the Training Environment

In order to use the collected data, we build a one-lane training environment where the AV follows behind the trajectory collected from the human drivers. The human driver is placed at the front of a simulated platoon, followed by the AV, followed by five vehicles driving according to the Intelligent Driver Model (IDM) [68] with a set of parameters that are string

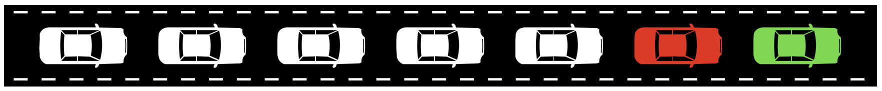


Figure 7.4: Vehicle formation used in simulation. A trajectory leader (in green) driving a speed profile drawn from the dataset is placed in front of an AV (in red) which is followed by a platoon of 5 human vehicles (in white), modeled using the Intelligent Driver Model.

unstable below $18\frac{\mathrm{m}}{\mathrm{s}}$. Although having a full micro-simulation of the I-24 would allow for training on a model with complex long-range interactions between the vehicles, the simulator proposed here allows us to train on realistic driving dynamics that are representative of both the types of waves on this highway and how drivers react to wave formation. As an additional benefit, this single-lane simulation using half a dozen vehicles achieved 2000 steps per-second while a comparable micro-simulation of the full 14 kilometer road section would have thousands of vehicles in congestion and would be very computationally costly to evaluate.

Our collected dataset contains both the trajectory of our drivers and the vehicles in front of them (via space-gap and relative velocity data logged on the CAN). We discard the lead trajectories and do not use them for simulation for two reasons. First, the lead trajectories contain both cut-ins (a vehicle cuts in between the lead vehicle and the ego driver) and cut-outs (the lead vehicle changes lanes). While cut-outs are likely unaffected by the behavior of the ego driver, cut-ins are likely a function of the spacing between ego driver and lead vehicle. Since our trained controller will have different space-gap keeping patterns, it is possible that the observed cut-outs would not occur given the controller's choice of space-gaps. Secondly, since the lane changes cause sudden variations in the spacing to the leader vehicle, it is possible for the leader vehicle to wind up behind the AV if it keeps a closer gap to the leader than the vehicle that collected the data.

Finally, in contrast to standard data science practices of splitting the data into a training set, a validation set used to tune hyperparameters, and a test set for assessing generalization, we use the entirety of our data for training. This choice was made due to the relatively small size of the collected dataset and the even smaller number of low-speed segments within the data. Instead, we view the field deployment of our controller as our measure of generalization and robustness.

7.4 Controller Design

In this section we describe the control design and structure, the details on how the controller is trained on the trajectory data, and the software deployment pipeline that is used to screen candidates before deployment on a vehicle.

Environment Structure

Due to the ability to acquire information about state solely through radar (and optionally GPS), we model our problem as a Partially Observed Markov Decision Process (POMDP) [7]. Below we describe the state and action spaces that are feasible to implement given our available sensing and actuation capabilities and our reward function.

State space: $[v, v_{\text{lead}}, h]$ where v is the AV speed, v_{lead} the speed of the vehicle right in front of it, and h the space-gap. All of these features can be acquired by using the forward-facing radar and the data collection software[23, 40] that we place on our vehicles. Note that this state is partially observed; the vehicle is not provided with information such as the state of vehicles in the platoon behind it nor information needed to predict the evolution of its leader vehicle.

Action space: an instantaneous acceleration a, bounded between $[-3.0, 1.5]\,\frac{\mathrm{m}}{\mathrm{s}^2}$, to be applied to the AV. Note that we do not allow the AVs to lane-change in this work to minimize safety considerations.

Reward function: The reward the AV receives at time-step t is a combination of minimizing energy consumption, acceleration regularization and penalties for leaving too small or too large gaps. It is given by

$$r_t = 1 - c_0 E_t - c_1 a_t^2 - c_2 P_t \ .$$

Here E_t is the instantaneous gallons of fuel consumed by the AV (given by a piece-wise polynomial energy model calibrated to a RAV-4 Toyota vehicle; the fitting procedure and function coefficients are given in [78]), a_t the AV's instantaneous acceleration in $\frac{m}{s^2}$ and P_t its gap penalty, all at time-step t. The first term is intended to discourage fuel consumption, the second to encourage smooth driving, and the third to discourage the formation of large gaps that induce cut-ins or small gaps that might be unsafe. P_t is essential as the energy-minimizing solution is to come to a full-stop; P_t both removes this solution and is used to encourage the vehicle to drive with a “sensible“ distance to its lead vehicle.

For our reward functions, we use coefficients $c_0 = 1.0\frac{1}{\mathrm{Gal}}$, $c_1 = 0.002\frac{s^2}{m}$ and $c_2 = 2$, and penalize with $P_t = 1$ when the gap is below 7m, above 120m or when the time-gap (i.e., space-gap over speed) to the leader is below 1 second. These particular values were selected via an informal hyperparameter search and found to yield improved fuel consumption of the platoon while maintaining both plausible roadway behaviors (via not opening too large a gap) and driver comfort (via not getting too close to the leader.

Termination condition and start state: Each episode is run with a fixed horizon of 1000 steps. For a leader-trajectory of length M, we sample the start-point of a trajectory uniformly from the first $M-1000$ steps. Termination of the trajectory occurs when the fixed horizon is reached.

System dynamics: The dynamics of all vehicles in the system are double integrators updated with a ballistic update [145] and a 0.1 second time-step with the exception of the lead vehicle whose position is directly updated from the pre-recorded data. The vehicles following behind the RL vehicle are updated using a Gaussian-perturbed IDM model described in Appendix Sec. 2.2.

Note that our reward function does not directly optimize the stated objective of optimizing miles-per-gallon. Unfortunately, mile-per-gallon is a quantity that can only be calculated at the completion of a trajectory and is therefore challenging to use as a reward function since RL methods struggle with infrequent rewards [5]. However, we note that for a fixed-length trajectory, subject to the constraint that the vehicle must complete the entire trajectory before the episode terminates, minimization of consumed fuel and MPG maximization are equivalent. This is simply because, in the fixed length trajectory, the numerator in the MPG term becomes a constant.

Finally, there is an incongruity between the finite-horizon objective and our infinite-horizon field deployment. The shorter horizons are used here as gradient-based controller design methods (in particular the ones used here) can struggle with long horizons due to a linear dependency of the variance of the gradient estimator on the horizon [118]. It is possible that optimizing the gallons consumed on the finite horizon generates behavior that is sub-optimal for a longer horizon.

Algorithm

We train our policy using Proximal Policy Optimization [124] (PPO), a policy gradient algorithm. We modify the standard PPO algorithm by providing the value function with a few additional inputs: the total distance traveled from start to time t, the total energy consumed by the agent at time t, and time t. The value function V^π estimates the reward-to-go from a given state s_t and a particularly controller π as

$$V^\pi(s_t) = \mathbb{E}\left[\sum_{j=t}^{T} \gamma^(j-t) r(s_j, a_j) | s_j\right]. \tag{7.1}$$

This quantity is difficult to estimate without the additional information we provide due to the partially observed state described in Sec. 7.4. The non-local information provided to the value function is used exclusively during training for variance reduction (see [124] for details on the usage of value function), and these additional inputs are neither available nor needed by the controller during evaluation.

Training was done using the PPO implementation provided in Stable Baselines 3 [112] version 1.0, a Pytorch-based deep RL library. Training details, a script to reproduce the results in the paper, and hyperparameters are provided in the linked code-base.

Software Controller Verification

Before evaluating the controller as a candidate for deployment on hardware, a transfer-learning test is conducted to assess the effectiveness of controller when exposed to varying dynamics that could potentially be experienced during the real-world test. We refer to this as the *Software-in-the-loop* (SWIL) tests. The SWIL test is intended to elucidate potential failure modes of the RL controller by running it in a different environment with out-of-distribution dynamics changes from the training environment. In particular, the SWIL test contains different sized time-steps for integrating the dynamics, unseen vehicular dynamics

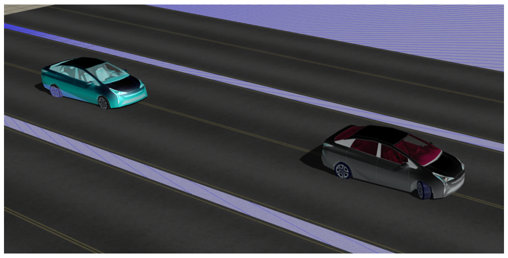


Figure 7.5: The 2-vehicle simulated scenario used for assessing the behavior of the RL controller. The lead car performs a series of abrupt starts and stops that the RL car behind it must safely respond to.

and a new set of leader trajectories. The transfer-learning environment consists of a 2-vehicle leader-follower pair (in which our controller is the follower) where both vehicles are controlled via ROS [110] and simulated in Gazebo [71]. The vehicles in Gazebo have significantly different dynamics than our simulator

An initial software-in-the-loop (SWIL) step is taken to check functional correctness and interface testing in a 2-vehicle Gazebo simulation [71] as is shown in Fig. 7.6. In this 2-vehicle scenario, structured velocity profiles (e.g., constant acceleration, sinusoidal, trapezoidal) are given as an input to the leader vehicle and the RL controller is deployed on its follower vehicle to ensure outputs are not unusual (e.g. acceleration is bounded within reasonable values, appropriate safety gaps are maintained, etc.) and yield safe behavior. An example of one of the tests, consisting of sharp starts-and-stops of the lead vehicle is depicted in Figure 7.6.

7.5 Hardware Pipeline

For hardware-in-the-loop (HWIL) deployment on the physical vehicle, there is a series of three tests to mitigate safety risks from the transition from simulation to physical vehicle before testing the controller on the I-24 segment. All three tests have varied input from the leader vehicle to ensure performance in non-equilibrium states. Such tests assess the performance for initial response, vehicle collision, and space-gap between two vehicles as the test progresses over the time.

First, the controller is tested in a 'Ghost Mode' as in [99] where the vehicle follows a simulated 'Ghost' vehicle as its leader. This provides the opportunity for a bad implementation to fail and crash into a virtual vehicle instead of a real one. The full HWIL setup is used with the modification that the real sensing done by the vehicle is replaced by a spoofed recording of a lead vehicle ahead using [40]. Second, the controller is tested in a 'CAN Coach Mode' as in [99] where the controller feedback is sent through a human-in-the-loop (HIL) for actuation. This second test occurs on a low-traffic, high speed route. Here the

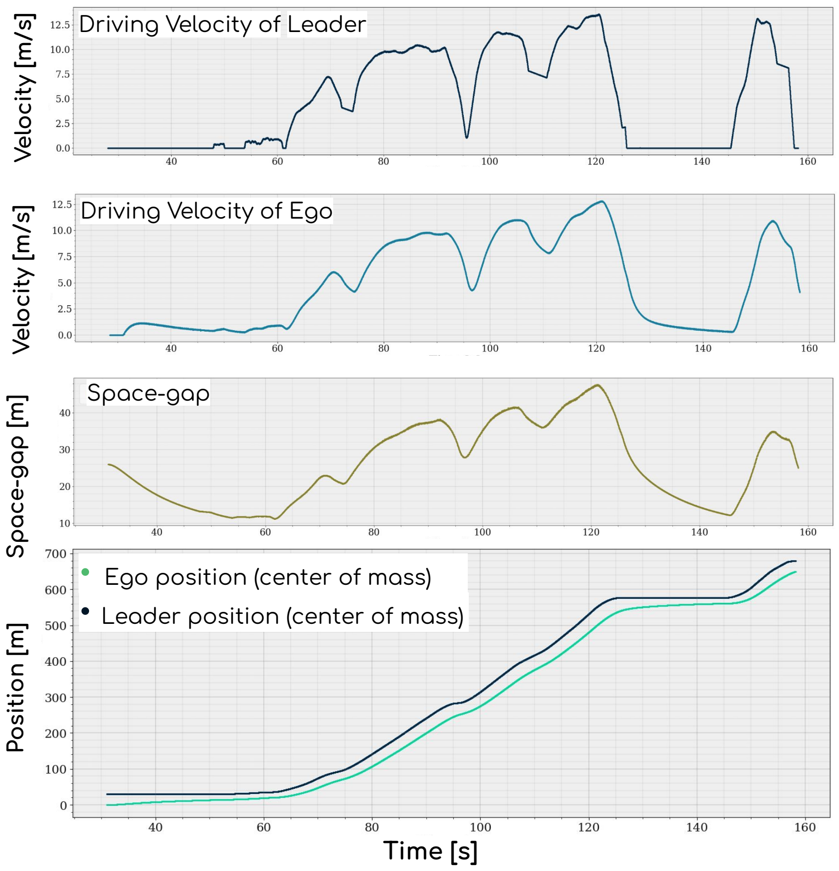


Figure 7.6: Resultant RL velocity and space-gap profiles of the RL controller when following behind a rapidly accelerating and decelerating vehicle in Gazebo.

vehicle sensors feed real-time data into the controller, and the controller outputs a desired velocity. A passenger periodically reads out the desired velocity to the driver who attempts to faithfully actuate it. If the controller provides unsafe input to the HIL it is rejected by the driver to maintain safety and replaced with human control.

Finally, the controller is used on a low-traffic, high-speed route testing the complete HWIL control loop. Once these are successfully finished, the controller is ready to be tested on the heavy traffic, high speed I-24 roadway segment.

Hardware stack

Like many modern vehicles, the Toyota RAV-4 is designed as a set of many small modules that communicate information between each other using the Controller Area Network (CAN) protocol standard. This enables the drive-by-wire, allowing operation of actuators over the CAN bus. Sensors also report information over the CAN bus. Because of this, it is possible

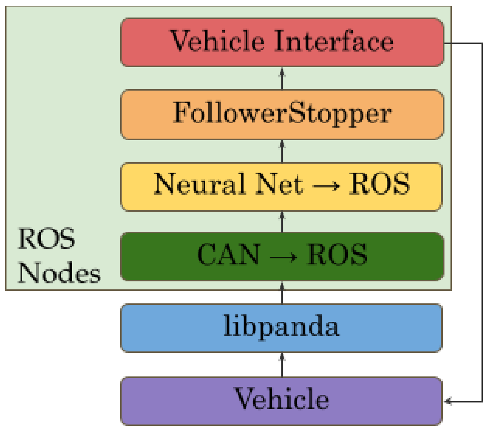


Figure 7.7: Diagram showing how information flows though the hardware-in-the-loop (HWIL) system when deployed. The vehicle sensors send data on the CAN bus. Libpanda[23] records the data and data are translated into ROS [40]. The neural net is embedded in a ROS node subscribing to pertinent CAN-to-ROS data, and its output is filtered through a supervisory FollowerStopper wrapper controller to get v_{safe}. This value is sent to the vehicle interface which takes a desired ROS command and sends it via CAN to the vehicle.

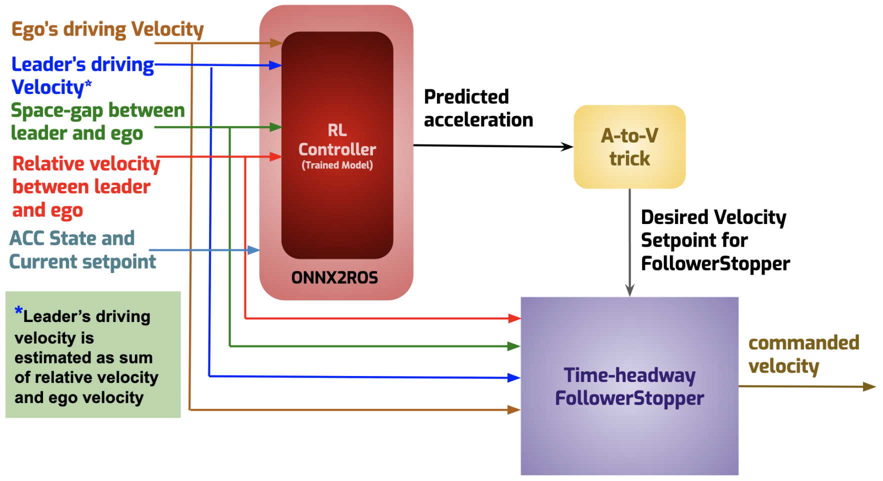

Ego's driving Velocity
Leader's driving Velocity*
Space-gap between leader and ego
Relative velocity between leader and ego
ACC State and Current setpoint
*Leader's driving velocity is estimated as sum of relative velocity and ego velocity
RL Controller (Trained Model)
ONNX2ROS
Predicted acceleration
A-to-V trick
Desired Velocity Setpoint for FollowerStopper
Time-headway FollowerStopper
commanded velocity

Figure 7.8: A schematic diagram of controller structure for traffic smoothing experiment. **w**, **th**, and **a** are parameters to time-headway based Followerstopper controller discussed in detail in [29]. ONNX2ROS framework is used to make prediction over trained model in real-time manner.

to connect third party devices to the CAN bus to both read and send information to a vehicle's modules. This concept was leveraged for these experiments to replace the vehicle's stock cruise controller, a CAN device, with our custom cruise controller.

Because CAN is standard protocol the method of parsing CAN messages is a trivial task with CAN hardware. Many companies provide hardware to interface with the CAN bus in vehicles, a common one being an OBD-II reader for vehicle diagnostics and emmision compliance. One company named comma.ai sells a device called the Panda that is a bit more invasive requiring the disconnection of some of the vehicle's CAN modules, but in doing so provides access to lower level information on the vehicle. The Panda can also intercept messages between modules and replace the messages with custom data. Such a device fits the use-case of this project, to read information on the state of the leader vehicle from the built-in radar while also allowing the operation of custom cruise controller algorithms.

Libpanda [23] was designed as a high performance library for data collection and vehicle control on devices like the Raspberry Pi. A node in the Robotics Operating System (ROS) /citeros was built on top of libpanda to expose components of the vehicle's cruise controller, allowing for acceleration requests.

Converting an acceleration controller to a velocity controller

In this section we briefly describe the procedure to convert our controller into a velocity-based controller that closely mimics the original acceleration-based controller. This step is necessary as our controller outputs an acceleration but the vehicle control stack requires velocity-based controller. We will seek to find a simple controller $v_{\text{des}}(t) = v(t) + Ta$ where T is a constant and a is the output of our controller. When fed through the control architecture, this function should ensure that the commanded acceleration closely matches our desired acceleration.

First, we fit a model of the vehicle with a first-order transfer function $\Psi(s) = \frac{1}{1+\tau s}$ using data collected from the drives. In state-space form this would correspond to the relaxation ODE $\dot{v}(t) = \frac{1}{\tau}\left(v_{\text{des}}(t) - v(t)\right)$. If we then plug the relationship $v_{\text{des}}(t) = v(t) + Ta$ into this ODE and simplify we will arrive at $\dot{v}(t) = \frac{T}{\tau}a$ so selecting $T = \tau$ gives us the desired property of our output acceleration matching out desired acceleration. Based on a fit to the original transfer function, we observed the $\tau = 0.6$ closely fits the original transfer function and so our controller is simply $v_{\text{des}}(t) = v(t) + 0.6a$. We refer to this as the *A-To-V Trick*; this is what we use to output velocity commands to the vehicle's cruise controller.

Safety Controller

At deployment time, we wrap our controller in a safety wrapper to minimize potential risks. We use the FollowerStopper from [132] with tuned coefficients to decrease the range of actions where the safety controller might override our controller. The FollowerStopper serves as a velocity controller, navigating the speed of individual vehicles to a predefined desired speed v_{des} while maintaining a safe gap with the vehicle ahead. Following this model, the command velocity v^{cmd} of an AV is defined as:

$$v^{\text{cmd}} = \begin{cases} 0 & \text{if } h_\alpha \leq \Delta x_1 \\ v\frac{\Delta x - \Delta x_1}{\Delta x_2 - \Delta x_1} & \text{if } \Delta x_1 \leq h_\alpha \leq \Delta x_2 \\ v + (v_{\text{des}} - v)\frac{\Delta x - \Delta x_2}{\Delta x_3 - \Delta x_2} & \text{if } \Delta x_2 \leq h_\alpha \leq \Delta x_3 \\ v_{\text{des}} & \text{otherwise} \end{cases} \tag{7.2}$$

where $v = \min(\max(v_l, 0), U)$ and Δx_k is defined as:

$$\Delta x_k = \Delta x_k^0 + \frac{1}{2d_k}(\Delta v_-)^2, \quad k = 1, 2, 3 \tag{7.3}$$

where $\Delta v_- = \min(\Delta v, 0)$ is the negative arm of difference between the speed of the lead vehicle and the AV. Here v_{des} is the output of the controller post *A-To-V Trick*.

Post-hoc controller modifications

We wrap our controller with a few changes to handle out-of-distribution behavior between the simulator and the field deployment test. In particular, there are several significant changes in the distribution of observed states caused by the presence of lane-changes in the field-deployment test that require careful handling.

First, there are challenges related to headways (gaps to the leader) that exceed the values observed in the simulator and are consequently out-of-distribution (OOD). As discussed in Sec. 7.4, we penalize the agent if it gets more than 120 meters from the lead vehicle. As a consequence, the AV learns to successfully stay less than 120 meters away in all the trajectories evaluated in the later stages of training. Consequently, it is possible (and we observe it to be the case) that for headways significantly above 120 meters, that the controller has unexpected and undesirable behavior. In particular, at high speeds and above 150 meters, the controller begins to slowly decelerate.

In the field-deployment, these large headways can occur via two distinct mechanisms. First, if the AV experiences several rapid cut-outs of the lead vehicle in the field deployment test, the headway can occasionally increase to large values. Additionally, the radar can occasionally not detect a lead vehicle. This occurs when either a lead vehicle is more than 250 meters away i.e. outside the range of the radar or when the vehicle is taking a sharp curve and the lead vehicle ceases to be visible. In both of these cases, the radar returns a distance of 250 meters for when the lead vehicle is missing which puts us into the range of the OOD behavior of the controller. We handle this case and ensure that the gap above 120 meters is rapidly closed by smoothly interpolating between the acceleration output by the controller and and acceleration of $0.75\frac{\text{m}}{\text{s}^2}$ using a logistic function.

However, for very large gaps, continually accelerating at $0.75\frac{\text{m}}{\text{s}^2}$ can lead to unreasonable speeds. For this reason, we cap the speed at $35\frac{\text{m}}{\text{s}}$ and apply an additional safety filter that smoothly interpolates between the desired acceleration down to an acceleration of -3.0 if

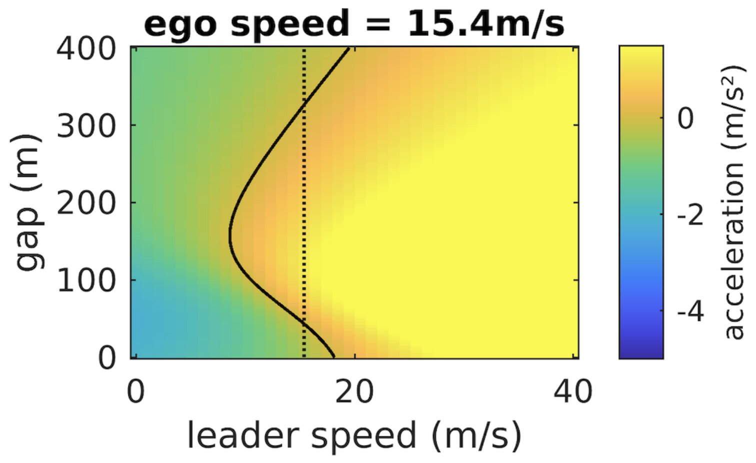


Figure 7.9: Acceleration behavior of the RL controller (ego) as a function of the leader speed and gap to the leader for a fixed speed of the RL controller. The set of equilibrium values is given by the intersection of the thick black curve and the dotted curve. Due to OOD behavior there are two unique equilibria, one of which occurs at an undesirably large gap.

the vehicle Time-To-Collision (time to close the headway to a stopped leader) falls below 6.5.

After applying all these changes, the resultant controller is the following:

$$\begin{aligned} c_h &= \frac{1}{(1.0 + \exp(-0.1 * (h_t - 120.0)))} \\ a_t &= \pi\left(s_t\right) * (1 - c_h) + c_h * 0.75 \\ \text{TTC} &= \frac{h_t}{\max(v_t - v_l,\, 0.1)} \\ c_{\text{TTC}} &= \frac{1}{(1.0 + \exp(-1.5 * (\text{TTC} - 6.5)))} \\ a_{\text{out}} &= a_t * (1 - c_{\text{TTC}}) - c_{\text{TTC}} * 3.0 \end{aligned}$$

where $\pi(s_t)$ is the controller, h_t is the headway, v_t is the ego speed and v_l is the leader speed and soft-logistics are used rather than explicit if-else statements to account for the type of operands that are supported by the tool that converts Pytorch models to ONNX modules.

Finally, we note that this controller is still wrapped in the safety controller described in Sec. 7.5; the modifications described here intended to keep the controller within the set of states that are in-distribution and override undesirable out-of-distribution behavior.

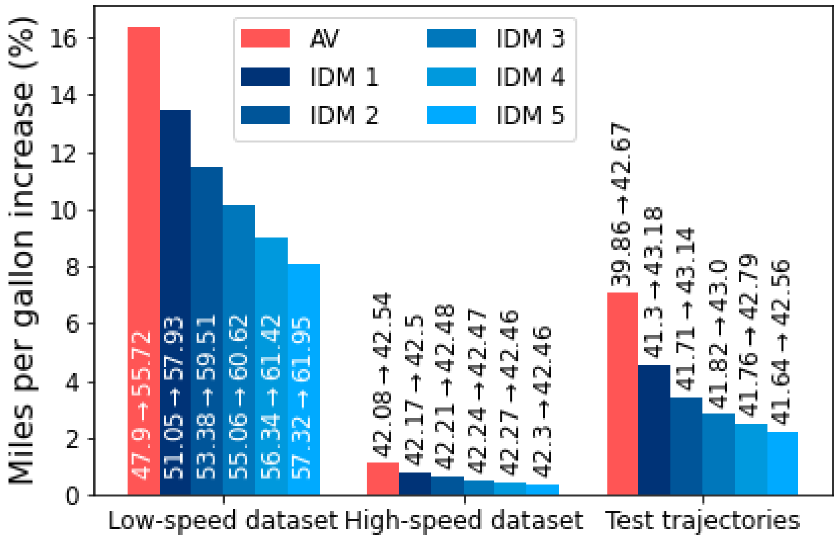


Figure 7.10: Percent improvement in MPG relative to a baseline in an IDM vehicle leads the platoon in Fig. 7.4. Each column contains both percent improvement on the y-axis and MPG values used to compute this improvement inside each column with IDM (AV) on the left (right) of the arrow. High and low speed columns are over the training set. The "Test trajectories" column is the controller evaluated on data from the physical test.

7.6 Results

Simulation Results

Here we analyze the performance of the controller in terms of energy efficiency improvements in miles per gallon (MPG) observed in our simulator. In Fig. 7.10, we compare the energy consumption of the AV and all vehicles in the platoon (as shown in Fig. 7.4) when the AV is using our RL controller compared to an IDM controller, over the whole training dataset. We split the trajectories by leader speed, computing the energy savings at leader speeds above and below $18\frac{\mathrm{m}}{\mathrm{s}}$, which is the speed boundary beyond which IDM vehicles with the parameters used in this work go from being string-unstable to string-stable. The results in the left and middle columns indicate that most of the expected energy improvements from the controller will come at low speeds. While these savings are significant, in more complex settings imperfections in actuation, modeling of human drivers, and cut-ins would likely lower the actual improvement. The rightmost column is described in Sec. 7.6.

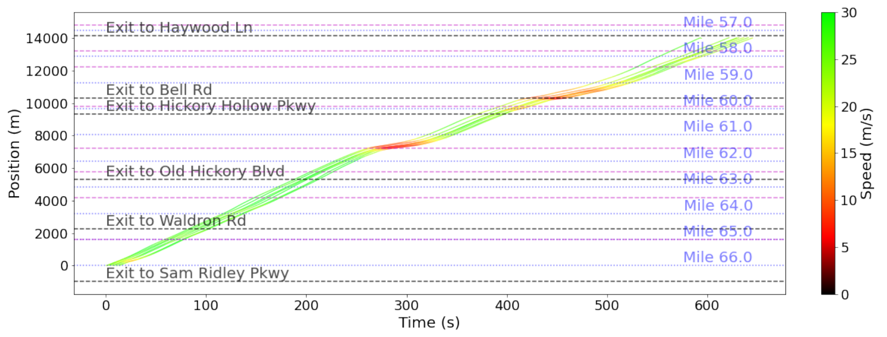


Figure 7.11: Time-space diagram showing the trajectories of our platoon of vehicle during the first test. We can observe two regions of congestion (visible in red) where the AV may have a smoothing effect.

Experimental Results

In this section, we describe the validation experiment conducted on the segment of I-24 shown in Fig. 7.1. We assess the success of the controller deployment onto AVs by showing an accurate match between simulation and reality. Finally, we seek to determine whether our controller improved the energy efficiency of its platoon.

Fig. 7.14 shows four vehicles from the eleven-vehicle platoon of alternating humans and AVs that we deployed on I-24. The vehicles are arranged with two human drivers at the front of the platoon to serve as test probes. These vehicles are unaffected by the behavior of the AVs and can serve as a proxy measure for the MPg of the unsmoothed traffic. We then alternate four AVs and human drivers going down the platoon. We chose the alternating order rather than a continious platoon as we expect the AVs to get gradually spread apart by human drivers that lane-change in. Each human-AV pair constitutes a small sub-platoon on which we can merge the influence of the AV on its following vehicles.

For each test, we got the platoon onto the highway without any non-platoon vehicles lane-changing into it. Once on the highway, non-platoon traffic cut in and out of our platoon. Since our vehicles were only instrumented to sense the vehicle in front of them, the number of vehicles that managed to enter into our platoon is unknown. We ran experiments on August 2nd, 4th, and 6th of 2021, each day launching the platoon of vehicles three times and bringing the vehicles back to the start of the highway section in between each run. Due to other controllers being tested, as well as errors in the region that the safety wrapper considered safe, the controller presented here was only actuated on 08/06, over three tests that occurred at 6:45, 7:29 and 8:36 AM. Fig. 7.11 shows individual vehicle trajectories on a time-space diagram from the 6:45 AM test; the two regions of red correspond to potential sources of congestion that the controller may have reduced.

The deployment of the controller from simulation to real vehicles was overall successful, all tests having ran safely and smoothly. We then investigate the effect of the sim-to-real

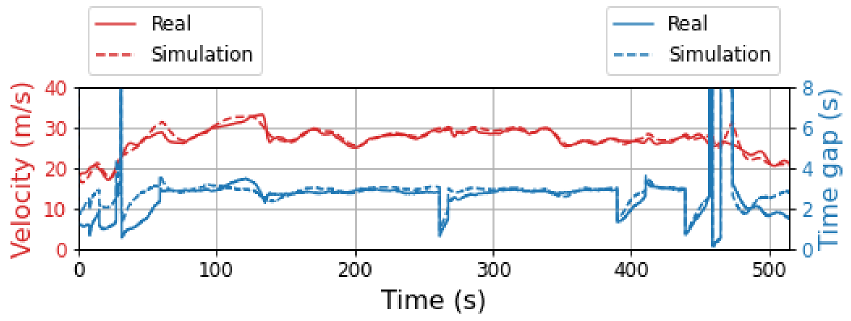


Figure 7.12: Comparison of velocity and time-gap between a real (solid) and simulated (dashed) roll-out. There are small divergences that occur around the cut-ins but the car mostly maintains a three-second time-gap in both cases.

gap induced by the presence of cut-ins and cut-outs, which we did not have when training our controller, as well as imperfect modeling of the transfer function of the AV. First, we attempt to compute a counterfactual baseline in which we replay our controller in simulation behind a trajectory collected during the tests. We note, however, that this mechanism is imperfect as the real-world trajectory has cut-ins and replaying a different controller behind it might affect the cut-in frequency. Without a model of lane changing, we cannot perform this counterfactual perfectly and so we instead make the calculations assuming that both the times when cut-ins occur and the space-gap directly after the cut-in are unchanged. Occasionally, we choose to relax this latter condition in order not to experience, in simulation, cut-ins that would be more aggressive than what the real-world AV experienced. To that end, at each time-step t where a cut-in would leave the AV with a space-gap h_t^{sim} while the real-world AV experienced a space-gap h_t^{real}, we set $h_t^{\text{sim}} = \max(h_t^{\text{sim}}, \min(h_{t-1}^{\text{sim}}, h_t^{\text{real}}))$.

Fig. 7.12 shows the velocity and time-gap (space-gap divided by velocity) of an AV from the validation experiment as well as the replay of the trajectory in our simulation using the counterfactual cut-in mechanism mentioned above. The velocity profile of the vehicle closely matches its expected behavior computed in simulation. Besides, although there are mismatches around cut-ins and cut-outs (regions where time-gap changes discontinuously), the time-gaps are relatively close and we can observe the vehicle roughly tracking a three-second time-gap in both cases. We observe similar results on the other trajectories we collected during the tests.

Finally, we analyze the potential fuel efficiency improvements from the validation experiment. The third column in Fig. 7.10 depicts the energy savings obtained when replaying in simulation using the trajectories collected during the experiments (instead of the training data which did not have lane changes) using the counterfactual cut-in mechanism mentioned earlier. We observe that the fuel efficiency of the AV has improved by 8% with additional small gains for the IDM vehicles. Fig. 7.13 shows the density of accelerations taken by the

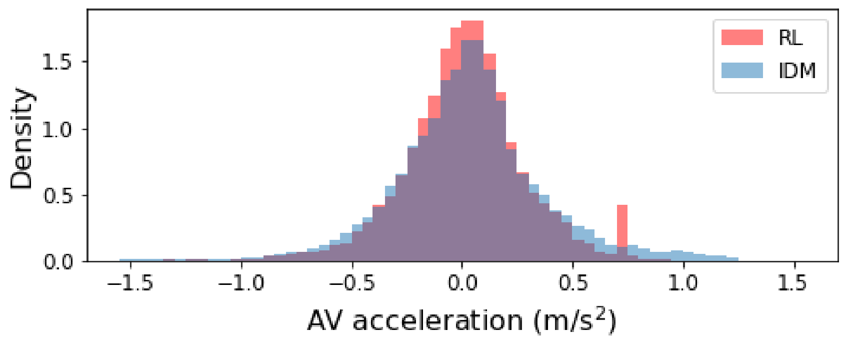


Figure 7.13: Histogram showing the density of the AV acceleration when simulating an AV or an IDM vehicle behind leader trajectories from the tests. The AV case places less mass at high, energy-consuming accelerations. The peak observed at 0.75 corresponds to situations in which the lead vehicle is out of range of radar due to a cut-out.

IDM vs. the AV; the higher density of large accelerations of the IDM vehicle are likely the reason for the improved fuel efficiency of the RL AV over the IDM AV. Unfortunately, the day of the deployment featured limited congestion so potential improvements are smaller than might be observed in heavier traffic conditions. More experimental testing on a number of days are needed to provide conclusive experimental energy savings results. Here we compute estimates from simulations using our models on experimental trajectory data.

7.7 Conclusions and Future Work

In this chapter we propose and test a pipeline that allows for effective validation and training of traffic smoothing controllers. We collect over 700 km of training data that is used to build a controller validation system. This system avoids the fundamental modeling issues that have restricted the learning or design of traffic smoothing controllers to relatively simple settings, or prevented them from deployment on real cars. In our validation system, we use Policy Gradient methods to train a controller that improves the MPG of an AV by 16% and has benefits for the following human vehicles. We then construct a pipeline for porting these controllers to four AVs and perform physical validation experiments over three days. The behavior of the vehicle on the validation experiment closely matches its expected simulation behavior, suggesting that our pipeline is an effective mechanism for validating controllers.

We observe that the main feature missing in our environment is the presence of counterfactual lane-changes. In future work, this can be addressed using the observed lane changes in the data to build a single-lane lane changing model that can be used to extend our simulation. Additional field experiments can support the assessment of our approaches in a range of traffic congestion levels.

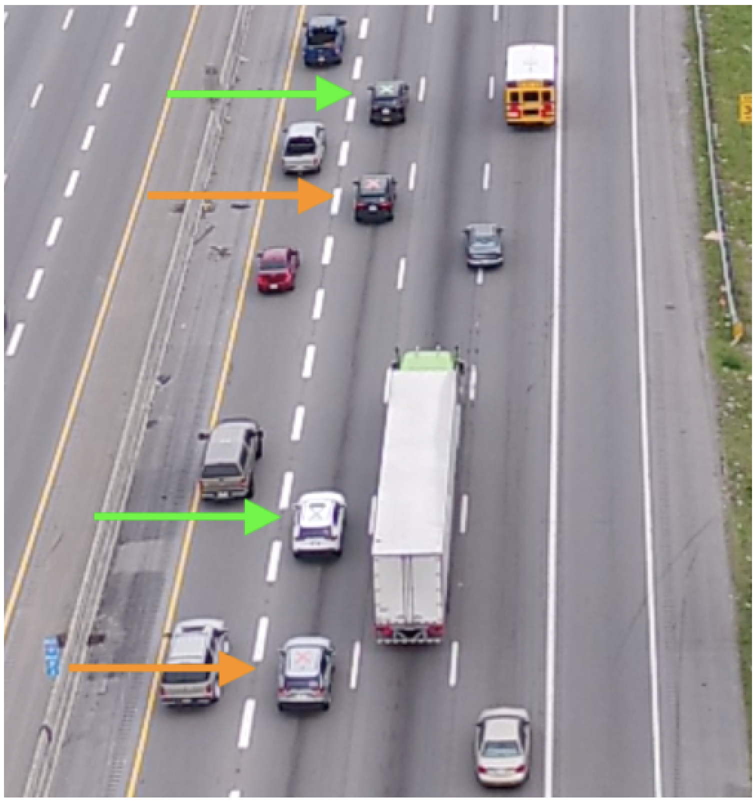


Figure 7.14: 4 of 11 vehicles in formation on the roadway. Green arrows and green X on roof indicate AV (AV), orange arrows and orange X on roof indicate human driven sensing vehicle (H). During experiments platoon formed in this order: [H, H, AV, H, AV, H, AV, H, AV, H, H], with no control over traffic flow consistently cutting in and out.

Chapter 8

Future Directions and Open Problems

In this work we have demonstrated that deep reinforcement learning can be used to design CAVs in a wide variety of complex scenarios. In simple scenarios, we were able to quickly design controllers for CAVs that sharply improved metrics such as throughput or energy consumption. As we approached more complex, multi-agent scenarios with open networks, we observed challenges with convergence and simulation speed. Finally, when moving to full-size networks we were still able to design controllers that removed stop-and-go waves but the cost of 30 or more hours of simulation time. As we moved towards a field deployment, the cost of simulation and gaps between our simulator and the real-world became increasingly critical and it became necessary to move to a smaller, fast, data-driven simulator. Using this simulator we were able to successfully field deploy four traffic-smoothing controllers and validate that our simulation strategy yielded sensible behaviors despite a potential sim-to-real gap. What remains in the short-term is to deploy this strategy at significant enough penetration rates such that wave-smoothing is both qualitatively observable and statistically observable.

Looking forward, we can draw some clear lessons and opportunities that can inform future research directions. In particular, simplifications required to complete this work expose several challenges that need to be addressed to achieve the goal of designing CAVs in large-scale simulations that can then be deployed zero-shot to the roadway. While in this work we were still able to design and deploy controllers using a single-lane simulator that captured the relevant features of stop-and-go waves, this strategy would likely not work for different phenomena that we might want to optimize such as merges, bottlenecks, or interactions between CAVs and traffic lights. We highlight a few potential areas whereby rapid progress can likely be made.

8.1 Data Opportunities

One key challenge observed in Chapter 7, inspired by the results of Chapter 5 was in getting simulated waves that even qualitatively looked like the type of waves observed on the roadway. As the characteristics of the waves cannot be sufficiently constrained using macroscopic information such as vehicle counts, microscopic data is needed for calibration. This data is increasingly available as new sources of data such as overhead cameras, dashboard cameras, data-sets from self-driving companies, and GPS traces from cloud-connected vehicles can all be combined with advances in computer vision to generate huge data-sets of vehicle trajectories. While car-following models have been a huge boon to research, their parameters are frequently calibrated using small data-sets [143] that may not correctly reproduce the full complexity of human highway driving. Converting the abundance of data into a form suitable for easy calibration using open-source object tracking and computer vision models and the development of the corresponding calibrated car-following models that spontaneously form waves is an exciting direction for future research.

8.2 Algorithmic Opportunities

On the algorithmic front, while there has been immense progress in the design of MARL algorithms capable of solving cooperative tasks like SMAC [119], Google Football [76], or Hanabi [12], there has been comparatively less attention to assessing whether these algorithms scale to settings with hundreds of agents. These settings are challenging as the joint action space blows up exponentially in the number of agents and the use of centralized methods may cause challenges with hardware as the memory requirements can be large. While there are benchmarks in this setting such as MAgent [165], standard evaluation procedures for algorithms tend towards settings that contain a smaller number of agents.

For these larger settings, one promising avenue is to develop methods that somehow “avoid“ the exponential blow-up in agent numbers by implicitly or explicitly splitting the agents into groups that have significant interactions. Such approaches might include learning or inferring a factorization of the coordination structure [82, 52], using heuristics that explicitly depend on spatial distance, or mean-field-like approaches that model the interactions between agents through a reduced set of coupling variables [136, 137]. There may be promising algorithmic advances that do not have the exponential dependence on the number of agent actions [63]. Finally, instead of using the full-scale system, once could consider transferring agents progressively from smaller to larger simulators [159, 87] in a transfer learning or curriculum learning approach.

8.3 Simulation Opportunities

As we scale to larger settings, simulation speed rapidly becomes a problem due to a near linear scaling of the simulation step-time with the number of agents in the system for many micro-simulators. In other domains, huge gains have been made by employing GPUs for simulation [92, 47]: they can both be used to accelerate the simulator and to accelerate the learning procedure. Furthermore, the co-location of the simulator and the learning algorithm onto the GPU can avoid expensive CPU-GPU transfers. While maximizing the effectiveness of GPUs for traffic simulation may be a challenging problem, prior work has suggested that there may be large speed-ups available [115]. Separately, there may be modeling strategies akin to those employed here where a reduced simulator is used that captures most of the needed effects. It is reasonable to imagine, as an extension of the work done here, a simulator that couples a single-lane simulator to a set of macroscopic variables and infers how microscopic changes in the driving behavior affect the macroscopic quantities and vice-versa. Finally, the actual process of calibrating these simulators is challenging but perhaps could be eased by replacing our simulators with differentiable simulators whose parameters could be tuned to match collected data; such an approach is widely used in rendering [101, 27] and physics simulation [8, 56]. As a side-benefit, such simulators could allow us to flexibly trade off accuracy for speed as needed for particular applications.

8.4 Looking Forwards to Large Scale Field Deployments

Finally, we briefly speculate on the challenges and possibilities of a large-scale field deployment of the controllers designed here. While we observed promising wave-smoothing behavior in simulation, wave-smoothing is difficult to observe in a multi-lane highway using a small platoon of vehicles all operating in the same lane as waves can dissipate or form for reasons unrelated to our controllers e.g. a human driver that has entered the platoon might lane-change out / in and incidentally dissipate / cause a wave. Furthermore, potential observed improvements within the lane could come at the cost of deterioration of fuel efficiency in adjacent lanes.

However, at significant penetration rates where the CAVs are distributed widely throughout the lanes, we can perform a proper comparison as we can now roughly treat the lanes as homogeneous i.e. we can assume that each lane is experiencing the same conditions and a vehicle cannot lane-change to avoid the impact of our control strategy. We look forward to such a test in 2022 where we aim to deploy 100 vehicles onto the roadway, constituting between 2-5% of the vehicles on the day of the deployment. At this scale we will be able to definitively confirm by comparing the traffic patterns of the day to historical data whether we have caused a statistically significant improvement in the miles-per-gallon of the roadway. If so, this would constitute the largest deployment of traffic-smoothing cruise control and may constitute sufficient evidence to convince existing vehicle manufacturers to adopt more energy-friendly cruise control strategies.

Looking forwards more speculatively, the work presented here suggests that there are many avenue by which CAVs can improve the roadway. Since these vehicles are already deployed, in some sense improving our highway systems is as easy as flipping a switch and uploading a new and improved set of cruise controllers onto existing platforms. Perhaps it is not too far in the future where we will see the potential benefits of CAVs widely deployed into our existing systems.

Bibliography

[1] Argonne National Laboratory (ANL). *Autonomie Compiled Vehicles.* A technical manual for training in Autonomie. 9700 S Cass Ave, Lemont, IL 60439, United States, 2020.

[2] Joshua Achiam, Ethan Knight, and Pieter Abbeel. "Towards characterizing divergence in deep q-learning". In: *arXiv preprint arXiv:1903.08894* (2019).

[3] Assad Al Alam, Ather Gattami, and Karl Henrik Johansson. "An experimental study on the fuel reduction potential of heavy duty vehicle platooning". In: *13th International IEEE Conference on Intelligent Transportation Systems.* IEEE. 2010, pp. 306–311.

[4] Saleh Albeaik et al. "Limitations and improvements of the intelligent driver model (idm)". In: *arXiv preprint arXiv:2104.02583* (2021).

[5] Marcin Andrychowicz et al. "Hindsight experience replay". In: *Advances in neural information processing systems* 30 (2017).

[6] Tyler Ard et al. "Energy and flow effects of optimal automated driving in mixed traffic: Vehicle-in-the-loop experimental results". In: *Transportation Research Part C: Emerging Technologies* 130 (2021), p. 103168.

[7] Karl J Astrom. "Optimal control of Markov decision processes with incomplete state estimation". In: *J. Math. Anal. Applic.* 10 (1965), pp. 174–205.

[8] Filipe de Avila Belbute-Peres et al. "End-to-end differentiable physics for learning and control". In: *Advances in neural information processing systems* 31 (2018).

[9] Sangjae Bae et al. "Ecological Adaptive Cruise Control of Plug-in Hybrid Electric Vehicle with Connected Infrastructure and On-Road Experiments". In: *Journal of Dynamic Systems, Measurement, and Control* (2021).

[10] Michael Bain and Claude Sammut. "A Framework for Behavioural Cloning." In: *Machine Intelligence 15.* 1995, pp. 103–129.

[11] Jaime Barceló and Jordi Casas. "Dynamic network simulation with AIMSUN". In: *Simulation approaches in transportation analysis.* Springer, 2005, pp. 57–98.

[12] Nolan Bard et al. "The hanabi challenge: A new frontier for ai research". In: *Artificial Intelligence* 280 (2020), p. 103216.

[13] Marc G Bellemare et al. "Increasing the action gap: New operators for reinforcement learning". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 30. 1. 2016.

[14] Marc G Bellemare et al. "The Arcade Learning Environment: An evaluation platform for general agents". In: *J. Artif. Intell. Res.(JAIR)* 47 (2013), pp. 253–279.

[15] Jon Louis Bentley. *Decomposable searching problems*. Tech. rep. CARNEGIE-MELLON UNIV PITTSBURGH PA DEPT OF COMPUTER SCIENCE, 1978.

[16] Carl Bergenhem et al. "Overview of platooning systems". In: *Proceedings of the 19th ITS World Congress, Oct 22-26, Vienna, Austria (2012)*. 2012.

[17] Julian Bernhard et al. "BARK: Open behavior benchmarking in multi-agent environments". In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2020, pp. 6201–6208.

[18] Daniel S Bernstein et al. "The complexity of decentralized control of Markov decision processes". In: *Mathematics of operations research* 27.4 (2002), pp. 819–840.

[19] B. Besselink and K. H. Johansson. "String stability and a delay-based spacing policy for vehicle platoons subject to disturbances". In: *IEEE Transactions on Automatic Control* (2017).

[20] Herman Bouma. "Interaction effects in parafoveal letter recognition". In: *Nature* 226.5241 (1970), pp. 177–178.

[21] Mark Brackstone and Mike McDonald. "Car-following: a historical review". In: *Transportation Research Part F: Traffic Psychology and Behaviour* 2.4 (1999), pp. 181–196.

[22] Greg Brockman et al. "Openai gym". In: *arXiv preprint arXiv:1606.01540* (2016).

[23] Matthew Bunting, Rahul Bhadani, and Jonathan Sprinkle. "Libpanda: A High Performance Library for Vehicle Data Collection". In: *Proceedings of the Workshop on Data-Driven and Intelligent Cyber-Physical Systems*. 2021, pp. 32–40.

[24] Panpan Cai et al. "Summit: A simulator for urban driving in massive mixed traffic". In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2020, pp. 4023–4029.

[25] Jordi Casas et al. "Traffic simulation with aimsun". In: *Fundamentals of traffic simulation*. Springer, 2010, pp. 173–232.

[26] Chacha Chen et al. "Toward A thousand lights: Decentralized deep reinforcement learning for large-scale traffic signal control". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 34. 04. 2020, pp. 3414–3421.

[27] Wenzheng Chen et al. "Learning to predict 3d objects with an interpolation-based differentiable renderer". In: *Advances in Neural Information Processing Systems* 32 (2019).

[28] Kyunghyun Cho et al. "Learning phrase representations using RNN encoder-decoder for statistical machine translation". In: *arXiv preprint arXiv:1406.1078* (2014).

[29] Fang-Chieh Chou et al. "Reachability Analysis for FollowerStopper: Safety Analysis and Experimental Results". In: *Proceedings of 2021 IEEE International Conference on Robotics and Automation (ICRA)*. 2021.

[30] Kai-ching Chu. "Decentralized control of high-speed vehicular strings". In: *Transportation science* 8.4 (1974), pp. 361–384.

[31] Koohong Chung, Jittichai Rudjanakanoknad, and Michael J Cassidy. "Relation between traffic density and capacity drop at three freeway bottlenecks". In: *Transportation Research Part B: Methodological* 41.1 (2007), pp. 82–95.

[32] Mladen Čičić, Li Jin, and Karl Henrik Johansson. "Coordinating Vehicle Platoons for Highway Bottleneck Decongestion and Throughput Improvement". In: *arXiv preprint arXiv:1907.13049* (2019).

[33] Mladen Čičić and Karl Henrik Johansson. "Stop-and-go wave dissipation using accumulated controlled moving bottlenecks in multi-class ctm framework". In: *2019 IEEE 58th Conference on Decision and Control (CDC)*. IEEE. 2019, pp. 3146–3151.

[34] David Crundall, Geoffrey Underwood, and Peter Chapman. "Driving experience and the functional field of view". In: *Perception* 28.9 (1999), pp. 1075–1087.

[35] Jiaxun Cui et al. "Scalable Multiagent Driving Policies For Reducing Traffic Congestion". In: *arXiv preprint arXiv:2103.00058* (2021).

[36] Shumo Cui et al. "Stabilizing traffic flow via a single autonomous vehicle: Possibilities and limitations". In: *2017 IEEE Intelligent Vehicles Symposium (IV)*. IEEE. 2017, pp. 1336–1341.

[37] Thomas A Dingus et al. *The 100-car naturalistic driving study, Phase II-results of the 100-car field experiment*. Tech. rep. 2006.

[38] Alexey Dosovitskiy et al. "CARLA: An open urban driving simulator". In: *Conference on robot learning*. PMLR. 2017, pp. 1–16.

[39] Yan Duan et al. "Benchmarking Deep Reinforcement Learning for Continuous Control". In: *CoRR* abs/1604.06778 (2016). URL: `http://arxiv.org/abs/1604.06778`.

[40] Safwan Elmadani et al. "From CAN to ROS: A Monitoring and Data Recording Bridge". In: *Proceedings of the Workshop on Data-Driven and Intelligent Cyber-Physical Systems*. 2021, pp. 17–21.

[41] Cristofer Englund, Lei Chen, and Alexey Voronov. "Cooperative speed harmonization for efficient road utilization". In: *Communication Technologies for Vehicles (Nets4Cars-Fall), 2014 7th International Workshop on*. IEEE. 2014, pp. 19–23.

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

[43] Jakob Erdmann. “SUMO’s lane-changing model”. In: *Modeling Mobility with Open Data*. Springer, 2015, pp. 105–123.

[44] Scott Ettinger et al. “Large scale interactive motion forecasting for autonomous driving: The waymo open motion dataset”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2021, pp. 9710–9719.

[45] Martin Fellendorf and Peter Vortisch. “Microscopic traffic flow simulator VISSIM”. In: *Fundamentals of traffic simulation*. Springer, 2010, pp. 63–93.

[46] Morris R Flynn et al. “Self-sustained nonlinear waves in traffic flow”. In: *Physical Review E* 79.5 (2009), p. 056113.

[47] C Daniel Freeman et al. “Brax–A Differentiable Physics Engine for Large Scale Rigid Body Simulation”. In: *arXiv preprint arXiv:2106.13281* (2021).

[48] Scott Fujimoto, Herke Van Hoof, and David Meger. “Addressing function approximation error in actor-critic methods”. In: *arXiv preprint arXiv:1802.09477* (2018).

[49] Jiyang Gao et al. “Vectornet: Encoding hd maps and agent dynamics from vectorized representation”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 11525–11533.

[50] Offer Grembek et al. “Introducing an intelligent intersection”. In: *ITS Reports* 2018.13 (2018).

[51] Yan Gu et al. “Efficient BVH construction via approximate agglomerative clustering”. In: *Proceedings of the 5th High-Performance Graphics Conference*. 2013, pp. 81–88.

[52] Carlos Guestrin, Daphne Koller, and Ronald Parr. “Multiagent planning with factored MDPs”. In: *Advances in neural information processing systems* 14 (2001).

[53] George Gunter et al. “Are commercially implemented adaptive cruise control systems string stable?” In: *IEEE Transactions on Intelligent Transportation Systems* (2020).

[54] Paul Young Joun Ha et al. “Leveraging the capabilities of connected and autonomous vehicles and multi-agent reinforcement learning to mitigate highway bottleneck congestion”. In: *arXiv preprint arXiv:2010.05436* (2020).

[55] Fred L Hall and Kwaku Agyemang-Duah. “Freeway capacity drop and the definition of capacity”. In: *Transportation research record* 1320 (1991).

[56] Eric Heiden et al. “Interactive differentiable simulation”. In: *arXiv preprint arXiv:1905.10706* (2019).

[57] Juan C Herrera et al. “Evaluation of traffic data obtained via GPS-enabled mobile phones: The Mobile Century field experiment”. In: *Transportation Research Part C: Emerging Technologies* 18.4 (2010), pp. 568–583.

[58] Hengyuan Hu et al. ““Other-Play” for Zero-Shot Coordination”. In: *International Conference on Machine Learning*. PMLR. 2020, pp. 4399–4410.

[59] Maximilian Igl et al. “Symphony: Learning Realistic and Diverse Agents for Autonomous Driving Simulation”. In: *arXiv preprint arXiv:2205.03195* (2022).

[60] Georgia-Roumpini Iordanidou et al. “Feedback-based mainstream traffic flow control for multiple bottlenecks on motorways”. In: *IEEE Transactions on Intelligent Transportation Systems* 16.2 (2014), pp. 610–621.

[61] Kathy Jang et al. “Simulation to scaled city: zero-shot policy transfer for traffic control via autonomous vehicles”. In: *Proceedings of the 10th ACM/IEEE International Conference on Cyber-Physical Systems.* ACM. 2019, pp. 291–300.

[62] Liming Jiang et al. “Reinforcement Learning based cooperative longitudinal control for reducing traffic oscillations and improving platoon stability”. In: *Transportation Research Part C: Emerging Technologies* 141 (2022), p. 103744.

[63] Chi Jin et al. “V-Learning–A Simple, Efficient, Decentralized Algorithm for Multiagent RL”. In: *arXiv preprint arXiv:2110.14555* (2021).

[64] I Ge Jin and Gábor Orosz. “Dynamics of connected vehicle systems with delayed acceleration feedback”. In: *Transportation Research Part C: Emerging Technologies* 46 (2014), pp. 46–64.

[65] I Ge Jin et al. “Experimental validation of connected automated vehicle design among human-driven vehicles”. In: *Transportation research part C: emerging technologies* 91 (2018), pp. 335–352.

[66] Li Jin and Karl Henrik Johansson. “Coordinating vehicle platoons for highway bottleneck decongestion and throughput improvement”. In: *arXiv preprint arXiv:1907.13049* (2019).

[67] Arne Kesting and Martin Treiber. “Calibrating car-following models by using trajectory data: Methodological study”. In: *Transportation Research Record* 2088.1 (2008), pp. 148–156.

[68] Arne Kesting, Martin Treiber, and Dirk Helbing. “Enhanced intelligent driver model to access the impact of driving strategies on traffic capacity”. In: *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 368.1928 (2010), pp. 4585–4605.

[69] Victor L Knoop et al. “Platoon of sae level-2 automated vehicles on public roads: Setup, traffic interactions, and stability”. In: *Transportation Research Record* 2673.9 (2019), pp. 311–322.

[70] Florian Knorr et al. “Reducing traffic jams via VANETs”. In: *IEEE Transactions on Vehicular Technology* 61.8 (2012), pp. 3490–3498.

[71] Nathan Koenig and Andrew Howard. “Design and use paradigms for gazebo, an open-source multi-robot simulator”. In: *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566).* Vol. 3. IEEE. 2004, pp. 2149–2154.

[72] Parth Kothari et al. “Drivergym: Democratising reinforcement learning for autonomous driving”. In: *arXiv preprint arXiv:2111.06889* (2021).

[73] Robert Krajewski et al. “The highd dataset: A drone dataset of naturalistic vehicle trajectories on german highways for validation of highly automated driving systems”. In: *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*. IEEE. 2018, pp. 2118–2125.

[74] Daniel Krajzewicz et al. “Recent development and applications of SUMO-Simulation of Urban MObility”. In: *International Journal On Advances in Systems and Measurements* 5.3&4 (2012).

[75] Stefan Krauß. “Microscopic modeling of traffic flow: Investigation of collision free vehicle dynamics”. PhD thesis. 1998.

[76] Karol Kurach et al. “Google research football: A novel reinforcement learning environment”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 34. 04. 2020, pp. 4501–4510.

[77] Daniel A Lazar et al. “Learning how to dynamically route autonomous vehicles on shared roads”. In: *Transportation Research Part C: Emerging Technologies* 130 (2021), p. 103258.

[78] Jonathan W. Lee et al. “Integrated Framework of Dynamics, Instabilities, Energy Models, and Sparse Flow Controllers”. In: *Proceedings of the Workshop on CPS Data for Transportation and Smart cities with Human-in-the-loop*. 2021.

[79] Joel Z Leibo et al. “Scalable evaluation of multi-agent reinforcement learning with melting pot”. In: *International Conference on Machine Learning*. PMLR. 2021, pp. 6187–6199.

[80] Edouard Leurent. *An Environment for Autonomous Driving Decision-Making*. `https://github.com/eleurent/highway-env`. 2018.

[81] Quanyi Li et al. “Metadrive: Composing diverse driving scenarios for generalizable reinforcement learning”. In: *arXiv preprint arXiv:2109.12674* (2021).

[82] Sheng Li et al. “Deep implicit coordination graphs for multi-agent reinforcement learning”. In: *arXiv preprint arXiv:2006.11438* (2020).

[83] Eric Liang et al. “Ray RLLib: A Composable and Scalable Reinforcement Learning Library”. In: *arXiv preprint arXiv:1712.09381* (2017).

[84] Eric Liang et al. *RLlib: Abstractions for Distributed Reinforcement Learning*. 2017. arXiv: `1712.09381 [cs.AI]`.

[85] Eric Liang et al. “RLlib: Abstractions for distributed reinforcement learning”. In: *International Conference on Machine Learning*. 2018, pp. 3053–3062.

[86] Timothy P Lillicrap et al. “Continuous control with deep reinforcement learning”. In: *arXiv preprint arXiv:1509.02971* (2015).

[87] Qian Long et al. “Evolutionary population curriculum for scaling multi-agent reinforcement learning”. In: *arXiv preprint arXiv:2003.10423* (2020).

[88] Pablo Alvarez Lopez et al. “Microscopic Traffic Simulation using SUMO”. In: *The 21st IEEE International Conference on Intelligent Transportation Systems*. IEEE, 2018. URL: `https://elib.dlr.de/124092/`.

[89] Ryan Lowe et al. “Multi-agent actor-critic for mixed cooperative-competitive environments”. In: *Advances in neural information processing systems*. 2017, pp. 6379–6390.

[90] Xiao-Yun Lu et al. “A new approach for combined freeway variable speed limits and coordinated ramp metering”. In: *13th International IEEE Conference on Intelligent Transportation Systems*. IEEE. 2010, pp. 491–498.

[91] Jiaqi Ma et al. “Freeway Speed Harmonization”. In: *IEEE Transactions on Intelligent Vehicles* 1.1 (2016), pp. 78–89. DOI: `10.1109/TIV.2016.2551540`.

[92] Viktor Makoviychuk et al. “Isaac gym: High performance gpu-based physics simulation for robot learning”. In: *arXiv preprint arXiv:2108.10470* (2021).

[93] Michail Makridis et al. “Empirical study on the properties of adaptive cruise control systems and their impact on traffic flow and string stability”. In: *Transportation research record* 2674.4 (2020), pp. 471–484.

[94] Horia Mania, Aurelia Guy, and Benjamin Recht. “Simple random search provides a competitive approach to reinforcement learning”. In: *arXiv preprint arXiv:1803.07055* (2018).

[95] V. Milanés et al. “Cooperative Adaptive Cruise Control in Real Traffic Situations”. In: *IEEE Transactions on Intelligent Transportation Systems* 15.1 (2014), pp. 296–305. ISSN: 1524-9050. DOI: `10.1109/TITS.2013.2278494`.

[96] Volodymyr Mnih et al. “Playing atari with deep reinforcement learning”. In: *arXiv preprint arXiv:1312.5602* (2013).

[97] Matthias Müller et al. “Sim4cv: A photo-realistic simulator for computer vision applications”. In: *International Journal of Computer Vision* 126.9 (2018), pp. 902–919.

[98] Sai Krishna Sumanth Nakka, Behdad Chalaki, and Andreas Malikopoulos. “A Multi-Agent Deep Reinforcement Learning Coordination Framework for Connected and Automated Vehicles at Merging Roadways”. In: *arXiv preprint arXiv:2109.11672* (2021).

[99] Matthew Nice et al. “CAN coach: vehicular control through human cyber-physical systems”. In: *Proceedings of the ACM/IEEE 12th International Conference on Cyber-Physical Systems*. 2021, pp. 132–142.

[100] Nice, M., Lichtle, N., Gumm, G., Roman, M., Vinitsky, E., Elmadani, S., and Bunting, M., Bhadani, R., Gunter,G., Kumar, M., and McQuade, S., Denaro, C., Delorenzo, R., Piccoli, B., Work, D. Bayen, A. Lee, J., Sprinkle, J. and Seibold, B. *The I-24 Trajectory Dataset*. `doi.org/10.5281/zenodo.6366761`. 2021.

[101] Merlin Nimier-David et al. "Mitsuba 2: A retargetable forward and inverse renderer". In: *ACM Transactions on Graphics (TOG)* 38.6 (2019), pp. 1–17.

[102] Ryosuke Nishi et al. "Theory of jam-absorption driving". In: *Transportation Research Part B: Methodological* 50 (2013), pp. 116–129.

[103] Frans A Oliehoek. "Decentralized pomdps". In: *Reinforcement Learning*. Springer, 2012, pp. 471–503.

[104] Avik Pal et al. "Emergent road rules in multi-agent driving environments". In: *arXiv preprint arXiv:2011.10753* (2020).

[105] Praveen Palanisamy. "Multi-agent connected autonomous driving using deep reinforcement learning". In: *2020 International Joint Conference on Neural Networks (IJCNN)*. IEEE. 2020, pp. 1–7.

[106] Markos Papageorgiou, Habib Hadj-Salem, Jean-Marc Blosseville, et al. "ALINEA: A local feedback control law for on-ramp metering". In: *Transportation Research Record* 1320.1 (1991), pp. 58–67.

[107] Liam Paull et al. "Duckietown: an open, inexpensive and flexible platform for autonomy education and research". In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2017, pp. 1497–1504.

[108] Aleksei Petrenko et al. "Sample factory: Egocentric 3d control from pixels at 100000 fps with asynchronous reinforcement learning". In: *International Conference on Machine Learning*. PMLR. 2020, pp. 7652–7662.

[109] Rattaphol Pueboobpaphan, Fei Liu, and Bart van Arem. "The impacts of a communication based merging assistant on traffic flows of manual and equipped vehicles at an on-ramp using traffic flow simulation". In: *Intelligent Transportation Systems (ITSC), 2010 13th International IEEE Conference on*. IEEE. 2010, pp. 1468–1473.

[110] Morgan Quigley et al. "ROS: an open-source Robot Operating System". In: *ICRA workshop on open source software*. Vol. 3. 3.2. Kobe, Japan. 2009, p. 5.

[111] Craig Quiter. *Deepdrive Zero*. Version alpha. 2020. DOI: `10.5281/zenodo.3871907`. URL: `https://doi.org/10.5281/zenodo.3871907`.

[112] Antonin Raffin et al. *Stable Baselines3*. `https://github.com/DLR-RM/stable-baselines3`. 2019.

[113] Mizanur Rahman et al. "Review of microscopic lane-changing models and future research opportunities". In: *IEEE transactions on intelligent transportation systems* 14.4 (2013), pp. 1942–1956.

[114] Aravind Rajeswaran et al. "Towards generalization and simplicity in continuous control". In: *Advances in Neural Information Processing Systems* 30 (2017).

[115] Daniel Rajf and Tomas Potuzak. "Comparison of road traffic simulation speed on CPU and GPU". In: *2019 IEEE/ACM 23rd International Symposium on Distributed Simulation and Real Time Applications (DS-RT)*. IEEE. 2019, pp. 1–8.

[116] Tabish Rashid et al. "QMIX: Monotonic value function factorisation for deep multi-agent reinforcement learning". In: *arXiv preprint arXiv:1803.11485* (2018).

[117] Meead Saberi and Hani S Mahmassani. "Empirical characterization and interpretation of hysteresis and capacity drop phenomena in freeway networks". In: *Transportation Research Record: Journal of the Transportation Research Board, Transportation Research Board of the National Academies, Washington, DC* (2013).

[118] Tim Salimans et al. "Evolution strategies as a scalable alternative to reinforcement learning". In: *arXiv preprint arXiv:1703.03864* (2017).

[119] Mikayel Samvelyan et al. "The starcraft multi-agent challenge". In: *arXiv preprint arXiv:1902.04043* (2019).

[120] Anirban Santara et al. "Madras: Multi agent driving simulator". In: *Journal of Artificial Intelligence Research* 70 (2021), pp. 1517–1555.

[121] Stefan Schaal. "Learning from demonstration". In: *Advances in neural information processing systems* 9 (1996).

[122] Tom Schaul et al. "Prioritized experience replay". In: *arXiv preprint arXiv:1511.05952* (2015).

[123] Martin Schönhof and Dirk Helbing. "Empirical features of congested traffic states and their implications for traffic modeling". In: *Transportation Science* 41.2 (2007), pp. 135–166.

[124] John Schulman et al. "Proximal policy optimization algorithms". In: *arXiv preprint arXiv:1707.06347* (2017).

[125] John Schulman et al. "Trust Region Policy Optimization". In: *ICML*. 2015, pp. 1889–1897.

[126] S. E. Shladover. "Review of the state of development of advanced vehicle control systems (AVCS)". In: *Vehicle System Dynamics* 24.6-7 (1995), pp. 551–595.

[127] Steven E Shladover, Dongyan Su, and Xiao-Yun Lu. "Impacts of cooperative adaptive cruise control on freeway traffic flow". In: *Transportation Research Record* 2324.1 (2012), pp. 63–70.

[128] Stef Smulders. "Control of freeway traffic flow by variable speed signs". In: *Transportation Research Part B: Methodological* 24.2 (1990), pp. 111–132.

[129] Anastasia D Spiliopoulou, Ioannis Papamichail, and Markos Papageorgiou. "Toll plaza merging traffic control for throughput maximization". In: *Journal of Transportation Engineering* 136.1 (2010), pp. 67–76.

[130] Thomas Spooner, Nelson Vadori, and Sumitra Ganesh. "Causal Policy Gradients: Leveraging Structure for Efficient Learning in (Factored) MOMDPs". In: *arXiv preprint arXiv:2102.10362* (2021).

[131] Raphael E Stern et al. "Dissipation of stop-and-go waves via control of autonomous vehicles: Field experiments". In: *arXiv preprint arXiv:1705.01693* (2017).

[132] Raphael E Stern et al. "Dissipation of stop-and-go waves via control of autonomous vehicles: Field experiments". In: *Transportation Research Part C: Emerging Technologies* 89 (2018), pp. 205–221.

[133] Raphael E Stern et al. "Quantifying air quality benefits resulting from few autonomous vehicles stabilizing traffic". In: *Transportation Research Part D: Transport and Environment* 67 (2019), pp. 351–365.

[134] Raphael E. Stern et al. *Data and source code.* `https://doi.org/10.15695/vudata.cee.1`. 2017.

[135] Raphael E. Stern et al. "Dissipation of stop-and-go waves via control of autonomous vehicles: Field experiments". In: *CoRR* abs/1705.01693 (2017). URL: `http://arxiv.org/abs/1705.01693`.

[136] Miguel Suau et al. "Distributed Influence-Augmented Local Simulators for Parallel MARL in Large Networked Systems". In: *arXiv preprint arXiv:2207.00288* (2022).

[137] Miguel Suau et al. "Influence-Augmented Local Simulators: A Scalable Solution for Fast Deep RL in Large Networked Systems". In: *International Conference on Machine Learning*. PMLR. 2022, pp. 20604–20624.

[138] Yuki Sugiyama et al. "Traffic jams without bottlenecks—experimental evidence for the physical mechanism of the formation of a jam". In: *New journal of physics* 10.3 (2008), p. 033001.

[139] Simon Suo et al. "Trafficsim: Learning to simulate realistic multi-agent behaviors". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021, pp. 10400–10409.

[140] D. Swaroop and J.K. Hedrick. "String stability of interconnected systems". In: *IEEE Transactions on Automatic Control* 41.3 (1996), pp. 349–357.

[141] Y. Taniguchi et al. "A Demonstration Experiment of a Theory of Jam-Absorption Driving". In: *Traffic and Granular Flow '13*. Springer International Publishing, 2015, pp. 479–483.

[142] Behrad Toghi et al. "Altruistic Maneuver Planning for Cooperative Autonomous Vehicles Using Multi-agent Advantage Actor-Critic". In: *arXiv preprint arXiv:2107.05664* (2021).

[143] US Department of Transportation. *NGSIM – Next Generation Simulation.* 2007. URL: `http://www.ngsim.fhwa.dot.gov` (visited on 12/17/2016).

[144] Martin Treiber, Ansgar Hennecke, and Dirk Helbing. "Congested traffic states in empirical observations and microscopic simulations". In: *Physical review E* 62.2 (2000), p. 1805.

[145] Martin Treiber and Venkatesan Kanagaraj. "Comparing numerical integration schemes for time-continuous car-following models". In: *Physica A: Statistical Mechanics and its Applications* 419 (2015), pp. 183–195.

[146] Martin Treiber and Arne Kesting. "Evidence of convective instability in congested traffic flow: A systematic empirical and theoretical investigation". In: *Transportation Research Part B: Methodological* 45.9 (2011), pp. 1362–1377.

[147] Martin Treiber and Arne Kesting. "The Intelligent Driver Model with Stochasticity-New Insights Into Traffic Flow Oscillations". In: *Transportation Research Procedia* 23 (2017), pp. 174–187.

[148] Martin Treiber, Arne Kesting, and Dirk Helbing. "Understanding widely scattered traffic flows, the capacity drop, and platoons as effects of variance-driven time gaps". In: *Physical review E* 74.1 (2006), p. 016123.

[149] US Department of Transportation. *NGSIM - Next Generation Simulation*. Feb. 28, 2017. URL: `https://ops.fhwa.dot.gov/trafficanalysistools/ngsim.htm`.

[150] Eugene Vinitsky et al. "Optimizing Mixed Autonomy Traffic Flow With Decentralized Autonomous Vehicles and Multi-Agent RL". In: *arXiv preprint arXiv:2011.00120* (2020).

[151] Meng Wang et al. "Connected variable speed limits control and car-following control with vehicle-infrastructure communication to resolve stop-and-go waves". In: *Journal of Intelligent Transportation Systems* 20.6 (2016), pp. 559–572.

[152] Cathy Wu et al. "Emergent Behaviors in Mixed-Autonomy Traffic". In: *Proceedings of the 1st Annual Conference on Robot Learning*. Ed. by Sergey Levine, Vincent Vanhoucke, and Ken Goldberg. Vol. 78. Proceedings of Machine Learning Research. PMLR, 2017, pp. 398–407. URL: `http://proceedings.mlr.press/v78/wu17a.html`.

[153] Cathy Wu et al. "Flow: A Modular Learning Framework for Autonomy in Traffic". In: *arXiv preprint arXiv:1710.05465* (2017).

[154] Cathy Wu et al. "Flow: A modular learning framework for mixed autonomy traffic". In: *IEEE Transactions on Robotics* (2021).

[155] Cathy Wu et al. "Framework for control and deep reinforcement learning in traffic". In: *Intelligent Transportation Systems (ITSC), 2017 IEEE 20th International Conference on*. IEEE. 2017, pp. 1–8.

[156] F. Wu et al. "Tracking vehicle trajectories and fuel rates in oscillatory traffic". In: *Transportation Research Part C: Emerging Technologies* (2017).

[157] Fangyu Wu et al. "Tracking vehicle trajectories and fuel rates in phantom traffic jams: Methodology and data". In: *Transportation Research Part C: Emerging Technologies* 99 (2019), pp. 82–109.

[158] Yuankai Wu, Huachun Tan, and Bin Ran. “Differential variable speed limits control for freeway recurrent bottlenecks via deep reinforcement learning”. In: *arXiv preprint arXiv:1810.10952* (2018).

[159] Zhongxia Yan and Cathy Wu. “Reinforcement learning for mixed autonomy intersections”. In: *2021 IEEE International Intelligent Transportation Systems Conference (ITSC)*. IEEE. 2021, pp. 2089–2094.

[160] Tsung-Yen Yang et al. “Projection-based constrained policy optimization”. In: *arXiv preprint arXiv:2010.03152* (2020).

[161] Chao Yu et al. “The Surprising Effectiveness of PPO in Cooperative, Multi-Agent Games”. In: *arXiv preprint arXiv:2103.01955* (2021).

[162] Xinshi Zang et al. “MetaLight: Value-Based Meta-Reinforcement Learning for Traffic Signal Control”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 34. 01. 2020, pp. 1153–1160.

[163] Huichu Zhang et al. “Cityflow: A multi-agent reinforcement learning environment for large scale city traffic scenario”. In: *The World Wide Web Conference*. 2019, pp. 3620–3624.

[164] Yulin Zhang et al. “Learning a Robust Multiagent Driving Policy for Traffic Congestion Reduction”. In: *arXiv preprint arXiv:2112.03759* (2021).

[165] Lianmin Zheng et al. “Magent: A many-agent reinforcement learning platform for artificial collective intelligence”. In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 32. 1. 2018.

[166] Yang Zheng, Jiawei Wang, and Keqiang Li. “Smoothing traffic flow via control of autonomous vehicles”. In: *IEEE Internet of Things Journal* 7.5 (2020), pp. 3882–3896.

[167] Ming Zhou et al. “Smarts: Scalable multi-agent reinforcement learning training school for autonomous driving”. In: *arXiv preprint arXiv:2010.09776* (2020).

Appendix A Appendix

A.1 Benchmarks for Reinforcement Learning in Mixed Autonomy Traffic

Hyperparameter search

Hyperparameter searches were conducted on the various algorithms to yield optimally performing training results. For ARS and ES, a grid search was conducted over SGD step sizes of [0.01, 0.02] and noise standard deviations of [0.01, 0.02]. The best hyperparameters are reported in Table A.1.

Table A.1: Description of controller-specific hyperparameters.

Controller	Selected Hyperparameters
ARS	SGD step size=0.02, $\sigma = 0.02$
ES	Adam step size=0.02, $\sigma = 0.02$
TRPO	Adam step size=0.01
PPO	Num epochs=10
	`merge`, `bottleneck`: λ(GAE)$= 0.97$, Adam step size=5×10^{-4}
	`grid0`: λ(GAE)$= 0.5$, Adam step size=5×10^{-5}
	`grid1`: λ(GAE)$= 0.3$, Adam step size=5×10^{-4}

a) ARS and ES: σ is the standard deviation of the noise used to perturb parameters; b) TRPO and PPO: use of actor-critic method, the former with advantages as empirical returns minus the baseline and batch gradient descent, the latter with GAE advantages and stochastic gradient descent

A.2 Optimizing Mixed Autonomy Traffic Flow With Decentralized Autonomous Vehicles and Multi-Agent Reinforcement Learning

Detailed MDP

Here we provide significantly more details about the MDP defined in Sec. 4.3. We have three possible state spaces that we investigate. The first state set we call the *radar state* as the states accessed would be readily available via an on-board radar and GPS. This is the state space that would be most easily implemented using existing technology on an autonomous vehicle. In the radar environment, the state set is:

- The speed and headway of one vehicle ahead in each of the lanes and one vehicle behind. If the vehicle is on a segment with four lanes, it will see one vehicle ahead in each of the lanes and one vehicle behind in each of the lanes. A missing vehicle is indicated as two zeros in the appropriate position. Subject to some restrictions on sensing range, this information can be acquired via radar. Refer to Fig. 4.5 for a diagrammatic description.

- The speed of the ego vehicle as well as its lane where lanes are numbered in increasing order from right to left.

- The edge number and position on the edge of the ego vehicle where the edge numbering is according to Fig. 4.2. This information would readily be available via GPS. We supplement this with an “absolute” position on the network, indicating how far the vehicle has traveled. This latter state is technically redundant and could be inferred from edge number and position.

- A counter that indicates how long the speed of the vehicle has been below 0.2 meters per second. This is used to endow the AV with a memory that allows it to track how long it has been stopped / waiting at the bottleneck entrance.

- A global time counter indicating how much time has passed.

Note that in the *radar state* that information about the bottleneck is only indirectly available; it can only examine the states of visible vehicles and use it to infer information about the bottleneck state.

The second set of states would be available given appropriate loop sensing infrastructure or a sufficient number of overhead cameras distributed throughout the bottleneck. These states endow the AV with macroscopic information about the bottleneck. We refer to this as the *aggregate state*. Here the additional states are:

- The average speed of the vehicles on edges 3, 4, and 5.

- The number of vehicles in the bottleneck.
- A global time counter indicating how much time has passed.

Finally, the final set of states we consider is a significantly pruned state set in which we have hand-picked what we believe to be a minimal set of states with which the task can be accomplished. We refer to this as the *minimal state*. This state space should yield the fastest learning due to its small size. Here the states are:

- Total distance travelled.
- The number of vehicles in the bottleneck.
- A counter that indicates how long the speed of the vehicle has been below 0.2 meters per second.
- Ego speed.
- Leader speed.
- Headway.
- The amount of time our feedback controller described in Sec. 4.2 would wait before entering the bottleneck. This state is intended to ease the learning process since initially the vehicles can simply learn to imitate this value.

From these three potential sets of states, we form three combined state spaces that we study: radar + aggregate, minimal + aggregate, and minimal alone. Each of these represents a different set of assumptions on what sensing technology will be available, as is illustrated in Fig. 4.5. We characterize the relative performance of these different state spaces in Sec. 4.4.

The action space is simply a 1-dimensional acceleration. The vehicles enter the network at 25 meters per second and roughly maintain that speed as they travel along. Since our intent is for the controllers to stop at edge three and determine the optimal time to enter the bottleneck, we want to increase the likelihood of them coming to a stop on edge three. To increase the likelihood of sharp decelerations, we bound our action space between $\frac{1}{8}[-4.5, 2.6]$ and re-scale the actions by multiplying them by eight. The neural network weight initialization scheme we use (see appendix for details) tends to output actions bounded between $[-1, 1]$ at initialization time; this re-scaling scheme makes it likelier that large decelerations are applied.

While we could include lane-changes as a possible action, we made the assumption that it was unlikely that lane-changing behavior could be a positive and would only cause the training to take longer. To prevent the vehicles from forming unusual patterns at the entrance of the bottleneck, control is only applied on edge 3 (edges numbered according to Fig. 4.2). However, states and rewards are received at every time-step and consequently actions are computed at each time-step: we simply ignore the controller output unless we are on the third edge.

We are trying to optimize throughput, so as our reward function we simply use the number of vehicles that have exited in the previous time-step as a reward

$$r_t(s_t, a_t) = n_t/N$$

where n_t is the number of vehicles that have exited in that time-step and N is a normalization term that was use to keep the cumulative reward at reasonable values. We use $N = 50$ in this work. Since the outflow is exactly the quantity we are trying to optimize, optimizing our global reward function should result in the desired improvement. This is a global reward function that is shared by every agent in the network.

However, we note a few challenges that make this a difficult reward function optimize. First, the reward is global which causes difficulties in credit assignment. Namely, it is not clear which vehicle's action contributed to the reward at any given time-step. Secondly, there is a large gap between when an action is taken and when the reward is received for that action. That is, a vehicle choosing to enter the bottleneck does not receive any reward directly attributable to that decision for upwards of 20 steps. Finally, the bottleneck being fully congested is likely a local minimum that is hard to escape. Once congestion has onset, it cannot be removed without a temporary period where the inflow into the bottleneck is reduced. However, a single vehicle choosing to not enter the bottleneck would have negligible effect on the inflow, making it difficult for vehicles to learn that decongestion is even possible.

Training parameters

Here we outline the optimal hyperparameters and seed for every experiment presented in this paper. These hyperparameters were found, as discussed in Sec. A.2, by sweeping a fixed set of hyperparameters, picking the policy with the highest reward after 2000 iterations, and then sweeping 35 seeds.

Experiment details

For the training parameters for TD3, we primarily used the default parameters set in RL-lib [84][1] version 0.8.0, a distributed deep RL library. Both the policy and the Q-function are approximated by a neural network, each with two hidden layers of size $[400, 300]$ and a ReLU non-linearity following each hidden layer. We used a training buffer size of 100000 samples and use a ratio of 5 new samples from the environment for every gradient step. For each training run, we also perform a hyperparameter sweep over the following values:

- The learning rate for both the policy and the critic: $[1e-3, 1e-4]$.

[1]`https://github.com/ray-project/ray/python/ray/rllib`

		`actor_lr`	`critic_lr`	`n_step`	`prioritzed_replay`	`seed`
minimal	5%	0.001	0.0001	5	True	24
	10%	0.0001	0.0001	5	False	29
	20%	0.0001	0.001	5	True	15
	40%	0.0001	0.0001	5	False	9
	universal	0.0001	0.0001	5	False	None
minimal + aggregate	5%	0.0001	0.0001	5	False	28
	10%	0.001	0.0001	5	True	3
	20%	0.0001	0.0001	5	False	0
	40%	0.0001	0.001	5	True	9
	universal	0.001	0.001	5	False	None
radar + aggregate	5%	0.0001	0.0001	5	True	14
	10%	0.0001	0.001	5	False	16
	20%	0.0001	0.001	5	False	17
	40%	0.0001	0.001	5	True	19
	universal	0.0001	0.0001	5	False	None
no congest number	5%	0.0001	0.001	5	False	None

Table A.2: Parameters used during training with RLlib [85]'s implementation of the TD3 algorithm for the experiments trained at penetrations of 5%, 10%, 20%, 40% or universally (cf. Sec. 4.4) for the minimal, minimal + aggregate and radar + aggregate state spaces. The experiments trained at fixed penetration have been trained for 2000 iterations with a parameter search (cf. Sec A.2) followed by a grid search on the best parameters (cf. Sec A.2), after which the best seed was kept. The universal controllers have been trained for 400, 1200 and 2000 iterations for respectively the minimal, minimal + aggregate and radar + aggregate state spaces, and no seed search was ran for these three experiments. The last line of the table refers to the experiment that was trained without macroscopic information about the bottleneck's outflow (cf. Sec 4.4); it was trained for 1600 iterations and without seed search. Only parameters that differ from the default RLlib configuration for TD3 (`https://docs.ray.io/en/releases-0.6.6/rllib-algorithms.html#deep-deterministic-policy-gradients-ddpg-td3`) are detailed here.

- The length of the reward sequence before truncating the target with the Q-function (also known as *n-step return*): $[1, 10]$
- We test both using and not using prioritized experience replay [122].

The best performing value, in terms of final converged reward, is selected from the hyperparameters, after what we run 35 random seeds using the best hyperparameters. We select the highest reward at the end of training from these random seeds.

The percentage of autonomous vehicles varies among 5%, 10%, 20% and 40%. During each training rollout, we keep a fixed inflow of 2400 vehicles per hour over the whole horizon. At each time-step, a random number of vehicles are emitted from the start edge. Thus, the number of vehicles in each platoon behind the AVs will be of variable length and it is possible that at any time-step any given lane may have zero AVs in it. To populate the simulation fully with vehicles, we allow the experiment to run uncontrolled for 300 seconds. After that, the horizon is set to 1000 more seconds.

At training time, we use the re-routing technique discussed in Sec. 4.3 where vehicles are simply placed back at the beginning of the network after exiting. However, when performing the inflow-outflow sweeps to evaluate the efficacy of the policy / generate the graphs in this paper, we turn rerouting off to ensure that our policy's performance is not dependent on the policy using the rerouting to generate some unusual behavior. To compute the outflow at a given inflow value, we run the system for 1000 seconds and compute the outflow over the last 500 seconds.

We use the traffic micro-simulator SUMO [88] for running our simulations. We use a simulation step of 0.5 seconds and a first-order Euler integration for the dynamics. While we use a relatively small time-step to maintain sensible dynamics, we use action repetition and only select a new controller action every 2.5 seconds. Each action is thus repeated five times; this approach is useful for speeding up training when the system dynamics change at a slower time-scale than the dynamics update frequency. This technique is standard when applying deep RL to Atari games [96]. Similarly, states, rewards and actions are only computed once every 5 simulation steps during both training and evaluation. Thus, running the system for 1000 seconds corresponds to 400 environment steps but to 2000 simulation steps.

It is essential to note that for the multi-agent experiments we used a *shared controller*, all of the agents operate in a decentralized fashion but share the same controller.

Reproducibility and Experimental Details

The code used to run the experiments and plot all of the figures is available at our fork of `Flow`[2].

For each example, we perform the hyperparameter sweep indicated in Sec. A.2 and train at a fixed inflow of 2400 vehicles per hour. We then take the best hyperparameter set and

[2]`https://github.com/eugenevinitsky/decentralized_bottlenecks`

re-run the experiment using 35 different seeds. The seed with the highest outflow is taken as the controller for each example.

All experiments are run on c4.8xlarge machines on AWS EC2 which have 36 virtual cores each. Since we use a single-processor implementation of TD3, both our hyperparameter sweeps and seed sweeps fit on a single machine.

Feedback Controller Sweep Parameters

For our feedback controllers, we swept the following hyperparameters in a grid:

- $n_{\text{crit}} = [6, 8, 10]$
- $K = [1, 5, 10, 20, 50]$
- $q_{\text{init}} = [200, 600, 1000, 5000, 10000]$

Empirically, we found that these were the parameters that the control scheme was most sensitive to. Whenever a vehicle enters, we set $q_0 = q_{\text{init}}$ so each vehicle is maintaining its own counter of the appropriate wait-time.